Machine Learning: Foundations, Methodologies, and Applications

Eva Bartz
Thomas Bartz-Beielstein *Editors*

Online Machine Learning

A Practical Guide with Examples in Python

Springer

Machine Learning: Foundations, Methodologies, and Applications

Series Editors

Kay Chen Tan, Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

Dacheng Tao, University of Technology, Sydney, Australia

Eva Bartz · Thomas Bartz-Beielstein
Editors

Online Machine Learning

A Practical Guide with Examples in Python

Springer

Editors
Eva Bartz
Bartz & Bartz GmbH
Gummersbach, Germany

Thomas Bartz-Beielstein
Institute for Data Science, Engineering, and Analytics
TH Köln
Gummersbach, Germany

ISSN 2730-9908 ISSN 2730-9916 (electronic)
Machine Learning: Foundations, Methodologies, and Applications
ISBN 978-981-99-7006-3 ISBN 978-981-99-7007-0 (eBook)
https://doi.org/10.1007/978-981-99-7007-0

Foreword

Do you hear the rumble of the drums? That's the world of data analytics moving towards real time. A lot of effort is being poured into turning batch data warehouses into real-time data warehouses. It seems inevitable that more advanced use cases, such as machine learning, will also move towards real time. And yet, the field of online machine learning has already existed for decades. In fact, a lot of modern deep learning is powered by online learning methods. However, online machine learning is yet to be fully appreciated. The fact a model operates online is only scratching the surface. Doing online machine learning can reap great benefits if done comprehensively and properly. But it also requires a different mental model to what most practitioners are used to.

I've been working on online machine learning for over 5 years. Admittedly, I have not observed a great shift towards online machine learning. In spite of that, I've never been more convinced online machine learning has enormous merits that are yet to be uncovered and held in high regard. I believe there are several ways for online machine learning to grow in popularity. For of all, although it's quite clear Big Tech companies are running online models in production, there are not enough public details of how they do it. Practitioners have to be convinced by real and concrete examples. Secondly, there are not enough tools and libraries that make it easy to do online machine learning, akin to what scikit-learn did for batch machine learning. This is something I tried to resolve by creating River, although there are other great tools out there, such as Vowpal Wabbit. Thirdly, there is a lack of educational material that explains how to do online machine learning.

This book is a wonderful attempt to address the third point. It covers all the standard topics of machine learning, but with an online twist. It's a great introduction to online machine learning. I hope it will inspire more people to do online machine learning, to appreciate the value of processing data online, and to do so properly. Once you have understood the concepts in this book, you will be able to view the world of machine learning through a different lens. You will be able to see the world as a stream of data, and you will be able to process it as such. You will be able to build models that learn from data as it arrives. You will be able to build models that adapt to change. You will be able to build models that are always up-to-date. You will be able to build models that are always learning. You will be able to build models that are evaluated in real time. Trust me, it's worth it.

Paris, France
July 2023

Max Halford
Head of Data at Carbonfact
and Co-creator of River

Preface

This book deals with the exciting, seminal topic of Online Machine Learning (OML). It is divided into three parts: First, we look in detail at the theoretical foundations of OML. We describe what OML is and ask how it can be compared to Batch Machine Learning (BML) and what criteria one should develop for a meaningful comparison. In the second part, we provide practical considerations, and in the third part, we substantiate them with concrete practical applications.

Why OML? Among other things, it is about the decisive time advantage. This can be months, weeks, days, hours, or even just seconds. This time advantage can arise if Artificial Intelligence (AI) can evaluate data continuously, i.e., online. It does not have to wait until a complete set of data is available, but can already use a single observation to update the model. Does OML have other advantages besides the obvious time advantage? If so, what are they? We ask, are there limitations of BML that OML overcomes? It needs to be carefully examined at what price one gets these advantages from OML. How high is the memory requirement compared to conventional methods? Memory requirements also mean financial costs, e.g., due to higher energy requirements. Is OML possibly energy-saving and thus more sustainable, i.e., Green IT? Is it possible to obtain comparably good results? Does the quality (performance) suffer, do the results become less accurate? In order to answer these questions reliably, we first give an understandable introduction to OML in the theoretical part, which is suitable for beginners as well as for advanced users. Then we justify the criteria we found for the comparability of OML and BML, namely a well-comprehensible representation of quality, time, and memory requirements. In the second part, we address the question of exactly how OML can be used in practice. We are joined by experts from the field who report on their practical experiences, e.g., requirements for official statistics. We give reasons for recommendations for the practical use of OML.

We comprehensively present the software packages currently available for OML, especially "River",[1] and offer Sequential Parameter Optimization Toolbox for River (spotRiver), a software we developed specifically for OML. We deal in detail with

[1] https://riverml.xyz/.

special problems that can occur with data streams. The central problem for data streams is drift. We deal with the explainability of AI models, interpretability, and reproducibility as required in upcoming regulations for AI systems. These aspects can contribute to higher acceptance of AI.

In the application section, we present two detailed studies, one of which uses a large dataset with one million data. We provide evidence of when OML performs better than BML. Of particular interest is the study on hyperparameter tuning of OML. Here we show how OML can perform significantly better by optimizing hyperparameters.

Notebook
Supplementary program code for the applications and examples from this book can be found in so-called "Jupyter Notebooks" in the GitHub repository https://github.com/sn-code-inside/online-machine-learning/. The notebooks are organized by chapter.

The consulting firm Bartz & Bartz GmbH[2] laid the foundation for this book when it was awarded a contract from a tender of the Federal Statistical Office of Germany in 2023.[3] The Federal Statistical Office of Germany wanted to know whether it makes sense to use OML now for the treasure trove of data and the evaluation on behalf of the public sector (see the comments in Chap. 7). The slightly sobering result of our expertise was: interesting perspectives for the future are opening up, but at the moment there is no immediate prospect of using it. In some cases, there are technical and organizational hurdles to adapting processes in such a way that the advantages of OML can really come into play. In some cases, OML processes and implementations are not yet mature enough.

The topic fascinated us so much that we decided to pursue it further. Prof. Dr. Thomas Bartz-Beielstein took the question of the practical relevance of OML with him to the TH Köln, where he continued his research in the field, which had been ongoing for years. Under his guidance, the research group at the Institute for Data Science, Engineering, and Analytics (IDE+A)[4] was able to develop software to the point where we believe we have advanced its suitability quite a bit. Thus, we have combined the expertise of Bartz & Bartz GmbH with the research at the TH Köln, which resulted in this book.

Overall, the book is equally suitable as a reference manual for experts dealing with OML, as a textbook for beginners who want to deal with OML, and as a scientific publication for scientists dealing with OML, since it reflects the latest state of research. But it can also serve as quasi-OML consulting, as decision-makers and

[2] https://bartzundbartz.de.
[3] https://destatis.de.
[4] https://www.th-koeln.de/idea.

practitioners can use our explanations to tailor OML to their needs and use it for their application, and ask whether the benefits of OML might outweigh the costs.

To name just a few examples from military and civilian practice:

- You use state-of-the-art sensor systems to predict floods. Here, faster prediction can save lives.
- You need to fend off terrorist attacks and use underwater sensors to do so. Here, it can be crucial that the AI “recognizes” more quickly whether harmless water sportsmen are involved.
- You are responsible for observing the airspace. Reconnaissance drones, for example, can be used more efficiently if they can be programmed and trained with very recent AI data evaluations.
- You must be very expeditious in adjusting the production of critical infrastructure goods, such as vaccines, protective clothing, or medical equipment. Here, it can be useful to keep the entire production process, including the raw materials to be used, as up-to-date as possible. This can be achieved by real-time evaluation and translation into requirements based on hospital bed occupancy or sick notes.
- You are a payment service provider and you need to detect fraud attempts virtually in real time.

In conclusion, we note: OML will soon become practical, it is worthwhile to get involved with it now. This book already presents some tools that will facilitate the practice of OML in the future. A promising breakthrough is to be expected, because practice shows that due to the large amounts of data that accumulate, the previous BML is no longer sufficient. OML is the solution to evaluate and process data streams in real time and to deliver results that are relevant for practice. Specifically, the book covers the following topics:

Chapter 1 describes the motivation for this book and the objective. It describes the drawbacks and limitations of BML and the need for OML. Chapter 2 gives an overview and evaluation of methods and algorithms with special focus on supervised learning (classification and regression). Chapter 3 describes procedures for drift detection. Updateability of OML procedures is discussed in Chap. 4. Chapter 5 explains procedures for the evaluation of OML methods. Chapter 6 deals with special requirements for OML. Possible OML applications are presented in Chap. 7 and evaluated by experts in official statistics. The availability of the algorithms in software packages, especially for R and Python, is presented in Chap. 8.

The computational effort required for updating the OML models, also in comparison to an algorithmically similar offline procedure (BML), is examined experimentally in Chap. 9. There, it is also discussed to what extent the model quality could be affected, especially in comparison to similar offline methods. Chapter 10 describes hyperparameter tuning for OML. Chapter 11 presents a summary and gives important recommendations for practitioners.

Gummersbach, Germany Eva Bartz
July 2023

Contents

Contributors

Eva Bartz Bartz & Bartz GmbH, Gummersbach, Germany

Thomas Bartz-Beielstein Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany

Florian Dumpert Federal Statistical Office of Germany, Wiesbaden, Germany

Lukas Hans Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany

Christian Jung SMS Group GmbH, Siegen, Germany

Steffen Moritz Federal Statistical Office of Germany, Wiesbaden, Germany

Chapter 1
Introduction: From Batch to Online Machine Learning

Thomas Bartz-Beielstein

Abstract Batch Machine Learning (BML), which is also referred to as "offline machine learning", reaches its limits when dealing with very large amounts of data. This is especially true for available memory, handling drift in data streams, and processing new, unknown data. Online Machine Learning (OML) is an alternative to BML that overcomes the limitations of BML. In this chapter, the basic terms and concepts of OML are introduced and the differences to BML are shown.

1.1 Streaming Data

The volume of data generated from various sources has increased tremendously in recent years. Technological advances have enabled the continuous collection of data. The "three Vs" (volume, velocity, and variety) were initially used as criteria to describe big data[1]: Volume here refers to the large amount of data, velocity refers to the high speed at which the data is generated, and variety refers to the large variety of data.

The data streams (streaming data) considered in this book pose an even greater challenge to Machine Learning (ML) algorithms than big data. In addition to the three big data Vs, there are other challenges that arise, in particular, from volatility and the possibility that abrupt changes ("drift") can occur.

Definition 1.1 (*Streaming-Data*) Streaming data is data that is generated in a continuous data stream. It is loosely structured, volatile, always "flowing", and contains unpredictable, sometimes abrupt, changes. Streaming data is a subset of big data with the following characteristics:

- Volume: Streaming data is generated in very large quantities.

[1]The three Vs were expanded over time by adding veracity and value to the "five Vs".

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_1

- Velocity: Streaming data is generated at a very high rate.
- Variety: Streaming data is available in very different formats. We refer to this property as “vertical variety”.
- Variability: Streaming data is structureless and varies over time. For example, drift can occur gradually or abruptly. We refer to this property as “horizontal variety”.
- Volatility: Streaming data is available only once.

Example: Streaming-Data

A great deal of data is generated during various daily transactions, such as online shopping, online banking, or online stock trading. In addition, there is sensor data, social media data, data from operational monitoring and data from the Internet of Things, to name just a few examples.

Streaming data requires real-time or near real-time analysis. Since the data stream is constantly being produced and never ends, it is not possible to store these enormous volumes of data.

Definition 1.2 (*Static Data*) By static data we mean data that have been collected at a certain point in time and are no longer changed. They are used in the field of classical ML and have the following properties:

- Volume: Static data usually have a manageable volume.
- Persistence: Static data can be retrieved as often as required. They do not change their structure.
- Structure: Static data are usually structured and are available in tabular form.

The idea for this book is based on a study conducted for the German Federal Statistical Office. The algorithms described here may also become relevant for official statistics. One of the main objectives of the Federal Statistical Office is to publish statistics at regular intervals. New data is continuously being accumulated, which has to be evaluated. The publication intervals and data volumes are still manageable, but the current trend goes towards new digital data and shorter publication cycles. The large data volumes and analysis requirements that will then exist could necessitate novel ML algorithms. This issue is explored in Sect. 7.1.

1.2 Disadvantages of Batch Learning

In this book, we distinguish between algorithms and models: Models are built using algorithms and data. Most ML algorithms use static data in three steps to build models:

1. After loading the data, the data is preprocessed.
2. Then, a model is fitted to the data. This step is also called training. During training, the data can be used multiple times.
3. Finally, the performance of the model is calculated on test data that was not used during training.

Definition 1.3 (*Batch Machine Learning*) Machine learning that (classically) uses the entire data set or large subsets of the data set (training data) to build the model is called "batch machine learning" (BML). Instances can be used multiple times. Relatively large amounts of time and memory are available. BML is also referred to as offline machine learning.

The BML reaches its limits when data streams are processed. In this case, volatile data is present that cannot be used multiple times. In addition, batch models may become outdated due to concept drift (i.e., the data distribution changes over time). This book presents approaches to solving the following problems that cannot be solved with classical batch models:

1. Large memory requirements
2. Drift
3. Unknown data
4. Accessibility of data.

These problems are described in detail in Sects. 1.2.1–1.2.4.

1.2.1 Memory Requirements

BML problems occur when the size of the data set exceeds the size of the available amount of Random Access Memory (RAM). Possible solutions are

- optimization of the data types ("sparse representations"),
- use of a partial data set ("out-of-core learning"), i.e., dividing the data into blocks or mini-batches, see Spark MLlib[2] or Dask,[3] and
- use of highly simplified models.

In these solutions, the data is fitted to the model rather than the model to the data. Therefore, the full potential of online data is not used.

1.2.2 Drift

In general, structural changes ("drift") in the data cause problems for ML algorithms.[4] For example, in energy consumption forecasting, previously known consumption

[2] https://spark.apache.org/mllib/.

[3] https://examples.dask.org/machine-learning.html.

[4] This section describes the different types of drift. The OML algorithms for drift detection and handling are described in Chap. 3.

levels are only one element needed for modeling. In practice, future demand is driven by a number of non-stationary forces such as climate variability, population growth, or by the introduction of disruptive clean energy technologies. These may require both gradual and sudden domain adjustments.

Drift causes problems for ML models because models can become outdated—they become unreliable over time because the relationships they capture are no longer valid. This leads to a decrease in the performance of these models. Therefore, prediction, classification, regression, or anomaly detection approaches should be able to detect and respond to concept deviations in a timely manner so that the model can be updated as soon as possible.

In time series applications, in many fields such as finance, e-commerce, economics, and healthcare, the statistical properties of the time series may change, rendering forecasting models useless. Although the concept of the drift problem has been well studied in the literature, surprisingly little effort has been made to solve it. We can distinguish three types of drift: feature, label, and concept drift.

In the following, (X, y) denotes a sample, where X is a set of features and y is the target variable. Features can be derived from attributes. Attributes are also referred to as independent variables, and target variables correspondingly as dependent variables. In classification problems, the target variable is a class label, in regression problems the predicted value. Often y is not only determined by X but also by a set of unknown underlying conditions. This leads to the definition of the concept:

Definition 1.4 (*Concept*) A concept is a relationship between X and y given a set of unknown constraints (a context K).

Definition 1.5 (*Feature Drift*) Feature drift describes a change in the independent variable X.

A regulatory intervention is an example of feature drift: New laws can change consumer behavior (Auffarth, 2021; Castle et al., 2021).

Definition 1.6 (*Label Drift*) Label drift is a change in the target y.

The increase in the average value of goods at retail is given here as an example of label drift.

Definition 1.7 (*Concept Drift*) Concept drift is a change in the concept, i.e., the relationship between the independent variables X and the target variable y.

ML models can not observe the underlying conditions that determine a concept and therefore have to make an assumption about which relationship applies to each sample. This is difficult when conditions change, leading to a change in the concept, which is called concept drift. The synthetically generated Friedman-Drift data set provides a vivid example of concept drift.

Definition 1.8 (*The Friedman-Drift Data Set*) Each observation in the Friedman-Drift data set consists of ten features. Each feature value is drawn equally distributed from the interval $[0, 1]$. Only the first six features, x_0 to x_5, are relevant. The dependent variable is defined by two functions that depend on whether drift is present:

$$f(x) = \begin{cases} 10\sin(\pi x_0 x_1) + 20(x_2 - 0.5)^2 + 10x_3 + 5x_4, & \text{if drift occurs} \\ 10\sin(\pi x_3 x_5) + 20(x_1 - 0.5)^2 + 10x_0 + 5x_1, & \text{otherwise.} \end{cases}$$

Note the change in active variables, e.g., from x_0 to x_3, which implements the change in concept.

The synthetically generated Friedman-Drift data set is used in Sect. 9.2.

Concept Drift in Practice

An example of concept drift may be the prediction of ozone levels (y) at a particular location based on sensor data (X). We may be able to predict y based on X, where the relationship may depend on the wind direction (the context K), which is not detected by the sensors.

A Simple Example of Concept Drift

The example shown in Fig. 1.1 illustrates how a simple concept drift occurs by combining three data sets.[5]

Abrupt and Gradual Concept Drift

The changes in data streams or concept drift patterns can be either gradual or abrupt. Abrupt changes in data streams or abrupt concept drift mean a sudden change in the properties of the data. For example, a change in mean, a change in variance, etc. It is important to recognize these changes because they have practical implications for applications in quality control, system monitoring, fault detection, and other areas.

If the changes in the distributions of the data in the data streams occur slowly but over a long period of time, then this is drawn as gradual concept drift. Gradual concept drift is relatively difficult to detect. Figure 1.2 shows the difference between gradual and abrupt drift.

In recurrent concept drift, certain features of older data streams reappear after some time.

[5] The example is based on the "Concept Drift" section in the River documentation, see https://riverml.xyz/dev/introduction/getting-started/concept-drift-detection/.

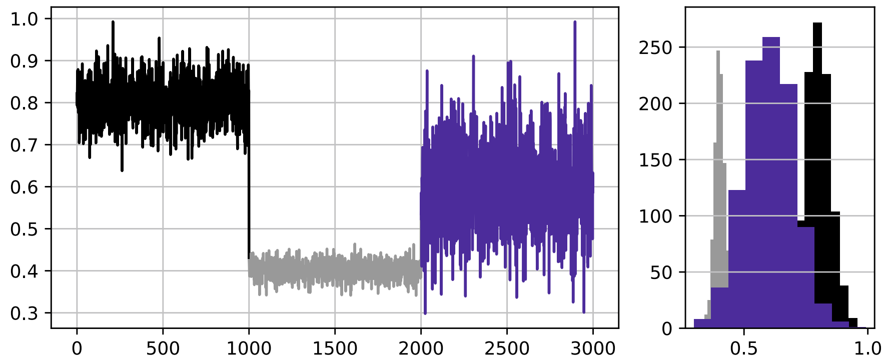


Fig. 1.1 Synthetically generated drift created by combining 1,000 data each from three different distributions. For the first thousand data, a normal distribution with mean $\mu_1 = 0.8$ and standard deviation $\sigma = 0.05$ was used. The second thousand data use $\mu_1 = 0.4$ and $\sigma_1 = 0.02$ and the last thousand data use $\mu_3 = 0.6$ and $\sigma_3 = 0.1$. The left figure shows the data over time and histograms of the three data sets are shown on the right

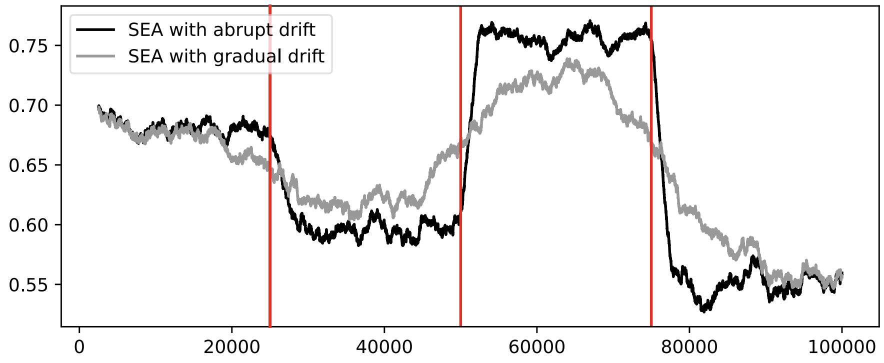


Fig. 1.2 Gradual and abrupt concept drift. The data were generated synthetically. The SEA synthetic data set (SEA) drift generator described in Sect. 5.4.2 was used for this purpose. Concept changes occurring after 25,000, 50,000, and 75,000 steps are indicated by *red* vertical lines

Drift and Non-stationarity
Please note that the term drift is often used in a broader sense for non-stationary behavior. Non-stationary behavior occurs, among other things, when new products are introduced, during hacker attacks, due to vacation periods, when weather conditions change, when the economic conditions change, or when sensors are poorly calibrated or new sensors are installed.

In BML, drift handling can be on-demand or periodic: Models for ML can be retrained regularly, i.e., at specified times (weekly) or according to specified criteria (event-based, e.g., when new data arrives) to avoid performance degradation.[6] Alternatively, training can be triggered on an as-needed basis, i.e., either based on performance monitoring of the models or based on change-detection methods.

1.2.3 New, Unknown Data

Another problem for BML is that it cannot easily learn from new data that contains unknown attributes. When unknown attributes appear in the data, the model must learn from scratch with a new data set composed of the old data and the new data. This is especially difficult in a situation where data with new attributes may occur every week, day, hour, minute, or even every time a measurement is taken.

Recommender Systems

For example, if a recommender system is to be created for an e-commerce app, then the model probably needs to be trained from scratch every week. With the popularity of the recommender service, the data set with which the model is trained grows. This leads to longer training times and may require additional hardware.

1.2.4 Accessibility and Availability of the Data

Features are generated from the attributes of the data when models are trained. Feature generation can improve ML performance by generating new features that correlate better with the target variable and are therefore easier to learn.

Definition 1.9 (*Feature Generation*) Feature generation describes the process of creating new features from one or more attributes of the data set.

Feature Generation

Feature generation can be accomplished by transforming an attribute into a new feature using a mathematical function such as the logarithmic transformation. A new feature can also be formed by calculating the distances between multiple attributes.

In practical applications, some attributes are no longer available after some time, e.g., because they have been overwritten or simply have been deleted. Thus, features

[6] This approach is implemented in the context of "mini batch machine learning", cf. Definition 1.11.

Table 1.1 Problems and solutions for BML for streaming data

Problem	BML solution	Disadvantages of solution
Memory requirements	Optimization of data types, mini-batch learning, simplified models	Performance degradation, lower accuracy
Drift	Re-training	High effort
New, unknown data	Re-training	High effort
Accessibility, availability of data	No general solution available	

that were available recently may no longer be available at the current time. In general, the provision of all data at the same time and in the same place is not always possible (Auffarth, 2021). Table 1.1 summarizes the problems and solutions for BML.

1.2.5 Other Problems

Furthermore, the common ML assumption that data is independent and uniformly distributed (IID) (stationarity) is false for most streaming data, since attributes and labels are often correlated. For example, for systems that detect attacks directed against a computer system or computer network, called intrusion detection systems, only the label "no-intrusion" occurs over a long period of time.

Furthermore, most assumptions made in the field of time series analysis do not apply to streaming data. This refers in particular to temporal correlations used to decompose time series (trend, seasonality, and residual component). Decomposition cannot be applied to streaming data because they have little structure and abrupt changes may occur.

1.3 Incremental Learning, Online Learning, and Stream Learning

The challenges of processing data streams described in Sect. 1.2 led to the development of a class of methods known as incremental or online learning methods, the development of which has been heavily promoted in recent years (Bifet, 2018; Losing et al., 2018). In particular, the development of the River[7] framework has helped incremental learning gain popularity in recent years (Montiel et al., 2021).

[7] https://riverml.xyz/.

The purpose of incremental learning is to fit an ML model into a stream of data. In this process, the data are not fully available, but observations are provided one at a time.

Since the term "online learning" is often used in the context of educational research, we use the term "online machine learning", or OML for short, in the following. Also in common use are the terms "incremental learning" and "stream learning". For the conventional ML approach, the term "batch machine learning" introduced in Definition 1.3 is used.

Definition 1.10 (*Online Machine Learning (OML)*) Online machine learning is ML that uses single instances to create and update the model. Instances can only be used once. There is relatively little time (real-time capability) and relatively little memory available.

While writing this book, it became clear that defining another class would be useful:

Definition 1.11 (*Mini-Batch Machine Learning*) Mini-batch machine learning is ML that uses subsets, so-called mini-batches, of the entire data set (training data) to repeatedly build the model. Mini-batches are usually used only once. There is relatively little time and relatively little memory available.

From the requirements described in this section, the axioms for stream learning can be derived.

Definition 1.12 (*Axioms for Stream Learning*) In the literature, e.g., Bifet (2018), the following five axioms are used:

1. Each instance can be used only once.
2. Processing time is severely limited.
3. Memory is limited.
4. The algorithm must be able to return a result at any time ("anytime property").
5. Data streams are assumed to change over time, i.e., data sources are not stationary.

Korstanje (2022) also distinguishes between "incremental learning" and "adaptive learning". Incremental learning uses models that can be updated with a single observation at a time. Adaptive learning is defined as follows:

Definition 1.13 (*Adaptive Learning*) Adaptive methods adjust the model to new situations. New trends that occur in the underlying data are taken into account.

We use the term OML for incremental and adaptive learning methods that provide methods for dealing with the axioms for stream learning presented in Definition 1.12.

1.4 Transitioning Batch to Online Machine Learning

An incremental learning algorithm can be approximated by using a batch learner with a sliding window. In this case, the model is re-trained each time a new window (of data points) arrives. Any batch learner, such as linear regression, can be applied to a data stream with a sliding window to approximate an incremental learning algorithm. Examples include the mini-batch gradient descent method and the Stochastic Gradient Descent (SGD) method.

Stochastic Gradient Descent

The gradient descent method[8] is a popular batch method to find the minimum of a function (the so-called cost or also objective function). For large data sets, a single update of the parameters takes a long time because the entire training data set is used for this purpose.

The SGD method is an iterative optimization method and can be considered as a stochastic approximation of the gradient descent method. The gradient, which is calculated from the entire data set in the gradient descent method, is replaced by an estimate that uses only a randomly selected subset of the data set. The SGD algorithm is an example of an OML algorithm that updates the model parameters at each training observation.

Notebook Example: Stochastic Gradient Descent
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning/ explains the difference between classical and stochastic gradient descent.

References

Auffarth, D. (2021). *Machine learning for time-series with Python: Forecast, predict, and detect*. Packt.

Bifet, A., et al. (2018). *Machine learning for data streams with practical examples in MOA*. MIT Press.

Castle, S., Schwarzenberg, R., & Pourvali, M. (2021). Detecting covariate drift with explanations. *Natural Language Processing and Chinese Computing: 10th CCF International Conference, NLPCC 2021, Qingdao, China, October 13–17, 2021, Proceedings, Part II* (pp. 317–322). Springer.

[8] See Definition A.1 in the Appendix.

Korstanje, J. (2022). *Machine learning for streaming data with Python*. Packt.

Losing, V., Hammer, B., & Wersing, H. (2018). Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing, 275*, 1261–1274.

Montiel, J., et al. (2021). River: Machine learning for streaming data in Python. *Journal of Machine Learning Research, 22*(1), 4945–4952. ISSN: 1532-4435.

Chapter 2
Supervised Learning: Classification and Regression

Thomas Bartz-Beielstein

Abstract This chapter provides an overview and evaluation of Online Machine Learning (OML) methods and algorithms, with a special focus on supervised learning. First, methods from the areas of classification (Sect. 2.1) and regression (Sect. 2.2) are presented. Then, ensemble methods are described in Sect. 2.3. Clustering methods are briefly mentioned in Sect. 2.4. An overview is given in Sect. 2.5.

2.1 Classification

2.1.1 Baseline Algorithms

In the area of OML classification, there are so-called "baseline algorithms" that are briefly presented here, as they serve as building blocks for more complex OML methods.

The *Majority-Class* classifier counts the occurrences of the individual classes and selects the class with the highest frequency for new instances. The *No-Change* classifier selects the last class from the data stream. The *Lazy Classifier* is a classifier that stores some already observed instances and their classes. A new instance is classified into the class of the nearest already observed instance.

Example: k-NN Classifier

Examples for lazy classifiers are k-nearest neighbor algorithms (k-NN algorithms). In k-NN, the class membership is determined based on a majority decision: The class that occurs most frequently in the neighborhood of the data point to be classified is chosen. k-NN is a "lazy classifier" because no training process is run. Only the training data set is stored. Learning only takes place when a classification is made.

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_2

2.1.2 *The Naive-Bayes Classifier*

The Naive-Bayes classifier is based on the Bayes theorem (see Theorem A.1 in the appendix). It calculates the probabilities of the individual classes based on the attributes and then selects the class with the highest probability. Since the Naive-Bayes classifier is a simple and inexpensive incremental method, it is briefly presented here. In addition, its elements play an important role in the creation of Hoeffding trees, which are presented in Sect. 2.1.3.1.

Naive-Bayes Classifier
We assume that there are k discrete attributes $x_1, x_2, \ldots, x_k$ and n_c different classes. In the following, v_j denotes the value of an attribute and c the class to which an observation belongs. The information from the training data is summarized in a table that stores a counter $n_{i,j,c}$ for each triple (x_i, v_j, c).

For example, if the observations shown in Table 2.1 are available and a new observation B with the values

$$(x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0)$$

arrives, whose class membership is to be determined, then the two probabilities

$$P(Y = 0|B) \simeq P(Y = 0)P(B|Y = 0)$$
$$P(Y = 1|B) \simeq P(Y = 1)P(B|Y = 1)$$

are calculated using Bayes' theorem. For the two classes "0" and "1", we obtain Table 2.2, the table of absolute frequencies. The Laplace correction is applied to calculate the frequencies for the classes that do not occur in the training data. The Laplace correction results from $n_{i,j,c} + 1$, i.e., the frequency for each class c is increased by 1.

After applying the Laplace correction, we obtain the values shown in Table 2.3, with which we can calculate the probabilities for $P(B|Y = 0)$ or $P(B|Y = 1)$. It holds:

Table 2.1 Labeled observations used as training data

y	x_1	x_2	x_3	x_4
1	0	1	0	1
0	1	0	0	0
1	1	1	1	1
0	1	0	0	1

$$\begin{aligned} P(B|Y=0) &= P(x_1=1, x_2=1, x_3=1, x_4=0|Y=0) \\ &= 1/2 \times 1/4 \times 1/4 \times 1/2 = 1/64 \end{aligned}$$

and thus $P(Y=0|B) = 1/2 \times 1/64 = 1/128$. In comparison,

$$\begin{aligned} P(B|Y=1) &= P(x_1=1, x_2=1, x_3=1, x_4=0|Y=1) \\ &= 1/2 \times 3/4 \times 1/2 \times 1/4 = 3/64 \end{aligned}$$

and thus $P(Y=1|B) = 1/2 \times 3/64 = 3/128$. Since $P(Y=1|B) > P(Y=0|B)$, the Naive-Bayes classifier selects the class "1" for the new observation B.

The table entries shown in Table 2.2 play an important role as statistics in trees, see Definition 2.1. They can be represented as triples (x_i, v_j, c) with values $n_{i,j,c}$. For the first entry $(x_i = 1, v_j = 0, c = 0)$ in Table 2.2b, we obtain $n_{1,0,0} = 1$. For the last entry $(x_i = 4, v_j = 1, c = 1)$ in Table 2.2b we obtain $n_{4,1,1} = 2$.

Prominent in the area of OML classification are tree-based methods (so-called "trees"), such as Hoeffding Trees (HTs) and Hoeffding Adaptive Trees (HATs). In addition to the tree-based methods presented in Sect. 2.1.3, we also present more specific methods such as Support Vector Machine (SVMs) and Passive-Aggressive (PA) in Sect. 2.1.4.

Table 2.2 Absolutely frequencies without Laplace correction

(a) Frequencies for $Y = 0$

y	x_1	x_2	x_3	x_4
0	1	2	2	1
1	1	0	0	1

(b) Frequencies for $Y = 1$

y	x_1	x_2	x_3	x_4
0	1	0	1	0
1	1	2	1	2

Table 2.3 Absolute frequencies after Laplace correction

(a) Frequencies for $Y = 0$

y	x_1	x_2	x_3	x_4
0	2	3	3	2
1	2	1	1	2

(b) Frequencies for $Y = 1$

y	x_1	x_2	x_3	x_4
0	2	1	2	1
1	2	3	2	3

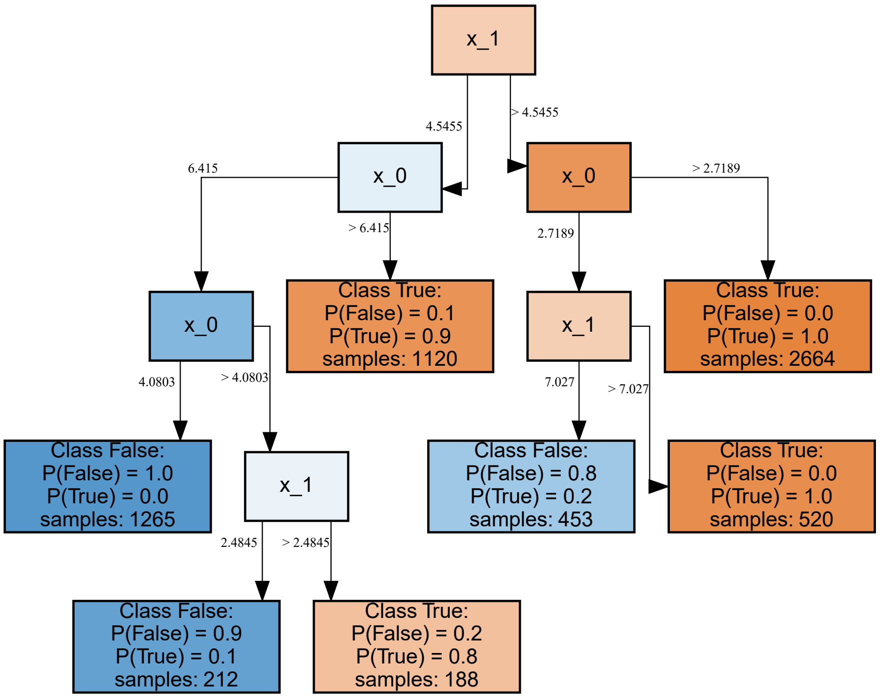


Fig. 2.1 Tree. Classification of SEA data set. The root of the tree is a node where the first test of the attributes x_1 takes place. It is tested whether x_1 is greater or less than 4.5455. The branches represent the results of the test. They lead to more nodes until the final nodes or leaves are reached. The leaves are the predictions for the classes $Y = 0$ and $Y = 1$. The color scale symbolizes the relative class frequencies in the nodes: from *dark blue* for high probability "false" to *light blue* and *light orange* to *dark orange* for high probability "true"

2.1.3 Tree-Based Methods

A challenge in processing data streams is the high memory requirement. It is impossible to store all data. Since trees allow a compact representation, they are popular methods in the area of OML. Figure 2.1 shows an example tree for the classification of the SEA synthetic data set (SEA).

Trees have the following elements:

1. Node: Testing an attribute
2. Branch: Result of the test
3. Leaf or terminal node: Prediction (of a class in classification).

We introduce two important representatives for tree-based OML methods in this section: HTs, also known as Very Fast Decision Trees (VFDTs), in Sect. 2.1.3.1 and Extremely Fast Decision Tree Classifier in Sect. 2.1.3.2.

2.1.3.1 Hoeffding-Trees

A Batch Machine Learning (BML) tree uses instances multiple times to calculate the best split attributes ("splits"). Therefore, the use of BML decision tree methods such as Classification And Regression Tree (CART) (Breiman, 1984) is not possible in a streaming-data context. Hoeffding trees are the OML counterpart to the BML trees (Domingos & Hulten, 2000). However, they do not use the instances multiple times, but work directly on the incoming instances. They thus fulfill the first axiom for stream learning (Definition 1.12).

Hoeffding trees are better suited for OML as incremental decision tree learners. They are based on the idea that a small sample is often sufficient to select an optimal split attribute. This idea is supported by the statistical result known as the Hoeffding bound, see Theorem A.2 in the appendix. This result can be simplified as shown in Example 2.1.3.1:

Example: Urn Model

An urn contains a very large number of red and black balls. We want to answer the question of whether the urn contains more black or more red balls. To do this, we draw a ball from the urn and observe its color, where the process can be repeated as often as desired.

After the process has been carried out ten times, we have obtained 4 red and 6 black balls, after one hundred attempts 47 red and 53 black balls, after one thousand attempts 501 red and 499 black balls. We can now (with a small uncertainty) say that the urn contains the same number of black and red balls, without having to draw all the balls. The probability that we are wrong is very low.

The Hoeffding bound depends on the number of observations and the permitted uncertainty. This uncertainty can be determined at the beginning using a confidence bound s.

Definition 2.1 (*Hoeffding Tree (HT)*) The Hoeffding tree stores the statistics S in each node to perform a split. For discrete attributes, this is the same information that is also used by the Naive-Bayes predictor: For each triple (x_i, v_j, c), a table with the counter $n_{i,j,c}$ of the instances with $x_i = v_j$ and for the count values $C = c$ is managed.

The HT uses two input parameters, the data stream D with labeled examples and a confidence bound s. The following code shows an algorithmic description of the HT algorithm according to Bifet (2018):

```
HoeffdingTree($D$, $s$)
1 let HT be a tree with a single leaf (root)
2 init counts $n_{i,j,c}$ at root
3 for each example $(x, y)$ in $D$
4       do HTGrow ($(x, y)$, $HT$, $s$)
```

```
HTGrow($(x, y)$, $HT$, $s$)
 1 sort $(x, y)$ to leaf $l$ using HT
 2 update counts $n_{i,j,c}$ at leaf $l$
 3 if examples seen so far at $l$ are not all of the same class
 4     then
 5         COMPUTE split gain $G$ for each attribute
 6         if $G$(best attribute) - $G$(second best)> $\sqrt{R^2 \ln(1/s)/(2n)}$
 7             then
 8                 SPLIT leaf at best attribute
 9                 for each branch
10                     do start new leaf and initialize counts
```

Hoeffding trees use the Hoeffding bound (Theorem A.2). The Hoeffding tree converges to a tree created by a BML algorithm (Bifet, 2018). However, streaming data can be very noisy. This can affect the performance (in terms of prediction accuracy or prediction error). In addition, very large trees can be generated. Hoeffding trees are also known as VFDT (Domingos & Hulten, 2000).

2.1.3.2 Extremely Fast Decision Tree Classifier

The Hoeffding Anytime Tree works similarly to a HT. The difference lies in the way a node is split. The HT delays the split of a node until the best split is identified. This decision is not re-evaluated. In contrast, the Hoeffding Anytime Tree splits at a node as soon as a useful split is available. It checks the decision for the availability of a better split. An instantiation of the Hoeffding Anytime Tree is the Extremely Fast Decision Tree (EFDT) (Manapragada et al., 2018).

2.1.4 Other Classification Methods

Logistic regression is one of the standard OML classification methods. It updates the model incrementally by taking a step towards the minimum of the cost function every time new data arrives.

From the field of SVM methods, the Approximative Large-Margin-Algorithms (ALMA) classifier is worth mentioning. This is an incremental implementation of SVMs. A detailed description can be found in Gentile (2022). The ALMA classifier is a regular SVM for (binary) classification. In Sect. 6.3.2, we will present an SVM, the so-called One-Class SVM, for anomaly detection.

The PA classifier is another well-known OML model. The term “passive-aggressive” is based on the consideration that an algorithm that learns too quickly from each new data point is considered too aggressive (Crammer et al., 2006). Therefore, PA remains passive when correctly classifying and only updates the model if a misclassification occurs (Ezukwoke & Zareian, 2021).

Notebook: Online SVM (ALMA)
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning shows how an online SVM model is implemented in Python and how the running error can be calculated incrementally.

2.2 Regression

2.2.1 Online Linear Regression

Batch-based regression models can be adapted for the online case. To implement a linear online regression model, the Stochastic Gradient Descent (SGD) is used to update the coefficients, see Sect. 1.4, since not all data is available at once. SGD is also commonly used to train neural networks.

Notebook: Online Linear Regression
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning shows how an online linear regression model is implemented in Python and how the running error can be calculated incrementally.

2.2.2 Hoeffding Tree Regressor

Further, tree-based methods are also popular for OML regression. The Hoeffding Tree Regressor (HTR) is similar to the Hoeffding Tree Classifier (HTC). It uses the Hoeffding bound to make split decisions. The HTR uses the reduction of variance in the target space to decide on the split candidates. It calculates predictions by fitting a linear perceptron model or calculating the sample average (Bifet, 2018).

2.3 Ensemble Methods for OML

Ensemble learning combines multiple models to improve the prediction accuracy for out-of-sample data. Typically (although not guaranteed), an ensemble learner performs better than the stand-alone base methods. Ensemble learning is a very popular Machine Learning (ML) technique for both regression and classification purposes. Some well-known ensemble learning techniques in the BML context are bagging, boosting, and stacking:

- "Bagging" (bootstrap aggregating) is, in simplified terms, a method that independently trains homogeneous weak learners in parallel and combines them in a deterministic averaging process.
- "Boosting" describes methods for the deterministic combination of homogeneous weak learners that are adaptively trained sequentially.
- "Stacking" describes methods that train heterogeneous weak learners in parallel and combine them into a meta-model.

These ensemble methods are also frequently used in OML. In the area of bagging, Oza Bag and Oza Bag Adwin are worth mentioning (Oza & Stuart, 2001). Random Forest methods, which can be understood as a modification of the bagging methods, are also available, e.g., Adaptive Random Forest. In the area of "boosting", OzaBoost is worth mentioning. Stacking is also used in OML. In addition, there are many other ensemble methods such as Streaming Random Patches, Voting Classifier, PA Classifier, and KNN Classifier. In Sect. 8.2, an overview and description of available software packages and their scope is presented.

2.4 Clustering

For the use case "clustering", a variety of methods are available for OML. The package `river` provides six different clustering methods: DBSTREAM, CluStream, StreamKM++, ClusTree, DenStream, and CobWeb. A detailed description can be found on the package's website.[1]

2.5 Overview: OML Methods

Table 2.4 provides an overview of the OML methods covered in this book in the area of classification and regression. In Sect. 8.2, an overview and description of available software packages and their scope is presented.

[1] https://riverml.xyz/dev/api/cluster.

Table 2.4 Overview OML methods

Method	Akronym	Task	Comments
Majority-class		Classification	Baseline algorithms
No-change		Classification	Baseline algorithms
Lazy classifier		Classification	Baseline algorithms, example: k-NN
Naive Bayes	NB	Classification	Baseline algorithms
Hoeffding tree classification/very fast decision tree	HT, HTC/VFDT	Classification	
Extremely fast decision tree classifier	EFDT	Classification	Instance of the Hoeffding anytime tree
Approximative large-margin-algorithms	ALMA	Classification	Online-SVM
Passiv-aggressive	PA	Classification	
Logistic regression		Classification	
Online linear regression		Regression	
Hoeffding tree regressor	HTR	Regression	Pendant von HTC für die Regression
Hoeffding adaptive tree regressor	HATR	Regression	Extension of HTR by drift detection using ADWIN (Sect. 3.3.3)
Adaptive random forest classifier	ARF, ARFC	Classification	Enable training of background trees that start training when a warning is detected and replace the active tree, when the warning becomes a deviation
Adaptive random forest regression	ARFR	Regression	Counterpart of ARF for regression

References

Bifet, A. (2018). *Machine learning for data streams with practical examples in MOA*. MIT Press.

Breiman, L. (1984). *Classification and regression trees*. Wadsworth.

Crammer, K., et al. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research, 7*(19), 551–585.

Domingos, P. M., & Hulten, G. (2000). Mining high-speed data streams. In R. Ramakrishnan et al. (Eds.), *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, USA*, August 20–23 (pp. 71–80). ACM.

Ezukwoke, K. I., & Zareian, S. J. (2021). Online learning and active learning: A comparative study of passive-aggressive algorithm with support vector machine (SVM). *Journal of Higher Education Theory and Practice, 21*(3), 105–110.

Gentile, C. (2002). A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research, 2*, 213–242.

Manapragada, C., Webb, G. I., & Salehi, M. (2018). Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'18, New York, NY, USA* (pp. 1953–1962). Association for Computing Machinery.

Oza, N. C., & Russell, S. (2001). Online bagging and boosting. In T. Jaakola & T. Richardson (Eds.), *8th International Workshop on Artificial Intelligence and Statistics* (pp. 105–112).

Chapter 3
Drift Detection and Handling

Thomas Bartz-Beielstein and Lukas Hans

Abstract Structural changes ("drift") in the data cause problems for many algorithms. Based on the drift definitions given in Chap. 1, methods for drift detection and handling are discussed. For the algorithms presented in Chap. 2, it is clarified to what extent concept drift is reacted to. In turn, the extent to which catastrophic forgetting is an issue is described in Sect. 4.3. Section 3.1 describes three architectures for implementing drift detection algorithms. Basic properties of window-based approaches are presented in Sect. 3.2. Section 3.4 presents commonly used drift detection techniques. Section 3.4 describes how the drift detection techniques introduced in Sect. 3.3 are used in Online Machine Learning (OML) algorithms and summarizes the tree-based OML techniques implemented in the River package. Section 3.5 introduces scaling methods for handling drift.

3.1 Architectures for Drift Detection Methods

Bifet et al. (2018) distinguished three architectures for drift detection methods: adaptive estimators, change detectors, and ensemble-based approaches.

3.1.1 Adaptive Estimators

Adaptive estimators are based on the idea that many algorithms internally compute statistics from the data stream and then build the model from their combination. These statistics can be absolute or relative values, probabilities, correlations between attributes, or frequencies of certain patterns. An important example is the

T. Bartz-Beielstein (✉) · L. Hans
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

L. Hans
e-mail: lukas.hans@th-koeln.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_3

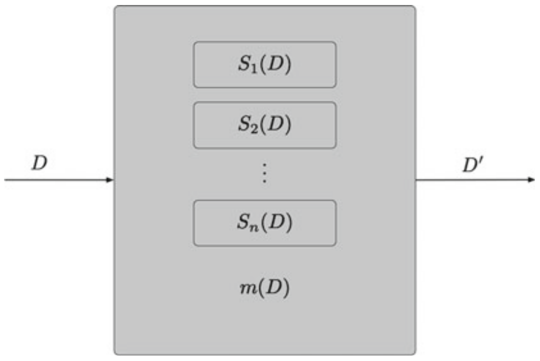


Fig. 3.1 Methods for drift detection: adaptive estimators (Bifet et al., 2018). The input data D are used by the algorithm m that builds the model and by the estimators $S_1, S_2, \ldots, S_n$ used. The estimators compute internal statistics that are used to build the model. The outputs of the model are denoted by D'

Naive-Bayes classifier described in Sect. 2.1.2, which counts the occurrence of attribute values. The perceptron algorithm, which updates the internal weights, predicting the correspondence between attributes and the expected result, also belongs to this category. Figure 3.1 illustrates this approach.

3.1.2 *Change Detectors*

Change detectors use one or more change detection algorithms in addition to and in parallel with the actual algorithm. They trigger a modification of the algorithm settings if a significant change in the data stream or a drop in the quality of the algorithm is observed. The modifications depend on whether an abrupt change has occurred (such that a new model must be built, for example) or a gradual change has been observed (such that the model must be recalibrated). Figure 3.2 illustrates the corresponding architecture.

3.1.3 *Ensemble-Based Approaches*

Ensemble-based approaches are also used for drift detection. For example, Oliveira et al. (2017) train multiple models simultaneously to enable drift detection. When

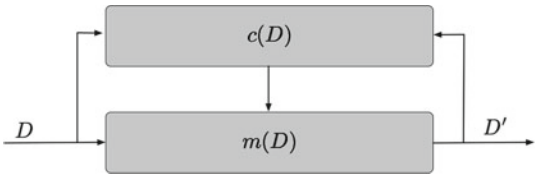


Fig. 3.2 Drift detection methods: in addition to the algorithm m that builds the model, an external change detector c is used (Bifet et al., 2018)

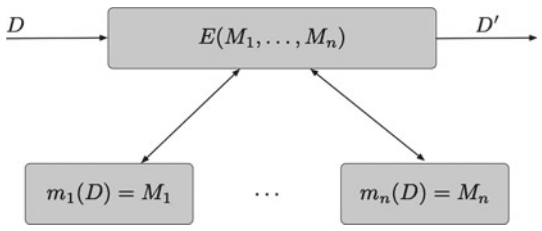


Fig. 3.3 Drift detection methods: ensemble-based approaches train multiple models $M_1, M_2, \ldots, M_n$ simultaneously (Bifet et al., 2018). An ensemble manager E coordinates the different models

the models diverge beyond a certain confidence interval, retraining of the models is triggered. This approach uses complete models that can, in principle, exist independently. An architecture for ensemble-based approaches to drift detection is shown in Fig. 3.3. In the tree-based methods described below (see Sect. 3.4.1), only incomplete submodels are built. These are activated or archived in a clever way, but cannot exist independently of the overall model.

3.2 Basic Considerations for Windowing Techniques

Window-based techniques are important not only for drift detection and handling but also for the evaluation techniques described in Chap. 5. Therefore, we present the basic properties of window-based techniques in this section.

Window-based methods are incremental methods that divide the data into temporal segments. They use a window W of size w that stores the data. The window can be defined as a snapshot of the data. This can be either observation count-based or time-based. A Batch Machine Learning (BML) method can be converted to an incremental method by using a window technique. Well-known windowing techniques include the sliding window model, the damped window model, and the landmark window model.

Definition 3.1 (*Landmark Window Model*) The landmark window model does not discard older data points. All data points are accumulated in the window. The window size increases as more data points arrive. This windowing technique requires large memory resources, especially when dealing with large data streams.

Definition 3.2 (*Sliding Window Model*) The sliding window model is a popular method for discarding older data points and considering only the newer data points for analysis.

Definition 3.3 (*Damped Window Model*) Data points are weighted in the damped window model. Current data points are given higher weight. An exponential fading strategy is used to discard old data. An aging function is used.

The current data points are given greater importance in the sliding and damped window models, and older data points are periodically discarded. The window size

w is an important parameter. The performance of the testing procedure depends on the window size (Bifet et al., 2018):

- If w is too large, predictions become more accurate when there are no changes. If changes occur, they are detected too late.
- If w is too small, patterns needed by the model (learner) are not detected. In addition, the model must be trained too often. However, small w values allow a fast reaction to changes.

3.3 Popular Drift Detection Methods

The drift detection techniques described in this section are not specific to OML. They are also used in the BML environment. Specific implementations for OML algorithms are presented in Sect. 3.4. Drift detection methods are divided into two categories: explicit and implicit methods (Sethi & Kantardzic, 2017). The explicit procedures are also referred to as supervised procedures since they are based on labeled data. The implicit procedures are also referred to as unsupervised procedures since they do not require labeled data. For now, we limit our discussion to explicit procedures, since they are the most commonly used. Implicit procedures are described in Sect. 3.3.4.

Definition 3.4 (*Explicit Drift Detection Methods*) Explicit drift detection methods rely on labeled data to calculate performance metrics such as accuracy, which they can monitor online over time. They detect a drop in performance and therefore efficiently signal changes.

3.3.1 Statistical Tests for Drift and Change Detection

Statistical tests use two data sources to detect a change. They are based on the hypothesis H_0: “The data from both data sources have the same distribution”. Hypothesis tests can be implemented in different ways, e.g. using a reference window W_0, which is not changed, and a sliding window, W_1, which is moved by one unit per time step. If a change is detected, then W_1 becomes W_0 and a new sliding window W_1 is created using the following elements.

3.3.2 Control Charts

Control charts (Montgomery, 2008), which implement statistical tests to detect changes, are also used for drift detection. As an example, consider the Drift Detection Method (DDM) (Gama et al., 2004). The DDM method monitors the error rate of the algorithm and can be applied with BML and OML algorithms. In the following, we consider a classification task. Control charts can also be used for regression tasks (Montgomery, 2008).

In a sample of n examples, the number of errors of the algorithm is a random variable from Bernoulli trials. The binomial distribution is used to model the number of errors in n examples. At time j, the error rate is calculated by p_j with a standard deviation of $\sigma = \sqrt{p_j(1-p_j)/j}$. The DDM method records $p_{\min}$ and $\sigma_{\min}$ while training the learning algorithm. If $p_j + \sigma_j \geq p_{\min} + 2\sigma_{\min}$, a warning is reported. If even $p_j + \sigma_j \geq p_{\min} + 3\sigma_{\min}$, a change in the data stream is reported. As can already be seen from this brief description, control chart-based approaches usually require many parameters (Bifet & Gavaldà, 2009).

Control Charts
More information on control charts can be found on the pages of the National Institute of Standards and Technology:

- NIST Engineering Statistics Handbook, Univariate and Multivariate Control Charts, see https://www.itl.nist.gov/div898/handbook/pmc/section3/pmc3.htm.

3.3.3 Adaptive Windowing (ADWIN)

Adaptive Windowing (ADWIN) manages a variable-length window W in which recently observed data points are stored (Bifet et al., 2018). The mean of these data points in window W is monitored. The size of the window is not fixed, but is dynamically calculated based on the observed rate of change from the data in the window. A window containing the most recent data points in a data stream is kept, and the older data points are continuously discarded if there is sufficient evidence that the mean of the new data points is different from that of the current reference window. The ADWIN algorithm increases the window size w if it detects no change in the data stream, and decreases the window size if it detects a change in the data stream. The data points within the window receive the updated statistics of the data stream, detect concept deviations, and update the model. Figure 3.4 shows the result of drift detection using ADWIN. A data set with clearly detectable abrupt drift was used.

False alarms may occur, if the drift is not abrupt but slow. This can be seen in Fig. 3.5. Here, the ADWIN drift detector was applied to the data set from Sect. 1.2.2. The drift here is not abrupt. A (short) time window is used where data is generated from the old and new concepts. The ADWIN drift detector detects these concept changes, but additionally provides false alarms. The false alarm rate can be influenced by the choice of window size and by other parameters used by ADWIN.

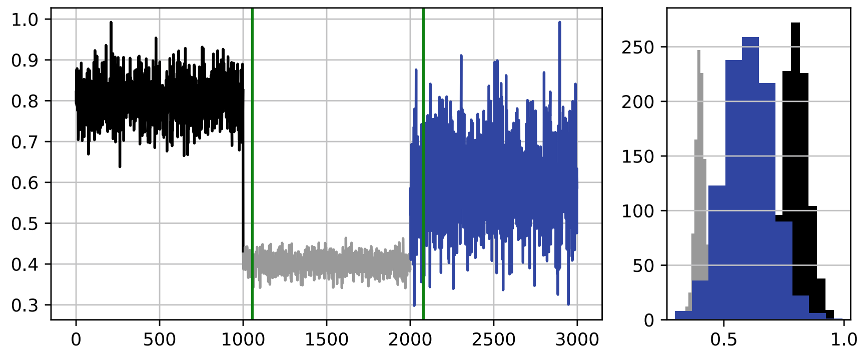


Fig. 3.4 ADWIN drift detection visualized using the data introduced in Sect. 1.2.2. The ADWIN drift detector reports concept changes after 1,055 and 2,079 steps. These are marked by the vertical *green* lines

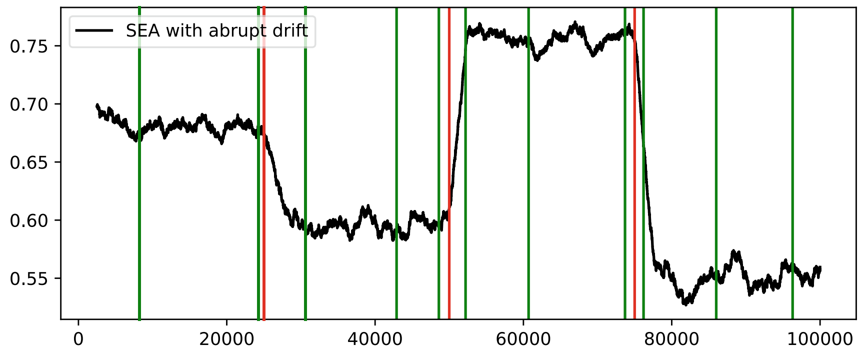


Fig. 3.5 ADWIN drift detection visualized using SEA synthetic data set (SEA). The concept changes reported by ADWIN are marked by the vertical *green* lines. In addition to the three changes (marked in red), changes in five other places are reported, which are false alarms, e.g., after 8,199 steps. This apparent change is due to noise in the data

In general, it is recommended to use adaptive windows (windows with variable size). The idea is to keep data points or observations as long as possible, since the intervals of change are usually unknown. Further guidance can be found in Bifet and Gavaldà (2007) and Bifet and Gavaldà (2009).

Table 3.1 compiles explicit (supervised) drift detection methods.

3.3.4 Implicit Drift Detection Algorithms

In addition to drift detection methods that use metrics, there are also methods that operate unsupervised (Sethi & Kantardzic, 2017).

Table 3.1 Explicit drift detection methods

Explicit drift detection (supervised)	Method	References
Sequential analysis	CUSUM	Page (1954)
	PHT	Page (1954)
	LFR	Wang and Abraham (2015)
Statistical process control	DDM	Gama et al. (2004)
	EDDM	Baena-Garcıa et al. (2006)
	STEPD	Nishida and Yamauchi (2007)
	EWMA	Ross et al. (2012)
Window based distribution	ADWIN	Bifet and Gavaldà (2007)
	DoD	Sobhani and Beigy (2011)
	Resampling	Harel et al. (2014)

Table 3.2 Implicit drift detection methods

Implicit drift detection (unsupervised)	Method	Literature
Novelty detection/clustering methods	OLINDDA	Spinosa et al. (2007)
	MINAS	Faria et al. (2013)
	Woo	Ryu et al. (2012)
	DETECTNOD	Hayat and Hashemi (2010)
	ECSMiner	Masud et al. (2011)
	GC3	Sethi et al. (2016)
Multivariate distribution monitoring	CoC	Lee and Magoules (2012)
	HDDDM	Ditzler and Polikar (2011)
	PCA-detect	Kuncheva and Faithfull (2014)
Model dependent monitoring	A-distance	Dredze et al. (2010)
	CDBD	Lindstrom et al. (2013)
	Margin	Dries and Ulrich (2009)

Definition 3.5 (*Implicit Drift Detection Algorithms*) Implicit (unsupervised) drift detectors rely on properties of the feature values of unlabeled data to signal deviations. They are susceptible to false alarms, but their ability to function without a label makes them useful in applications where labeling is expensive, time-consuming, or is simply not available.

Implicit procedures can be found in Table 3.1. Tables 3.1 and 3.2 are based on the overview prepared by Sethi and Kantardzic (2017).

After this classification of drift detection methods, we take a look at the corresponding implementations for OML algorithms in the following sections.

3.4 OML Algorithms with Drift Detection: Hoeffding-Window Trees

Hoeffding trees that use sliding windows are generally referred to as "Hoeffding-Window Trees" by Bifet and Gavaldà (2009). This includes the Concept-adapting Very Fast Decision Tree (CVFDT) algorithms, see Sect. 3.4.1, and the Hoeffding Adaptive Tree (HAT), see Sect. 3.4.2.

Definition 3.6 (*Hoeffding-Window Tree*) Hoeffding-Window Trees require

1. change detectors (e.g., control charts, see Sect. 3.3.2, or ADWIN, see Sect. 3.3.3) in each node,
2. procedures to create and manage (delete) alternate subtrees, and
3. estimators of statistic S (see Definition 2.1) in the nodes.

3.4.1 Concept-Adapting Very Fast Decision Trees (CVFDT)

The CVFDT method is suitable for data streams with concept drift. The key idea is that each time a change is detected on a subtree, a candidate subtree grows and eventually either the current subtree or the candidate subtree is deleted. CVFDTs update statistics S (see Definition 2.1) in the nodes and leaves by incrementing the counters when a new sample arrives and decrementing the counters of the oldest sample. This simulates forgetting. If the process is stationary, forgetting will have no effect. However, if the process has changed (concept drift), attributes other than the previous ones become important.

Definition 3.7 (*Concept-adapting Very Fast Decision Trees (CVFDT)*) Concept-adapting Very Fast Decision Trees implement the following steps:

1. An alternative subtree is generated if the previous one is questionable. I.e., attributes other than the previous ones are important. The new best attribute is chosen as the root node of the new ("alternative") tree.
2. The old subtree is replaced by the new one if the new one is better.

To allow these subtrees to be computed and evaluated, CVFDT stores the statistic S (see Definition 2.1) in each node, not just in the leaves. The nodes contain a unique identifier (sorted in ascending order) to determine age and manage forgetting. The window size w is changed for CVFDTs. It is reduced when many nodes are simultaneously no longer marked as reliable.

Hulten et al. (2001) have shown that CVFDTs learn a model that is as good as a Very Fast Decision Tree (VFDT) trained using a sliding window of size w in terms of accuracy. CVFDTs require three parameters:

1. Number of examples after which it is checked whether the splits are still correct. If there are better splits, then a subtree is calculated from the corresponding (better) node. The default value is defined by Hulten et al. (2001) $T_0 = 10{,}000$ is given.

2. Number of examples used to generate the alternative tree. Default value: $T_1 = 9{,}000$.
3. Number of examples (after T_1) that compute the quality of the new subtree.

Based on the default values, it can be concluded that changes can be expected after 10,000 instances. This is certainly not the case in every situation, so these default values are of limited use. Further details on CVFDT can be found in Hulten et al. (2001).

3.4.2 Hoeffding Adaptive Trees (HAT)

Section 3.3.3 introduced the ADWIN drift detection method using a sliding window W of dynamic size. The window size w is calculated dynamically, i.e., based on the rate of change observed from the data in the window. The ADWIN algorithm increases the window size when it detects no change in the data stream and decreases the window size when it detects a change in the data stream.

The HAT uses the ADWIN method and is a modified version of the Hoeffding tree. An alternate decision tree is generated at those nodes where the split no longer reflects the current situation. The old tree is then replaced by a new one that is more accurate. The HAT is a modification of the CVFDT described in Sect. 3.4.1. ADWIN monitors the error of each subtree and alternate trees. The HAT method uses the ADWIN estimates to make decisions about the leaves and growth of new trees or alternative trees. It provides a solution to handle the trade-off between windows chosen too small (and the resulting lack of robustness) and windows chosen too large (and the resulting slow detection of changes). However, ADWIN requires more computation time to do this than the CVFDT method. According to Bifet et al. (2018), a HAT is at least as good as a CVFDT.

3.4.3 Overview: Hoeffding-Window Trees

Table 3.3 summarizes key properties of the Hoeffding Window Trees discussed in this section.

Notebook: ADWIN
More details and an ADWIN sample implementation can be found in the GitHub repository https://github.com/sn-code-inside/online-machine-learning.

Table 3.3 Drift detection: tree-based OML-algorithms

Name	Akronym	Drift detection	Remarks
Hoeffding tree, very fast decision tree	`HT`, `VFDT`	No	Base algorithm
Concept-adapting very fast decision trees	`CVFDT`	Window size is changed based on specified parameters. Updates are performed according to previously specified time steps	Require many parameters to be set by the user. In terms of accuracy as good as a HT with fixed-size sliding window. No further theoretical guarantees on the achieved quality
Hoeffding adaptive tree	`HAT`	Adaptive windowing (`ADWIN`)	Adaptive extension of the principle implemented in `CVFDT`. Flexible, but increased computational overhead. Theoretical performance guarantee

3.4.4 *Overview: HT in River*

In the Python package River,[1] several Hoeffding Tree (HT) implementations already exist, see Table 3.4.

3.5 Drift Scaling in Online Machine Learning

As previously discussed in Sect. 1.2.2, the occurrence of drift is a phenomenon commonly observed in real-world data. Concept drift can manifest in diverse forms, encompassing both long-term and short-term variations. A long-term drift could be a changing consumer behavior guided by external influences, while a short-term drift could be represented by voltage interruptions in the power grid. Despite their distinct origins, both types of drift are treated similarly within the classical BML paradigm. The treatment entails two steps: drift detection followed by the initialization of a new model. In contrast, the OML approach bypasses this process by automatically incorporating each new data point into the model. However, OML models encounter challenges when faced with certain types of concept drift, particularly in cases involving structural disruptions. In such instances, the value range of input variables can

[1] https://riverml.xyz/.

Table 3.4 Hoeffding tree implementations in the River package. The "non-stationary" column indicates whether the algorithm implements methods for handling non-stationary data streams (e.g., drift). The acronym ADWIN stands for "Adaptive Windowing" and is explained in Sect. 3.3.3

Name	Acronym	Task	Non-stationary?	Remarks
Hoeffding tree classifier	`HTC`	Classification		Base-algorithm for OML-classification tasks
Hoeffding adaptive tree classifier	`HATC`	Classification	✓	Modified HTC by adding an instance of ADWIN to each node to detect and react to drift
Extremely fast decision tree classifier	`EFDT`	Classification		Implements splits as quickly as possible. Reviews decisions regularly and repeats them as needed
Hoeffding tree regressor	`HTR`	Regression		Base-Hoeffding-Tree for regression tasks. Adaptation of the HTC algorithm for regression
Hoeffding adaptive tree regressor	`HATR`	Regression	✓	Modifies `HTR` by adding an instance of `ADWIN` to each node to detect and react to drift
incremental structured-output prediction tree regressor	`iSOUPT`	Multi-target regression		Multi-target version of the `HTR`
Label combination Hoeffding tree classifier	`LCHTC`	Multi-label classification		Creates a numeric code for each combination of binary labels and uses `HTC` to learn from this coded representation. Decodes at prediction time the modified representation to get the original labels

rapidly and significantly change, thereby elongating the adaptation phases of OML models.

Normalization of data is a customary step in BML to prevent the dominance of numerically larger values over numerically smaller ones. Nevertheless, normalization alone does not ensure parity of significance among different parameters, as one feature may inherently carry greater weight than another. The study conducted by Singh and Singh (2020) explored the impact of various treatment techniques on model performance when compared to untreated data. Interestingly, their findings indicated that normalization of input variables occasionally led to a deterioration in algorithmic performance (Singh & Singh, 2020). Nonetheless, we adopt the underlying assumption shared by normalization methods in OML, namely that scaling the data can yield improved results. In Sect. 9.3, we conduct a detailed case study and scrutinize this hypothesis.

Broadly, scaling methods can be categorized into two primary classes: mean-standard deviation-based procedures and minimum-maximum-based procedures. Additionally, alternative categories include median-based procedures (Jain et al., 2005), hyperbolic tangent-based procedures (Hampel et al., 2005), and sigmoidal-based procedures (Priddy & Keller, 2005). Furthermore, it is important to note that normalization in deep learning is referred to as "batch normalization", which specifically pertains to scaling the weights of different layers. Batch normalization ensures stability in the distribution of layer inputs, consequently enhancing performance (Santurkar et al., 2019).

In the context of OML, where data points are received sequentially, this sequential nature influences statistical measures such as mean, variance, maximum, and minimum values. Consequently, the estimation of these measures must also be conducted sequentially. The sub-chapter follows a well-organized structure, commencing with the adaptation of statistical measures in a streaming fashion. Once the foundational concepts are established, we delve into various scaling techniques. These include mean-standard deviation-based methods, encompassing z-score normalization, mean-centering, and Pareto scaling. Subsequently, we explore the minimum-maximum normalization and the max-abs normalization methods.

3.5.1 Statistical Measures in a Sequential Manner

Let m represent the number of observations, denoted by x_i, where i ranges from 1 to m. Upon acquiring new observations denoted by x_i, where i ranges from $m+1$ to $m+n$, with n being equal to one in the OML setting and any value larger than one indicating a mini-batch approach, the following equations can be formulated:

$$\mu = \frac{1}{m+n} \sum_{i=1}^{m+n} x_i, \tag{3.1}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{3.2}$$

and

$$\mu = \frac{1}{n}\sum_{i=m+1}^{m+n} x_i. \tag{3.3}$$

Equation 3.1 represents the recalculated mean value, obtained by summing all the observations. Equation 3.2 captures the present empirical mean, while Eq. 3.3 denotes the mean of the newly acquired data point. Equation 3.1 can be simplified as follows:

$$\mu = \frac{m}{m+n}\mu_m + \frac{n}{m+n}\mu_n.$$

By employing this notation, the mean undergoes iterative updates upon the incorporation of successive data points, thereby circumventing the need to retain the entirety of individual data points. Once the mean has been transformed into its sequential representation, the subsequent task involves extending this approach to the calculation of the variance. To this end, we introduce the following equations:

$$\sigma^2 = \frac{1}{m+n}\sum_{i=1}^{m+n}(x_i - \mu)^2 \tag{3.4}$$

$$\sigma_m^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_m)^2 \tag{3.5}$$

$$\sigma_n^2 = \frac{1}{n}\sum_{i=m+1}^{m+n}(x_i - \mu_n)^2. \tag{3.6}$$

In this context, Eq. 3.4 denotes the revised variance, Eq. 3.5 represents the existing variance, and Eq. 3.6 signifies the variance associated with the newly introduced data portion. Furthermore, we observe that

$$\sigma^2 = \frac{1}{m+n}\sum_{i=1}^{m+n} x_i^2 - \mu^2 \tag{3.7}$$

$$\frac{1}{m}\sum_{i=1}^{m} x_i^2 = \sigma_m^2 + \mu_m^2 \tag{3.8}$$

$$\frac{1}{n}\sum_{i=m+1}^{m+n} x_i^2 = \sigma_n^2 + \mu_n^2. \tag{3.9}$$

With the help of Eqs. 3.4–3.9, we can formulate the variance in a sequential manner given by the following equation:

$$\sigma^2 = \frac{m}{m+n}(\sigma_m^2 + \mu_m^2) + \frac{n}{m+n}(\sigma_n^2 + \mu^2 n) - \mu^2$$
$$= \frac{m}{m+n}\sigma_m^2 + \frac{n}{m+n}\sigma_n^2 + \frac{mn}{(m+n)^2}(\mu_m - \mu_n)^2.$$

The update operation entails a linear combination of the observed variances, supplemented by an adjustment incorporating the means. Notably, at the initial stage, setting $m = \mu_m = \sigma_m^2 = 0$ allows the formulas to remain valid and produce accurate results. Having established the procedures for adjusting the mean and variance, we can now transition to the estimators.

3.5.2 Adapted Scaling Techniques

The z-score normalization employs both the mean and standard deviation to scale the values. The resulting scaled data exhibit a mean of zero and a variance of one (Kappal, 2019). The k-th feature of the l-th data point is denoted by $x_{k,l}$.

Each data point $x_{k,l}$ can be transformed into the normalized data point $x_{k,l}^*$ according to the following equation:

$$x_{k,l}^* = \frac{x_{k,l} - \mu_k}{\sigma_k},$$

where μ_k and σ_k represent the mean and standard deviation, respectively, of the k-th feature.

In mean centering, the mean of each data point is subtracted. We define this process as follows:

$$x_{k,l}^* = x_{k,l} - \mu_k.$$

Pareto scaling follows a similar rationale to z-score normalization, but in this case, the division is performed by the square root of the standard deviation (Noda, 2008). Consequently, we have

$$x_{k,l}^* = \frac{x_{k,l} - \mu_k}{\sqrt{\sigma_k}}.$$

By utilizing Pareto scaling instead of z-score normalization, it is possible to overcome limitations that may arise due to unit variance (van den Berg et al., 2006).

The various methods based on mean and standard deviation help minimize the impact of outliers in the data. However, all these methods face a common challenge; they do not bring individual features to the same numerical range, as the values may change over time (Pan et al., 2016).

To apply minimum-maximum-based methods, adjustments to the batch process are necessary. In BML, the maximum and minimum values are determined once for all data points. In OML, this process is adapted such that the minimum and maximum values are continuously updated. If the current data point exceeds the historical maximum or falls below the historical minimum, they are replaced with the current value.

The min-max method scales the data linearly to a defined upper and lower bound (Han et al., 2012). These bounds are typically set between zero and one or between minus one and one. The transformation is carried out using the following equation:

$$x^*_{k,l} = \frac{x_{k,l} - \min(x_k)}{\max(x_k) - \min(x_k)},$$

where min and max represent the minimum and maximum values of the k-th feature. This particular formula scales the data to the range of zero to one. To define a different range, a simple linear transformation can be applied to the data.

In the case of max-abs normalization, it is a variant of min-max normalization. Here, each feature is scaled to the range of -1 to 1 (Li & Liu, 2011). The scaling is performed using the following equation:

$$x^*_{k,l} = \frac{x_{k,l}}{\max(|x_k|)}.$$

By employing these scaling methods, the relationships of the original data are preserved in comparison to mean and standard deviation-based methods. However, these techniques are susceptible to the influence of outliers due to the reliance on maximum and minimum values (Dougherty, 2011).

As previously mentioned, the application of scaling methods is expected to enhance performance, particularly in the case of drift events. We delve into this hypothesis in Sect. 9.3, where case studies are conducted to address this issue.

References

Baena-Garcıa, M., et al. (2006). Early drift detection method. In *Fourth International Workshop on Knowledge Discovery from Data Streams* (Vol. 6, pp. 77–86).

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)* (pp. 443–448). SIAM.

Bifet, A., & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII. IDA'09* (pp. 249–260). Springer.

Bifet, A., Gavalda, R., et al. (2018). *Machine learning for data streams with practical examples in MOA*. MIT Press.

Ditzler, G., & Polikar, R. (2011). Hellinger distance based drift detection for nonstationary environments. In *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)* (pp. 41–48). IEEE.

Dougherty, G. (2012). *Pattern recognition and classification: An introduction*, 1st Ed. (pp. XI, 196). Springer. ISBN: 978-1-4614-5322-2. https://doi.org/10.1007/978-1-4614-5323-9

Dredze, M., Oates, T., & Piatko, C. (2010). We're not in kansas anymore: Detecting domain changes in streams. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing* (pp. 585–595).

Dries, A., & Rückert, U. (2009). Adaptive concept drift detection. *Statistical Analysis and Data Mining, 2*(5–6), 311–327.

Faria, E. R., Gama, J., & Carvalho, A. C. P. L. F. (2013). Novelty detection algorithm for data streams multi-class problems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (pp. 795–800).

Gama, J., et al. (2004). Learning with drift detection. In A. L. C. Bazzan & S. Labidi (Eds.), *Advances in artificial intelligence, SBIA 2004* (pp. 286–295). Springer.

Hampel, F. R., et al. (2005). *Robust statistics: The approach based on influence functions*. Wiley Series in probability and statistics. First published: 22 March 2005. Online ISBN: 9781118186435. Copyright–1986. John Wiley & Sons, Inc. All rights reserved. Wiley. ISBN: 9780471735779. https://doi.org/10.1002/9781118186435

Han, J., Kamber, M., & Pei, J. (2012). 3-Data preprocessing. In J. Han, M. Kamber, & J. Pei (Eds.), *Data mining*, 3rd Ed (pp. 83–124). The Morgan Kaufmann Series in data management systems. Morgan Kaufmann. ISBN: 978-0-12-381479-1. https://doi.org/10.1016/B978-0-12-381479-1.00003-4. https://www.sciencedirect.com/science/article/pii/B9780123814791000034

Harel, M., et al. (2014). Concept drift detection through resampling. In *International Conference on Machine Learning, PMLR* (pp. 1009–1017).

Hayat, M. Z., & Hashemi, M. R. (2010). A DCT based approach for detecting novelty and concept drift in data streams. In *2010 International Conference of Soft Computing and Pattern Recognition* (pp. 373–378). IEEE.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'01, New York, NY, USA* (pp. 97–106). Association for Computing Machinery.

Jain, A., Nandakumar, K., & Ross, A. (2005). Score normalization in multimodal biometric systems. *Pattern Recognition, 38*(12), 2270–2285. ISSN: 0031-3203. https://doi.org/10.1016/j.patcog.2005.01.012. https://www.sciencedirect.com/science/article/pii/S0031320305000592.

Kappal, S. (2019). Data normalization using median median absolute deviation (MMAD) based Z-score for robust predictions vs. min-max normalization. *London Journal of Research in Science: Natural and Formal, 19*(4), 39–44.

Kuncheva, L. I., & Faithfull, W. J. (2014). PCA feature extraction for change detection in multidimensional unlabeled data. *IEEE Transactions on Neural Networks and Learning Systems, 25*(1), 69–80.

Lee, J., & Magoules, F. (2012). Detection of concept drift for learning from stream data. In *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems* (pp. 241–245). IEEE.

Li, W., & Liu, Z. (2011). A method of SVM with normalization in intrusion detection. *Procedia Environmental Sciences, 11*. In *2011 2nd International Conference on Challenges in Environmental Science and Computer Engineering (CESCE 2011)* (pp. 256–262). ISSN: 1878-0296. https://doi.org/10.1016/j.proenv.2011.12.040

Lindstrom, P., Namee, B. M., & Delany, S. J. (2013). Drift detection using uncertainty distribution divergence. *Evolving Systems, 4*(1), 13–25.

Masud, M., et al. (2011). Classification and novel class detection in concept drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering, 23*(6), 859–874.

Montgomery, D. C. (2008). *Statistical quality control*. Wiley.

Nishida, K., & Yamauchi, K. (2007). Detecting concept drift using statistical testing. In *International Conference on Discovery Science* (pp. 264–269). Springer.

Noda, I. (2008). Scaling techniques to enhance two-dimensional correlation spectra. *Journal of Molecular Structure*, 883–884. In *Progress in two-dimensional correlation spectroscopy* (pp. 216–227). ISSN: 0022-2860. https://doi.org/10.1016/j.molstruc.2007.12.026. https://www.sciencedirect.com/science/article/pii/S0022286007008411

Oliveira, G., et al. (2017). Time series forecasting in the presence of concept drift: A PSO-based approach. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 239–246).

Page, E. S. (1954). Continuous inspection schemes. *Biometrika, 41*(1–2), 100–115.

Pan, J., Zhuang, Y., & Fong, S. (2016). The impact of data normalization on stock market prediction: Using SVM and technical indicators. In M. W. Berry, A. H. Mohamed, & B. W. Yap (Eds.), *Soft computing in data science* (pp. 72–88). Springer. ISBN: 978-981-10-2777-2.

Priddy, K. L., & Keller, P. E. (2005). *Artificial neural networks: An introduction* (Vol. TT68). SPIE Press. ISBN: 9780819459879.

Ross, G. J., et al. (2012). Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters, 33*(2), 191–198.

Ryu, J. W., et al. (2012). An efficient method of building an ensemble of classifiers in streaming data. In *International Conference on Big Data Analytics* (pp. 122–133). Springer.

Santurkar, S., et al. (2019). How does batch normalization help optimization? arXiv: 1805.11604 [stat.ML].

Sethi, T. S., & Kantardzic, M. (2017). On the reliable detection of concept drift from streaming unlabeled data. arXiv:1704.00023

Sethi, T. S., Kantardzic, M., & Hu, H. (2016). A grid density based framework for classifying streaming data in the presence of concept drift. *Journal of Intelligent Information Systems, 46*(1), 179–211.

Singh, D., & Singh, B. (2020). Investigating the impact of data normalization on classification performance. *Applied Soft Computing, 97*, 105524. ISSN: 1568-4946. https://doi.org/10.1016/j.asoc.2019.105524

Sobhani, P., & Beigy, H. (2011). New drift detection method for data streams. In *International Conference on Adaptive and Intelligent Systems*. Springer.

Spinosa, E. J., de Leon F. de Carvalho, A. P., & Gama, J. (2007). Olindda: A cluster-based approach for detecting novelty and concept drift in data streams. In *Proceedings of the 2007 ACM Symposium on Applied Computing* (pp. 448–452).

van den Berg, R. A., et al. (2006). Centering, scaling, and transformations: Improving the biological information content of metabolomics data. *BMC Genomics, 7*,. https://doi.org/10.1186/1471-2164-7-142. https://www.scopus.com/inward/record.uri?eid=2-s2.0-33747019547&doi=10.1186md5=30f58f90a22f5cb8c3643ee664385e30

Wang, H., & Abraham, Z. (2015). Concept drift detection for streaming data. In *International Joint Conference on Neural Networks (IJCNN)* (pp. 1–9).

Chapter 4
Initial Selection and Subsequent Updating of OML Models

Thomas Bartz-Beielstein

Abstract In Sect. 4.1, we describe a current best practice methodology for the initial model selection of Online Machine Learning (OML) models, taking into account that the model is continuously updated. In Sect. 4.2, we discuss possibilities for removing or changing observations/instances that have already been added to the model. We describe how completely new features can be added to the model afterwards. In addition, we show how it is ensured that the model quality is still adequate after a model update. Catastrophic forgetting (catastrophic interference) is considered in Sect. 4.3 in the OML context: The continuous updating of the OML models carries the risk that this learning is not successful if correctly learned older relationships are falsely forgotten ("de-learned").

4.1 Initial Model Selection

Based on the results of the studies described in Chap. 9, recommendations for the initial model selection are only possible to a very limited extent. This is due to the fact that currently no OML method is "out-of-the-box" ready for use. In addition to the data preprocessing, which is also required in the Batch Machine Learning (BML) environment, the determination of suitable hyperparameters is a major challenge for OML algorithms.

In an experimental environment, the use of OML methods is probably possible. Such an environment is more likely to be found in the academic field and not in productive use, such as at the Federal Statistical Office, where reliable results play a major role. The special requirements for machine learning in statistical institutions are discussed in Sect. 7.1.4.

In Chap. 8, the current development is described in detail. In particular, the combination of OML methods with methods for hyperparameter tuning (Bartz et al., 2022) opens up promising perspectives.

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_4

Recommendations for Initial Model Selection
Based on the current state, the following recommendations can be given:

1. In the run-up to the model selection, data preprocessing adapted to the OML situation (in particular the axioms for stream learning, see Definition 1.12) must be taken into account.
2. In addition, an adequate quality measure should be selected. It is crucial which data is used for training and which for testing (see the discussion in Chap. 5). If possible, several quality measures should be considered at the same time.
3. Besides the quality measures, computing time and memory consumption should also be observed.
4. After the first three points have been clarified, if possible, a simple BML method should be trained. This provides comparison values for assessing the quality.
5. During the initial selection of an OML method, simple methods should be used. Linear models or Hoeffding Adaptive Tree Regressor (HATR) for regression and logistic regression or Hoeffding Adaptive Tree Classifier (HATC) for classification are particularly suitable here.
6. The hyperparameters of these algorithms should be adjusted; otherwise, too large trees with very long computing times and high memory requirements are generated. For example, the value `max_depth` that sets the maximum tree depth can be adjusted. The default setting specifies that the trees can be infinitely large.
7. Are sufficient memory and time available and the quality is not satisfactory, ensemble methods can be used. Here, for example, "Adaptive Random Forest regressor" or "Adaptive Random Forest classifier" can be mentioned (Gomes et al., 2017).

4.2 Updating and Changing the Model

4.2.1 Adding New Features

One strength of OML methods is that they can handle new attributes and classes that appear in the data stream. The Python package River uses so-called "dictionaries" to manage the data. A dictionary can be empty at the beginning, e.g., if it is not yet known which features with which values are present in the data stream. The model is built without having seen any data. During training, the features are learned by the model. This gives OML methods a big advantage over BML methods. Many BML methods, e.g., the tree-based standard algorithms from the scikit-learn: Machine Learning in Python (sklearn) package, cannot handle the occurrence of new levels of a feature or even completely new features. Here, the BML model has to be retrained from scratch.

Notebook: Adding New Features
An example of how attributes can be added to a model afterwards can be found in the GitHub repository https://github.com/sn-code-inside/online-machine-learning.

4.2.2 *Manual Model Changes in Response to Drift*

If drift has been detected, the OML model attributes can be mutated. If the performance of the model deteriorates significantly, the model can be reset to the default settings by cloning.

Mutation and Cloning in River

The method `mutate()` changes attributes of a model, e.g., the learning rate of the optimization algorithm of the OML linear regression. The method `clone()` creates a deep copy of the model. The cloned model does not have any information about the data.

Tip: Mutation and Cloning
Further information can be found on the River project page:
river: recipes/cloning-and-mutating/.

4.2.3 *Ensuring Model Quality After a Model Update*

The Hoeffding-Window trees presented in Sect. 3.4 use internal procedures to ensure model quality after a model update. It is also interesting to consider performing the model update only when it is ensured that it achieves an improvement in model quality. This is taken into account by the Concept-adapting Very Fast Decision Tree (CVFDT) and Hoeffding Adaptive Tree (HAT) algorithms. An alternative subtree is created if the previous one is questionable. For example, attributes other than the previous ones are more important. The new best attribute is chosen as the root node of the new ("alternative") tree. The old subtree is only replaced by the new subtree if the new one is better. In addition, previously trained models can be archived and used if the new model shows a drop in performance.

In addition to the internal guarantee of model quality by the algorithm itself, the quality can also be ensured by combining several algorithms. In "bandit-based model selection", each model is assigned an arm. Each time the algorithm is called, a decision is made which arm/model should be pulled (Busa-Fekete et al., 2014).

4.3 Catastrophic Forgetting

Catastrophic forgetting, also known as catastrophic interference, describes the (undesirable) behavior of Machine Learning (ML) models to forget learned things when new updates are made. Correctly learned older tendencies are falsely forgotten or "de-learned" when new tendencies are learned from new data.

Catastrophic forgetting was originally defined as a problem that occurs in (deep) Neural Networks (NNs) (Chen et al., 2018; McCloskey & Cohen, 1989). The complexity of NNs makes them sensitive to the problem of catastrophic forgetting. The way a NN learns is to perform many update passes on the parameters (coefficients, weights). At each update, the model should fit the data a little better. Even for a simple NN, many coefficients have to be trained, especially compared to so-called "shallow ML" methods such as logistic regression. Catastrophic forgetting is relevant for OML: If models are updated with every new data point, it is to be expected that the coefficients will change over time.

The term "drift" is used to refer to drift either in the variables (data drift) or in the relationships between independent variables and dependent variables (concept drift) (see Definition 1.7). Since catastrophic forgetting is a problem of the trained model coefficients within the model, it makes sense to consider catastrophic forgetting independently of the phenomenon of drift. The explainability and interpretability of the ML algorithms and models (see Sect. 6.6) are directly related to the topic of catastrophic forgetting.

Notebook: Catastrophic Forgetting in the Context of OML
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning shows an example of how catastrophic forgetting can occur.

4.3.1 Strategies for Dealing with Catastrophic Forgetting

Three methods against catastrophic forgetting (model monitoring, drift detection, and explainability) are described below.

4.3.1.1 Model Monitoring (Performance and Error Rates)

Even if, in the step-by-step evaluation of the model, the individual error at each data point does not become particularly large, at the end of the process or after a certain period of time the model may have forgotten ("de-learned") the properties of the first data points. Re-evaluation of past instances can be used to observe catastrophic forgetting in practice.

If the performance of the model and the distribution of the data as well as other Key Performance Indicators (KPIs) and descriptive statistics are closely monitored, catastrophic forgetting should be detected and countermeasures taken in time.

4.3.1.2 Drift Detection

A second solution is the application of the drift detection methods described in Chap. 3.

4.3.1.3 Explainability

As a third tool for dealing with catastrophic forgetting, methods for explaining models can be used. ML models, including OML models, are often understood and used as black box models: The results of learning are the only thing that matters and are analyzed (see Chap. 5), but relatively little time is spent on examining the inner mechanisms of the models. This becomes a problem when incorrectly learned patterns occur. Therefore, the use of tools for explaining models as described in Sect. 6.6 is recommended.

The performance monitoring described in Sect. 4.3.1.1 is mainly a black box approach. It is interesting that we can also extract elements such as trees (dendrograms), coefficients, variable importance, and the like to see what has actually changed in the model. However, there is no simple, general method to examine and explain the inner workings of the models. Each ML model category implements its own specific method of adapting to the data. The approaches described in Sect. 6.6 for interpreting and explaining models are relevant.

References

Bartz, E., et al. (2022). *Hyperparameter tuning for machine and deep learning with R—A practical guide*. Springer. https://doi.org/10.1007/978-981-19-5170-1.

Busa-Fekete, R., & Hüllermeier, E., et al. (2014). A survey of preference-based online learning with bandit algorithms. In P. Auer (Ed.), *Algorithmic learning theory* (pp. 18–39). Springer.

Chen, Z., et al. (2018). *Lifelong machine learning*, 2nd ed. Morgan and Claypool Publishers.

Gomes, H. M., et al. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning, 106*(9), 1469–1495.

McCloskey, M., & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation - Advances in Research and Theory, 24*(C), 109–165.

Chapter 5
Evaluation and Performance Measurement

Thomas Bartz-Beielstein

Abstract This chapter discusses aspects to be considered when evaluating Online Machine Learning (OML) algorithms, especially when comparing them to Batch Machine Learning (BML) algorithms. The following considerations play an important role:

1. How are training and test data selected?
2. How can performance be measured?
3. What procedures are available for generating benchmark data sets?

Section 5.1 describes the selection of training and test data. Section 5.2 presents an implementation in Python for selecting training and test data. Section 5.3 describes the calculation of performance. Section 5.4 introduces the generation of benchmark data sets in the field of OML.

5.1 Data Selection Methods

When determining the data selection method and calculating the performance, there is the greatest difference between BML and OML. Among other things, in OML the resources (memory and time, but not the data) are severely limited. In addition, Cross Validation (CV) is not possible. It is very important to determine which instances are used for training and for testing (and possibly also for validation).

For each of the selection approaches presented in the following, a metric must be selected, e.g., accuracy or Mean Absolute Error (MAE).

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_5

5.1.1 Holdout Selection

In the holdout evaluation method, the performance of the model is evaluated against a test data set, which consists of examples that have not yet been sighted. These examples are used only for evaluation purposes and not for the training of the model.

Definition 5.1 (*Holdout*) In the holdout evaluation method, the performance is evaluated after each batch, i.e., after a certain number of examples or observations. For this purpose, two parameters must be defined:

1. Size of the (holdout-) window and
2. frequency of testing.

The holdout evaluation is best when current and representative holdout data are used.

Why are holdout data not always used for OML? It is not always easy or even possible to obtain these data. In addition, the holdout data set must be representative, which cannot be guaranteed with streaming data due to possible changes. The holdout data of today can already be outdated tomorrow. If the period in which the holdout data are collected is too short, these data may contain essential relationships.

5.1.2 Progressive Validation: Interleaved Test-Then-Train

In statistics, progressive validation is generally understood to be the validation over a longer period of time, e.g., by using control charts. In the streaming data context, the term is used for approaches in which the individual instances are first used for testing (determining the quality of the model, the model calculates a prediction) and then for learning (training the model). Each individual instance is analyzed according to its arrival order. In addition to simple progressive validation, we also consider prequential validation and delayed progressive validation.

5.1.2.1 Progresssive Validation

Definition 5.2 (*Progressive Validation*) Each observation can be denoted as (X_t, y_t), where X_t is a set of features, y_t is a label (or a prediction value), and t denotes the time (or simply the index). Before updating the model with the pair (X_t, y_t), the model calculates a prediction for X_t, so that $\hat{y}_t$ is calculated. Using the ground truth y_t and the predicted value $\hat{y}_t$ from the model, the online metric is then updated. Common metrics such as accuracy, MAE, Mean Squared Error (MSE), and Area Under The Curve, Receiver Operating Characteristics (ROC, AUC) are all sum values and can therefore be updated online.

This procedure can also be used for time series: If there are t observations $(x_1, x_2, \dots, x_t)$, then the values $(x_{t-k}, x_{t-k+1}, \dots, x_{t-1})$ can be used as X_t and the value x_t as y_t. Alternatively, additional features can be calculated from the values $(x_{t-k}, x_{t-k+1}, \dots, x_{t-1})$, which are then used as X_t. Typical features are the information about the day of the week or the season.

5.1.2.2 Prequential Validation

Definition 5.3 (*Prequential Validation*) Prequential validation works like progressive validation (interleaved test-then-train). However, the new instances are more important than the old ones. This is implemented by a sliding window or a decay factor.

5.1.2.3 Delayed Progressive Validation

Typically, an OML model calculates a prediction $\hat{y}_t$ and then learns. This was referred to as "progressive validation" in Sect. 5.1.2. The prediction and the observed value can be compared to measure the correctness of the model. This approach is often used to evaluate OML models. In some cases, this approach is not appropriate, because the prediction and the ground truth are not available at the same time. In this case, it makes sense to delay the process until the ground truth is available. This is called delayed progressive validation.

Delayed Progressive Validation

While evaluating a machine learning model, the goal is to simulate production conditions to get a trustworthy assessment of the model's performance. For example, consider the number of bicycles needed for a bike rental for the next week. Once 7 days have passed, the actual demand is known, and we can update the model. What we really want is to evaluate the model by, for example, forecasting seven days in advance and only updating the model when the ground truth is available (Grzenda et al., 2020).

The delayed progressive validation is of great importance for practice: Instead of updating the model immediately after it has made a prediction, it is only updated when the ground truth is known. In this way, the model more accurately reflects the real process.

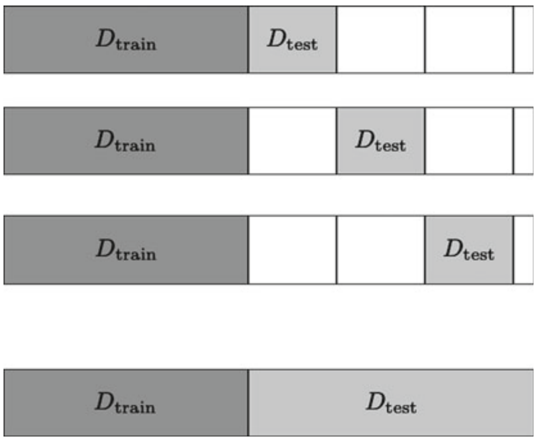


Fig. 5.1 Batch method with a prediction horizon. The training data set D_{train} is used once. The model M_{bml} trained on D_{train} is tested on the individual partitions of the test data set D_{test} one after the other. The lower figure shows (as a special case) the data sets when a classical holdout approach is used. In this case, the size of the test data set is equal to the size of the horizon

5.1.3 Machine Learning in Batch Mode with a Prediction Horizon

The method `eval_bml_horizon` implements the "classical" BML approach: The classical BML algorithm is trained once on the training data set, resulting in a model, say $M_{\text{bml}}^{(1)}$, which is not changed: $M_{\text{bml}}^{(1)} = M_{\text{bml}}$.

The model M_{bml} is evaluated on the test data, where the horizon, say $h \in [1, s_{\text{test}}]$, comes into play: h specifies the size of the partitions into which D_{test} is divided. If $h = s_{\text{test}}$, then the standard procedure of Machine Learning (ML) ("train-test") is implemented. If $h = 1$, a pure OML-setting is simulated. The OML procedure is only simulated in this case, since the model M_{bml} is not updated or retrained. The BML approach is shown in Fig. 5.1.

If the entire test data set is used for the prediction horizon in the batch method, i.e., $s_{\text{test}} = h$, then we obtain the classical holdout approach (see Sect. 5.1.1).

5.1.4 Landmark Batch Machine Learning with a Prediction Horizon

The method `eval_bml_landmark` implements a landmark approach. The first step is similar to the first step of the BML approach and $M_{\text{bml}}^{(1)}$ is available. The following steps are different: After a prediction with $M_{\text{bml}}^{(1)}$ for the batch of data instances from the interval $[s_{\text{train}}, s_{\text{train}} + h]$ has been calculated, the algorithm is retrained on the interval $[1, s_{\text{train}} + h]$ and an updated model $M_{\text{bml}}^{(2)}$ is available. In the third step of the landmark BML, $M_{\text{bml}}^{(2)}$ calculates predictions for $[s_{\text{train}} + h, \text{train} + 2 \times h]$ and a new model $M_{\text{bml}}^{(2)}$ is trained on $[1, \text{train} + 2 \times h]$. The landmark approach is shown in Fig. 5.2.

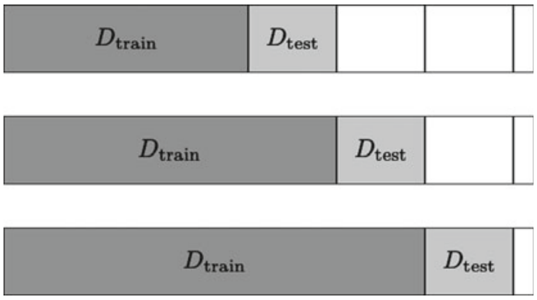


Fig. 5.2 Landmark batch method with an prediction horizon

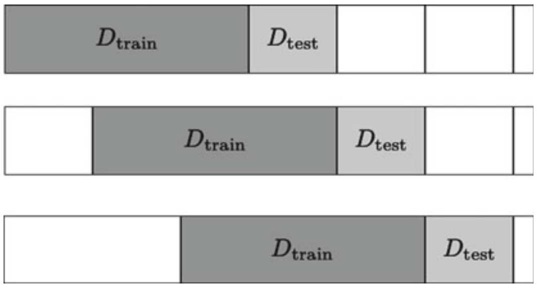


Fig. 5.3 Window-batch method with a prediction horizon. This division of the training and test data set ensures that the size of the training data set s_{train} remains unchanged and that a prediction horizon h of the same size is always used

5.1.5 Window-Batch Method with Prediction Horizon

The method `eval_bml_window` implements a window approach. Here, too, the first step is similar to the first step of the BML approach and $M_{\text{bml}}^{(1)}$ is available. The following steps are similar to the landmark approach, with one important exception: The algorithm is not trained on the complete set of seen data. Instead, it is trained on a sliding window of size s_{train}. The window batch approach is shown in Fig. 5.3.

5.1.6 Online-Machine Learning with a Prediction Horizon

The method `eval_oml_horizon` implements an OML approach. This approach differs fundamentally from the batch approaches of ML, since each individual instance is used for prediction and training. If $h = 1$, a “pure” OML algorithm is implemented. If $h > 1$, the OML calculations are performed h times.

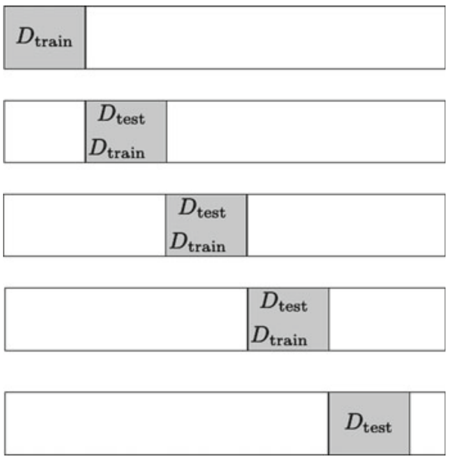


Fig. 5.4 Iterative OML method. If the window size h is one, then an example is used for testing and then for training (updating) the OML algorithm. If $h > 1$, then the calculations are performed h times and the average of these h results is calculated

5.1.7 *Online-Maschine Learning*

The method `eval_oml_iter_progressive` is based on the method `progressive_val_score` from the package `River`.[1] The iterative procedure is shown in Fig. 5.4.

Table 5.1 provides a comparative overview of the selection methods.

5.2 Determining the Training and Test Data Set in the Package spotRiver

5.2.1 *Methods for BML und OML*

The BML algorithms require a training data set D_{train} of size s_{train} to adapt the model. The test data set D_{test} of size s_{test} is used to evaluate the model on new (unseen) data. For the comparative evaluation of BML and OML algorithms, the package Sequential Parameter Optimization Toolbox for River (spotRiver) provides five different methods.

The four evaluation functions shown in Table 5.2 accept two data frames as arguments: a training and a test data set. In the pure OML environment, the fifth evaluation function `eval_oml_iter_progressive` is used. This uses only one (test) data set, as it implements the progressive validation. The parameters are shown in Table 5.3.

[1] See https://riverml.xyz/0.15.0/api/evaluate/progressive-val-score/.

Table 5.1 Selection methods. The batches are represented by intervals, e.g., $[a, b]$. In the OML approaches, each instance from the interval is passed to the online algorithm separately for prediction and updating (training)

Name	Step	Training interval/instances	Training batch size	Model	Prediction interval
BML horizon	1	$[1, s_{\text{train}}]$	s_{train}	$M^{(1)}$	$[s_{\text{train}} + 1, s_{\text{train}} + h]$
	n	$[1, s_{\text{train}}]$	0	$M^{(1)}$	$[s_{\text{train}} + (n - 1) \times h + 1, s_{\text{train}} + n \times h]$
BML landmark	1	$[1, s_{\text{train}}]$	s_{train}	$M^{(1)}$	$[s_{\text{train}} + 1, s_{\text{train}} + h]$
	n	$[1, s_{\text{train}} + (n - 1) \times h]$	$s_{\text{train}} + (n - 1) \times h$	$M^{(n)}$	$[s_{\text{train}} + (n - 1) \times h + 1, s_{\text{train}} + n \times h]$
BML window	1	$[1, s_{\text{train}}]$	s_{train}	$M^{(1)}$	$[s_{\text{train}} + 1, s_{\text{train}} + h]$
	n	$[1 + (n - 1) \times h, s_{\text{train}} + (n - 1) \times h]$	s_{train}	$M^{(n)}$	$[s_{\text{train}} + (n - 1) \times h + 1, s_{\text{train}} + n \times h]$
OML horizon	1	$[1, s_{\text{train}}]$	1	$M^{(1)}$	$[s_{\text{train}} + 1, s_{\text{train}} + h]$
	n	$[1, s_{\text{train}} + (n - 1) \times h]$	1	$M^{(n)}$	$[s_{\text{train}} + (n - 1) \times h + 1, s_{\text{train}} + n \times h]$
OML iter	1	$[1, 1]$	1	$M^{(1)}$	$[2, 2]$
	n	$[n, n]$	1	$M^{(n)}$	$[n + 1, n + 1]$

Table 5.2 Evaluation functions for BML und OML

Evaluation function	Description
`eval_bml_horizon`	Section 5.1.3
`eval_bml_landmark`	Section 5.1.4
`eval_bml_window`	Section 5.1.5
`eval_oml_horizon`	Section 5.1.6

Table 5.3 Parameter for configuring the methods `eval_bml_horizon`, `eval_bml_landmark`, `eval_bml_window` and `eval_oml_horizon` from the package spotRiver. A tuple of two data frames is returned. The first contains the evaluation metrics for each batch of size `horizon`. The second contains the true and predicted values for each observation in the test data set

Parameter	Description
`model`	Model. Regression- oder Classification, e.g., a model from `sklearn`
`train`	Initial training data set
`test`	Test data set. Will be split into mini-batches of size 'horizon'
`target_column`	Column name of the target variable
`horizon`	Prediction horizon
`metric`	Metric, e.g., from `sklearn`
`oml_grace_period`	Only used for `eval_oml_horizon`. (Short) period, in which the OML-model is trained, but not evaluated. Startup phase

Example for the Method `eval_oml_horizon`

```python
from river import linear_model, datasets, preprocessing
from spotRiver.evaluation.eval_bml import eval_oml_horizon
from spotRiver.utils.data_conversion import convert_to_df
from sklearn.metrics import mean_absolute_error
metric = mean_absolute_error
model = (preprocessing.StandardScaler() |
        linear_model.LinearRegression())
dataset = datasets.TrumpApproval()
target_column = "Approve"
df = convert_to_df(dataset, target_column)
train = df[:500]
test = df[500:]
horizon = 10
df_eval, df_preds = eval_oml_horizon(
    model, train, test, target_column,
    horizon, metric=metric)
```

The method `plot_bml_oml_horizon_metrics` visualizes (1) the error (e.g., MAE), (2) the memory consumption (MB), and (3) the calculation time (s) for different models of ML on a given data set. The function takes a list of Pandas data frames as input, each containing the metrics for one model. The parameters of the method `plot_bml_oml_horizon_metrics` are shown in Table 5.4. Figure 5.5 shows the output of the metrics and Fig. 5.6 shows the residuals, i.e., the difference between the current (actual) and the predicted values.

```python
from spotRiver.evaluation.eval_bml import (
    plot_bml_oml_horizon_metrics,
    plot_bml_oml_horizon_predictions)
df_labels = ["OML Linear"]
plot_bml_oml_horizon_metrics(
    df_eval,
    df_labels,
    metric=metric)
plot_bml_oml_horizon_predictions(df_preds,
     df_labels,
     target_column=target_column)
```

Table 5.4 Parameters for configuring the method `plot_bml_oml_horizon_metrics`

Parameter	Description
`df_eval`	A list of pandas data frames containing the metrics for each model. Each data frame should contain an index column with the name of the data set and three columns with the names of the metrics: “MAE”, “Memory (MB)”, “CompTime (s)”
`df_labels`	A list of strings containing the labels for each model. The length of this list should match the length of `df_eval`. If `None`, numerical indices are used as labels. Default is `None`
`log_x`	A flag indicating whether to use a logarithmic scale for the x-axis
`log_y`	A flag indicating whether to use a logarithmic scale for the y-axis
`cumulative`	A flag indicating whether to plot the cumulative average error, as done in `plot_oml_iter_progressive()` and in River’s `evaluate.iter_progressive_val_score()` method. Time is shown as cumulative sum (not averaged). Since memory is calculated differently than in River’s `evaluate.iter_progressive_val_score()`, the peak memory value `_ , peak = tracemalloc.get_traced_memory()` is not aggregated. Default is `True`

5.2.2 *Methods for OML River*

The methods presented so far (in Sect. 5.2.1) are equally suitable for evaluating BML and OML models for three different data splits (1. horizon, 2. landmark and 3. window). In this section, the method `eval-oml-iter-progressive` is presented, which is specifically designed for the evaluation of OML models on a streaming data set. This is based on a method used in the River package. This makes it possible to compare the results with those of River. However, it cannot be used to evaluate BML models.

The method `eval-oml-iter-progressive` evaluates one or more OML models on a streaming data set. The evaluation is done iteratively, and the models are tested in each “step” of the iteration. The results are returned in the form of

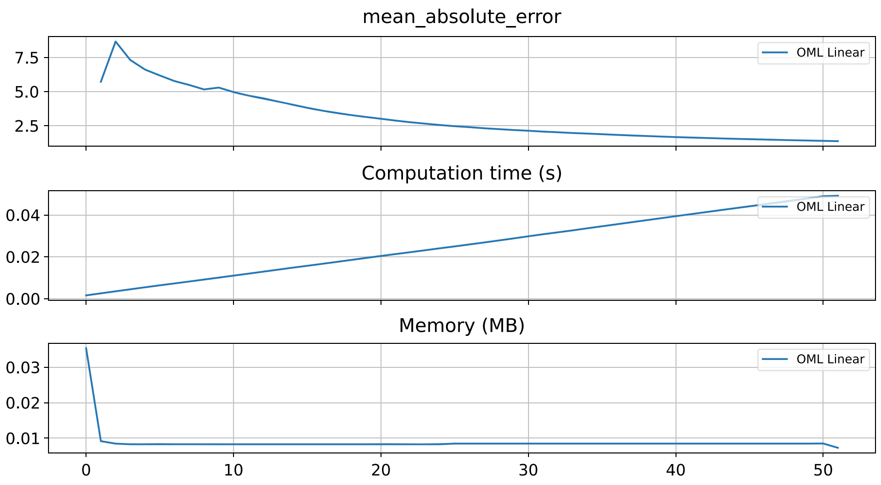


Fig. 5.5 Results of the method `plot_bml_oml_horizon_metrics`. Performance (here: MAE, computation time and memory consumption) of an OML linear model

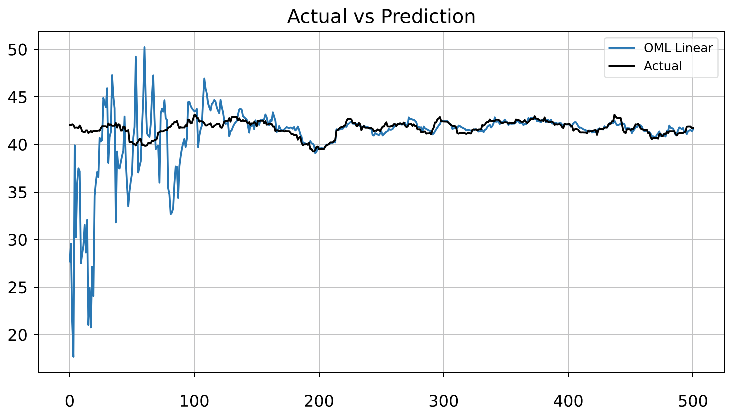


Fig. 5.6 Results of the method `plot_bml_oml_horizon_predictions`: Representation of the values predicted by the model and the ground truth ("Actual"). It becomes clear how the OML model approaches the ground truth over time and learns the underlying relationship

a dictionary with metrics and their values. Table 5.5 shows the parameters of the method `eval_oml-iter-progressive`.

The method `plot_oml_iter_progressive` visualizes the results based on the dictionary of evaluation results returned by `eval_oml_iter_progressive`. The visualization is based on the visualization in River.[2] Figure 5.7 shows the output.

[2] See (Incremental decision trees in River: the Hoeffding Tree case) [https://riverml.xyz/0.15.0/recipes/on-hoeffding-trees/].

Table 5.5 Parameter for the configuration of the method `eval_oml-iter-progressive` from the package spotRiver. A `dict` (dictionary) with the evaluation results is returned. The keys are the names of the models and the values are dictionaries with the following keys: `step`: A list of iteration numbers at which the model was evaluated, `error`: A list of weighted errors for each iteration, `r_time`: A list of weighted runtimes for each iteration, `memory`: A list of weighted memory consumption for each iteration and `metric_name`: The name of the metric used for evaluation

Parameter	Description
`data set`	A list of River.Stream objects containing the streaming data to be evaluated. If a single River.Stream object is specified, it is automatically converted to a list
`metric`	The metric to be used for evaluation
`models`	A dictionary of the OML models to be evaluated. The keys are the names of the models and the values are the model objects
`step`	The number of iterations at which results should be obtained. Only the predictions are considered, not the training steps. The default value is 100
`weight_coeff`	The results are multiplied by `(step/n_steps)**weight_coeff`, where `n_steps` is the total number of iterations. Results from the beginning have less weight than results from the end when `weight_coeff > 1`. If `weight_coeff = 0`, then the results are multiplied by 1 and each result has the same weight. The default value is `0`
`log_level`	The logging level to use. `0` = no logging, `50` = output only important information. Default value is `50`

```python
from river import datasets
from spotRiver.evaluation.eval_oml import (
    eval_oml_iter_progressive, plot_oml_iter_progressive)
from river import metrics as river_metrics
from river import tree as river_tree
from river import preprocessing as river_preprocessing
dataset = datasets.TrumpApproval()
model = (river_preprocessing.StandardScaler() |
         river_tree.HoeffdingAdaptiveTreeRegressor(seed=1))
res_num = eval_oml_iter_progressive(
    dataset = list(dataset),
    step = 1,
    metric = river_metrics.MAE(),
    models = {"HATR": model}
plot_oml_iter_progressive(res_num)
```

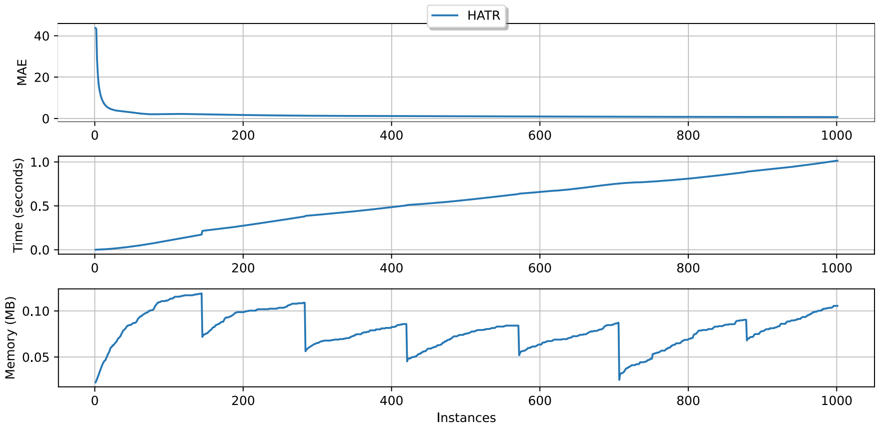


Fig. 5.7 Results of the method `plot_oml_iter_progressive`. The memory management of the HATR model is clearly visible

Notebook: Progressive Validation
An example of progressive validation can be found in the GitHub repository https://github.com/sn-code-inside/online-machine-learning shows how the delayed progressive validation can be applied using the moment and delay parameters in the method `progressive_val_score`. It is exploited that each observation in the data stream is shown to the model twice: once, to make a prediction and once to update the model when the true value is revealed.

The `moment` parameter determines which variable to use as a timestamp, while the `delay` parameter controls the waiting time before the true values are revealed to the model.

Tip
Further information on progressive validation can be found in the River package:
- river: Multi-class classification
- river: Bike-sharing forecasting

In addition, Grzenda et al. (2020) is worth mentioning, which deals with delayed, progressive validation.

5.3 Algorithm (Model) Performance

After the training and test data selection has been performed, the performance of the algorithm (or model) can be estimated. For this purpose, numerous metrics are available. Table 5.6 presents a selection of the metrics available in the package River. The selection of a suitable metric is crucial for the analysis of OML algorithms. For classification tasks, for example, accuracy is only a suitable metric if balanced classes are present. Kappa statistics (see Sect. A.4) are better suited for OML. Thomas and Uminsky (2022) give hints for the selection of suitable metrics.

The computation of the memory consumption is only simple at first glance. Programming languages such as Python or R perform memory management routines independently, which cannot be controlled by the user. For example, the garbage collector is not executed immediately after a call, since the program uses its own memory optimization routines, and it is sometimes more advantageous from its point of view not to delete the data. There are also many dependencies between individual objects, so they cannot simply be deleted even if this is desirable from the user's point of view. These remarks apply equally to BML and OML methods. According to our research (exchange with R experts), the estimation of the memory consumption in the programming language R is more difficult than in Python. This was one of the reasons why the studies presented in Chaps. 9 and 10 were carried out with Python. The module tracemalloc, introduced in Python 3.4, was used.

5.4 Data Stream and Drift Generators

Most software packages provide functions for generating synthetic data streams ("data-stream generators"). As an example, we have listed the generators available in the package `scikit-multiflow` in Sect. 5.4. We also describe the SEA synthetic dataset (SEA) and Friedman-Drift generators, which are used in many OML publications that examine drift.

5.4.1 Data Stream Generators in Sklearn

For example, the package `scikit-multiflow` provides the following data stream generators:

- Sine generator and anomaly sine generator
- Mixed data stream generator
- Random Radial Basis Function stream generator and Random Radial Basis Function stream generator with concept drift
- Waveform stream generator
- Regression generator.

Table 5.6 Metrics in the package River

`river` Class	Metric	Short Description
`accuracy`	Accuracy	Percentage of correct results
`balanced_- accuracy`	Balanced accuracy	Average of the recall obtained for each class, i.e., the average of the true positive rates for each class. It is used for unbalanced data sets
`CohenKappa`	Cohen's Kappa score	Computes the proportion of observations for which both classifiers predicted the same category and the probabilities that occur with a random prediction. See also Sect. A.4
`cross_entropy`	Cross Entropy	Multi-class generalization of the logarithmic loss
`f1`	F1	Binary F1 score
`fbeta`	Binary F-Beta score	A weighted harmonic mean between precision and recall
`fowlkes_mallows`	Fowlkes-Mallows Index	External evaluation method for determining the similarity between two clusters
`geometric_mean`	Geometric mean	Indicator of the performance of a classifier in the presence of class imbalance
`log_loss`	Binary logarithmic loss	Indicates how close the prediction probability is to the corresponding actual value. Also known as cross entropy
`mae`	Mean absolute error	Mean absolute error
`mcc`	Matthews correlation coefficient	Takes into account true and false positive and negative results. Also suitable for unbalanced classes
`mse`	Mean squared error	Mean squared error
`mutual_info`	Mutual Information between two clusterings	Measure of similarity between two labels of the same data
`precision`	Binary precision score	Measure of the classifier's ability to identify a sample as positive if it is actually positive
`r2`	Coefficient of determination (R^2) score	Ratio of explained variance to total variance
`rand`	Rand Index	Measure of similarity between two data clusters
`recall`	Binary recall score	Indicates how many of the actual positive cases were correctly identified as positive by the model
`roc_auc`	Receiving Operating Characteristic Area Under the Curve.	Approximation to the true ROC AUC
`silhouette`	Silhouette coefficient	Indicates how well an object fits to its own cluster
`smape`	Symmetric mean absolute percentage error	Accuracy measure based on relative errors
`WeightedF1`	Weighted-average F1 score	Computes the F1 score per class and then computes a global weighted average by using the support of each class

Tip
By sorting the observations, concept drift can be simulated (Bifet & Gavaldà, 2009).

5.4.2 SEA-Drift Generator

The SEA is a frequently cited data set. Its generator implements the data stream with abrupt drift as described in Street and Kim (2001). Each observation consists of three features. Only the first two features are relevant. The target variable is binary and positive (true) if the sum of the features exceeds a certain threshold. There are four threshold values to choose from. Concept drift can be introduced at any time during the stream by switching the threshold.

In detail, the SEA data set is generated as follows: First, $n = 60{,}000$ random points are generated in a three-dimensional feature space. The features have values between 0 and 10, with only the first two features (f_1 and f_2) being relevant. The n points are then divided into four blocks of 15,000 points each. In each block, the class membership of a point is determined by means of a threshold value τ_i, where i indicates the respective block. The threshold values $\tau_1 = 8$, $\tau_2 = 9$, $\tau_3 = 7$ and $\tau_4 = 9.5$ are chosen. In addition, the data is noisy ("We inserted about 10% class noise into each block of data.") by swapping 10% of the class memberships. Finally, a test set ($n_t = 10{,}000$) is determined, consisting of 2,500 data points from each block.

The Python package River provides the function `SEA` to generate the data. Figure 5.8 shows an instantiation of the SEA drift data.

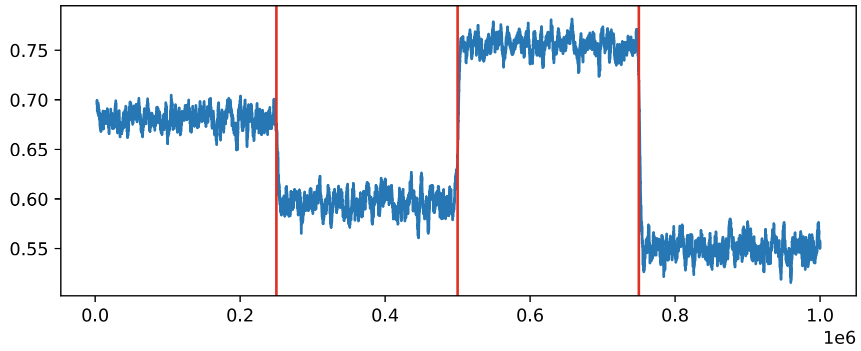


Fig. 5.8 SEA-Data with drift. Concept changes occur after 250, 000 steps

5.4.3 Friedman-Drift Generator

The Friedman-Drift generator introduced in Definition 1.8 is another generator that is frequently cited in the literature (Ikonomovska, 2012). It generates a data stream that simulates the characteristics of streaming data that occur in practice. The generator is implemented in River as `FriedmanDrift` and is used in Sect. 9.2.

5.5 Summary

The interleaved test-then-train (or prequential evaluation) is a general method for evaluating learning algorithms in streaming scenarios. Interleaved test-then-train opens up interesting possibilities: The system is able to monitor the development of the learning process itself and to diagnose its development itself. The delayed progressive evaluation is the subject of current research and enables a realistic analysis of complex changes in online data streams. In addition to quality, however, other criteria/metrics must be taken into account, which are imposed by data stream properties. The available memory is one of the most important constraints. Another aspect is time, because algorithms must process the examples as quickly as (if not faster than) they arrive.

Note
The experimental studies in Chap. 9 use the following three properties for the comparison of BML and OML methods:
1. performance,
2. memory consumption and
3. time consumption.

References

Bifet, A., & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII*, IDA'09 (pp. 249–260). Springer.

Grzenda, M., Gomes, H. M., & Bifet, A. (2020). Delayed labelling evaluation for data streams. *Data Mining and Knowledge Discovery, 34*(5), 1237–1266.

Ikonomovska, E. (2012). *Algorithms for learning regression trees and ensembles on evolving data streams*. Ph.D. Thesis, Jozef Stefan International Postgraduate School.

Street, W. N., & Kim, Y. S. (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*, KDD'01 (pp. 377–382). Association for Computing Machinery.

Thomas, R. L., & Uminsky, D. (2022). Reliance on metrics is a fundamental challenge for AI. *Patterns, 3*(5), 1–8.

Chapter 6
Special Requirements for Online Machine Learning Methods

Thomas Bartz-Beielstein

Abstract This chapter investigates whether Online Machine Learning (OML) algorithms require special steps and considerations compared to batch learning with respect to typical practice challenges such as missing data (Sect. 6.1), categorical attributes (Sect. 6.2), outliers (Sect. 6.3), imbalanced data (Sect. 6.4), or an extremely large number of variables (Sect. 6.5). Section 6.6 describes important aspects such as fairness (Fair Machine Learning (ML)) or interpretability (Interpretable ML) in the context of OML algorithms.

6.1 Missing Data, Imputation

Missing values in a data stream can often only be replaced (or “imputed”) using very simple strategies such as replacing missing values with zero, mean, median, or mode in existing OML frameworks. River provides the `StatImputer` method for this purpose. `StatImputer` replaces missing values with a statistic, e.g., the mean of the data already observed. During a call to `learn_one`, the `StatImputer` updates a statistic for each feature whenever a numerical feature is observed. When `transform_one` is called, each feature with a None value is replaced by the current value of the corresponding statistic.

Tip: Pipelines in River
Starting from version 0.19.0 calling learn_one in a pipeline will update each part of the pipeline in turn. Details can be found on https://riverml.xyz/0.21.0/releases/0.19.0/. The application of the methods learn_one and transform_one in pipelines is described in Sect. 8.1.4.

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

© The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2024
E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_6

Adequate imputations can be defined by providing a tuple for each feature to be imputed so that the statistics are conditioned on a specific feature. For example, a missing temperature value can be replaced by the average temperature of a specific weather condition (e.g., sun, rain, and snowfall). This allows data sets to be completed in which information on temperature, but not on weather conditions, is missing. Overall, however, it must be accepted that a much less mature methodology is available for imputation on data streams so far.

Further investigation is required into the different sensitivity to missing features in the data stream for OML and Batch Machine Learning (BML) methods: While the quality of BML methods usually deteriorates when a feature is removed, this does not lead to a significant deterioration in the tree-based OML method Hoeffding Tree Regressor (HTR). It is also interesting to note that the quality converges almost independently of the removed feature to a value. This value is comparable to the quality achieved by the BML method on the complete data set (Bifet et al., 2018).

Standard Data Preprocessing Methods
Similar to BML, OML provides various methods for data preprocessing, in particular for scaling (mean zero and standard deviation one) and normalization (range from a to b, typically between zero and one or minus one and one).

6.2 Categorical Attributes

In order to process categorical variables in a data stream, the procedures known from the classical BML context such as one-hot encoding are available. It is interesting for OML methods that new features or levels can be added during the process without having to retrain the model from scratch.

6.3 Outlier and Anomaly Detection

Standard OML methods for anomaly detection are the same as those known from BML methods. Simple thresholding methods such as constant thresholds or quantile thresholds are available.

Constant and Quantil Thresholds

If a constant threshold is used, but the model does not detect any values above this constant value, no observations of the anomaly class would be assigned at all.

In contrast, a percentile threshold would always classify values as anomalies: A percentile threshold of 95% will always classify 5% of the observations as anomalies.

These standard anomaly detection procedures do not require any information about the type or characteristics of the outliers. The anomaly detection algorithm only needs to learn what is normal in order to detect deviations (which can be larger or smaller than the normal values). Important: The model does not need to have seen a single anomaly in order to detect one.

6.3.1 Additional Anomaly Detection Methods for Time-Series Data

Besides these simple thresholding methods, there are a variety of other anomaly detection algorithms. Aggarwal (2017) describes subtle differences between the offline and online (streaming) settings, since in the former case the entire history of the stream is available for analysis, whereas in the latter case, only the stream up to the current time is available. In the offline environment, the retrospective view allows the discovery of outliers with more sophisticated models. Labels describing whether an anomaly is present may be available to monitor the anomaly detection process in both the time series and multidimensional outlier detection settings. In general, supervised methods almost always outperform unsupervised methods due to their ability to discover application-specific anomalies. The general recommendation is therefore to use supervised methods when they are available.

Time series can be interpreted as continuous data or discrete sequences. The concept of temporal continuity is defined differently for discrete data than for continuous data. For discrete data, a lack of ordering of the data values significantly affects the types of methods used for outlier analysis. For a more in-depth discussion, we refer to Aggarwal (2017).

6.3.2 One-Class SVM for Anomaly Detection

The one-class Support Vector Machine (SVM) is an unsupervised, binary classifier that learns a decision function for novelty detection: Classifying new data as similar to or different from the training set. The algorithm is based on the classification algorithm of the SVM. SVMs are able to generate a non-linear classification. The one-class SVM is an adaptation of the regular SVM: While in classical, supervised SVMs the classes (target variables) must be specified, the one-class SVM algorithm does not require this information. It acts as if all data were in a single class, which represents the normal state. Anomaly detection algorithms only have to learn one class. Everything that does not fit into this class is classified as an outlier.

6.3.3 Algorithms for Anomaly Detection in `river`

`river` provides two OML algorithms for anomaly detection: An online version of One-Class SVM (`OneClassSVM`) and an online version of Isolation Forests (`HalfSpaceTrees`).

6.4 Imbalanced Data

There are a number of standard approaches to dealing with imbalanced data (Korstanje, 2022). As an example, the "Synthetic Minority Oversampling Technique (SMOTE)" method can be mentioned here. SMOTE creates synthetic (or "fake") data points that are very similar to the data points in the positive class.

Notebook: Imbalanced Data
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning demonstrates how imbalanced data can be handled using

- Importance Weighting
- Focal Loss
- Undersampling of the majority class
- Oversampling of the minority class
- Sampling with a desired sample size
- Hybrid methods.

6.5 Large Number of Features (Attributes)

The method `SelectKBest` from the River package removes all but the k best features from a data set by calculating a similarity measure on the features. A "leader-board" stores the similarities of the features. Another method is `PoissonInclusion` (McMahan et al., 2013). This method decides randomly whether a new feature is added. A new feature is selected with probability p. The frequency with which a feature must be seen before it is added to the model follows a geometric distribution with expectation $1/p$. This feature selection method should be used when there is a very large number of features, of which only a few are meaningful, i.e., in situations with so-called “sparse features”. Finally, the `VarianceThreshold` method can be mentioned, which removes features with low variance.

Notebook: Large Number of Features
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning demonstrates the use of the method `SelectKBest` on a simulated data stream, the use of the method `PoissonInclusion` on the `TrumpApproval` data set, and the use of the method `VarianceThreshold` on a simulated data set.

6.6 FAIR, Interpretability, and Explainability

Chen et al. (2022) present a current overview of interpretability in ML. Zhang et al. (2021) describe the first online version of Random Forests with fairness constraints. This includes a mechanism to change the trade-off between accuracy and fairness so that it can be adapted to specific applications.

Halstead et al. (2021) describe the advantage of OML over BML with respect to explainability when concept drift occurs: In BML, adaptation to concept drift is achieved by deleting the current model and incrementally rebuilding it. Many OML algorithms additionally store previously created models and reuse them to adapt more efficiently when drift causes a previously known state to be assumed again (recurrent drift). Reuse provides improved classification performance over rebuilding a model and provides an indicator of the hidden state (the context K, as described in Definition 1.7) of the generative process. Changes in the context K cannot generally be observed while reusing a model is easy to observe. By linking context changes to an observable event, they become more transparent. Halstead et al. (2021) define the co-occurrence of models and contexts as the “transparency” of a system. The patterns of model reuse captured by a transparent system can be used for further performance and explanation benefits. Transparency can be used to improve performance and explainability at the same time.

Borchani et al. (2015) represent the context in terms of latent variables to generate explanations. They show that the changes in the latent variables are related to the changes in the real data. Real data could be economic factors such as unemployment. This approach could be used to explain the adjustments made by a system by referring to a change in the real world or to identify real factors that are important for the classification task, i.e., the factors that cause drift.

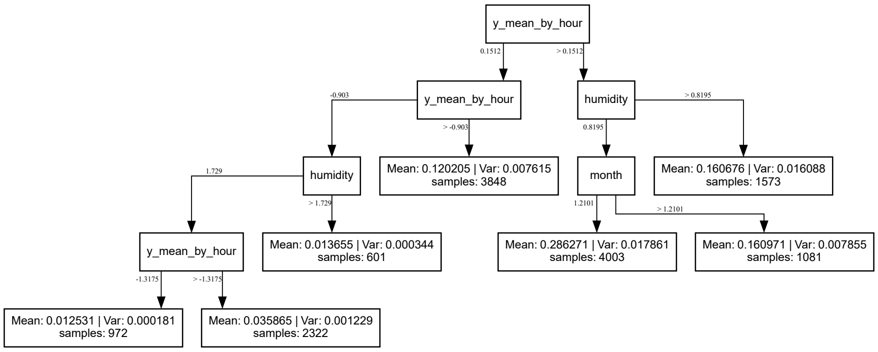


Fig. 6.1 Regression tree. The tree models the bike-sharing problem presented in Chap. 9. The hyperparameter `y_mean_by_hour` used in the root node has the greatest effect. This result suggests that the differences in the course of a day are greater than the differences between individual days of the week or the changes caused by weather conditions

Regression models have the great advantage that in many cases it is possible to explain the relationships on the basis of the model coefficients. However, if complex effects occur, such as higher-order interactions, this is no longer possible.

Trees have the advantage that they can be explained by the decision rules used to split the nodes. Thus, the entire tree can be viewed as a graph (dendrogram) by simply traversing the tree. The trees themselves provide an explanation of the model. Figure 6.1 shows a regression tree created during a hyperparameter tuning using the Sequential Parameter Optimization Toolbox (SPOT). A complete tuning process using additional tools for explainability is described in Chap. 10.

The hyperparameter tuning performed with Hyperparameter Tuning (HPT) tools such as SPOT facilitates explainability, since the complexity of the trees is usually also reduced. For complex models such as forests, it is often too difficult to look at all dendrograms. Estimates of "Variable Importance" are a suitable complement or even a possible replacement.

References

Aggarwal, C. C. (2017). *Outlier analysis*. Springer.

Bifet, A., et al. (2018). *Machine learning for data streams with practical examples in MOA*. MIT Press.

Borchani, H., et al. (2015). Modeling concept drift: A probabilistic graphical model based approach. In E. Fromont, T. De Bie, & M. van Leeuwen (Eds.), *Advances in intelligent data analysis XIV*. Lecture notes in computer science (Vol. 9385, pp. 72–83). Springer.

Chen, V., et al. (2022). Interpretable machine learning: Moving from Mythos to Diagnostics. *Queue, 19*(6), 28–56.

Halstead, B., et al. (2021). Recurring concept memory management in data streams: Exploiting data stream concept evolution to improve performance and transparency. *Data Mining and Knowledge Discovery, 35*(3), 796–836.

Korstanje, J. (2022). *Machine learning for streaming data with Python*. Packt.

McMahan, H. B., et al. (2013). Ad click prediction: A view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, New York, NY, USA* (pp. 1222–1230). Association for Computing Machinery.

Zhang, W., et al. (2021). FARF: A fair and adaptive random forests classifier. In *Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part II* (pp. 245–256). Springer.

Chapter 7
Practical Applications of Online Machine Learning

Steffen Moritz, Florian Dumpert, Christian Jung, Thomas Bartz-Beielstein, and Eva Bartz

Abstract This chapter addresses prerequisites, challenges, and potentials of applying Online Machine Learning (OML) methods in practice. These aspects are illustrated by means of domain-specific examples from different application fields. One of these surveyed application fields is official statistics (Sect. 7.1). Section 7.1.1 shows, that OML offers forward-looking potential for official statistics, but presently also comes with a lot of challenges. Especially compliance with quality assurance procedures (Sect. 7.1.2) and integration into existing process architectures (Sect. 7.1.3) prove to be major challenges. A survey about Machine Learning (ML) usage in German and other international statistical institutions shows that OML is currently still rather a niche topic in official statistics (Sect. 7.1.4). However, there are also domains, closely linked to official statistics, that either already feature OML applications or show promising potential for OML usage (Sect. 7.1.5). The second surveyed application field is the process of hot rolling in the steel industry (Sect. 7.2). In general, the process quality of hot rolling (Sect. 7.2.1) benefits from ML predictions (Sect. 7.2.2). However, because of being susceptible to drift, the complex hot rolling process cannot be adequately described without models that are continuously updated (Sect. 7.2.3). These characteristics make industrial hot rolling a suitable use case for the application of OML (Sect. 7.2.4). General aspects important for using OML in practice

S. Moritz (✉) · F. Dumpert
Federal Statistical Office of Germany, Wiesbaden, Germany
e-mail: steffen.moritz@destatis.de

F. Dumpert
e-mail: florian.dumpert@destatis.de

C. Jung
SMS Group GmbH, Siegen, Germany
e-mail: christian.jung@sms-group.com

T. Bartz-Beielstein
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz
Bartz & Bartz GmbH, Gummersbach, Germany
e-mail: eva.bartz@bartzundbartz.de

are briefly summarized in Sect. 7.3. These include reflections about model deployment (Sect. 7.3.1) and considerations regarding differences in required labor hours in comparison to Batch Machine Learning (BML) (Sect. 7.3.2).

7.1 Applications and Application Perspectives in Official Statistics

Official statistics are statistics compiled by official institutions on various topics, for example, demography, economy, environment, or health. The constitutive element for the production of these statistics is a statutory regulation. In Germany, entities such as the Federal Statistical Office, the statistical offices of the federal states, and other significant institutions, including the Deutsche Bundesbank, the German Environment Agency, and the Robert Koch Institute, are primarily responsible for the creation of official statistics.

Progressive digitization has greatly expanded the possibilities of official statistics in recent years. ML, in particular, plays a pivotal role in extracting additional insights and facilitating more rapid and efficient data processing. This efficiency augmentation is partially crucial to enable the utilization of emerging data sources, such as those associated with Big Data, within the purview of official statistics. However, akin to non-statistical sectors like industry, the integration of novel methods and technologies necessitates a progressive and gradual adaptation process.

This adaptation encompasses the assimilation of new methodologies into mature, established processes, the evaluation of potential impacts on facets such as data quality, and the revision of associated guidelines and procedures. Official statistics are presently undergoing this transformative phase of institutionalization and standardization. The discussion on quality aspects of this transformation can be found in the study by Yung et al. (2022). Consequently, ML solutions are increasingly becoming part of the repertoire at national statistical institutes, as evidenced by the work of Beck et al. (2018a).

As a special sub-field of ML, OML, yet again, offers new and extended possibilities, albeit with a corresponding increase in challenges. Detailed in Sect. 7.1.1, these extended possibilities seem the most promising in use cases that require the swift processing of extensive data sets. However, the sophistication and advancement of OML processes within the realm of official statistics cannot be equated with the foundation established for conventional batch-based ML.

Highlighted in Sect. 7.1.3, especially the "online aspect", i.e., the continual model updating, creates challenges when integrating OML into existing structures. Even in the context of traditional ML, it is necessary (and is currently being done) to determine the prerequisites for fulfilling the quality criteria of official statistics. OML introduces yet another layer of questions that necessitate resolution, as elucidated in Sect. 7.1.2.

This complexity partly explains why OML has been sparingly implemented in statistical institutions thus far. However, as discussed in Sect. 7.1.4, once these hurdles are effectively overcome, there is a multitude of intriguing potential applications that may be realized.

7.1.1 Potentials and Challenges

To ascertain the potential of OML for official statistics, it is pertinent to address the following queries:

1. What are the relevant aspects (possibilities, obstacles) for the use of OML from a practical perspective?
2. How can official statistics benefit from these new possibilities?
3. Which specific problems and obstacles arise in the context of official statistics?

A key aspect to consider here is the added value and extended possibilities that OML presents in comparison to the more established method of BML.

7.1.1.1 Relevant Aspects for Practical Utilization

As described in the previous book chapters, OML allows replacing a resource-intensive, computation-heavy batch run at the end of the data acquisition phase with a continuous model update that can already start during the acquisition phase. This capability facilitates model construction in scenarios where complete data set training would otherwise be unattainable (out-of-core). Moreover, OML is also useful in situations where it is necessary to adapt dynamically to slight shifts in the statistical properties of the data over time (concept drift). Furthermore, the sequential updating of the model during data acquisition allows the model's predictive capabilities to be already employed during the data collection process itself.

In summary, the following advantages hold practical relevance:

1. Balanced computational load (extremely long and intensive runs at the very end can be avoided).
2. Feasibility of processing extensive data sets (data that cannot be accommodated in the working memory in its entirety can be managed).
3. Dynamic adaptation to concept drift (manual model adjustments could become redundant).
4. Early usability of the model (predictions can be made well before the completion of data collection and processing).

However, it is evident that not every problem thus far addressed with traditional ML necessarily lends itself to an OML solution. Based on the experiments conducted in Chap. 9, a compromise in prediction quality is anticipated with OML as compared to BML. Furthermore, in terms of implementation and deployment (refer to Sect. 7.3), conventional BML methods typically involve less complexity and thus require less manual effort. Given that predictions can be generated while the data is being received continuously, it is crucial to ensure the model's quality at this stage. Unlike the BML approach, which mandates a single quality check towards the end, OML requires ongoing quality monitoring. Consequently, the design of the quality control process must be adapted accordingly.

In summary, the following disadvantages are relevant for practice:

1. Reduced prediction quality (when compared to BML).
2. Increased implementation efforts (more intricate implementation and maintenance).
3. Complex monitoring (continuous quality control is necessary).

Moreover, there exist scenarios where the online paradigm is fundamentally non-applicable. For instance, data may not always become sequentially available. In these circumstances, the deployment of OML is only reasonable for out-of-core issues, wherein the available hardware (memory) is inadequate for data processing.

Note: Practical Application
OML should be employed in applications where its strengths and additional capabilities over BML can be harnessed effectively. Implementing a traditional BML problem using an OML algorithm does not inherently confer any benefits.

7.1.1.2 Potentials of OML in Official Statistics

The significant potential offered by ML and Big Data has already been discussed elsewhere—for instance Schweinfest and Jansen (2021) or Radermacher (2018)—and both are integral to the future evolution of official statistics. However, the same is not yet true for OML. Therefore, our goal here is to assess the potential that OML could present for official statistics, beyond the basic capabilities of BML.

At first glance, the processes and data associated with official statistics surprisingly align well with OML. Firstly, statistics are published repeatedly at regular intervals, clearly a continuous process that could justify the efforts of adapting an OML model. Additionally, the data has very obviously a temporal nature: fresh data is generated prior to each subsequent statistics release. Concept drift can also be an issue, e.g., due to inflation or other developments between publication periods.

Yet, on closer inspection, it becomes evident that this is not the quintessential OML example of a constant influx of new data, possibly every second. Data is often published annually; for some statistics, it is also published quarterly or monthly. The single data points required for the statistics are often gathered individually, but are typically processed collectively afterwards (including overarching plausibility checks). During the steps in which OML would be relevant, data seldom arrives as a data stream, but rather as a comprehensive delivery or in the form of large partial deliveries.

Nevertheless, there exists potential for OML within official statistics:

Models for out-of-core data: This undisputed OML advantage may also be relevant for official statistics. New data sources in the big data domain, such as satellite data, but also existing collections of individual case statistical data, such as in tax statistics, can become too large for working memory. OML makes it possible to build models with the complete data, which can then be used to automate processing or to make predictions.

Time saving in single process steps: The statistical preparation process usually consists of several sub-steps. ML methods can be used in these sub-steps (for example, for automatic plausibility checks). An evaluation run over the entire data set can sometimes take several days, depending on the amount of data. If OML models are trained with subsets that are already available, the runtime required at the end can be reduced, thus also accelerating statistics production overall.

Continuous process steps: Currently, the statistics production process often necessitates that individual process steps are completed before the commencement of the next. In many cases, this is essential, for instance, all data must first be gathered to train a BML model. OML and online algorithms in general could allow individual data points to advance in the preparation process without waiting for all other data points to arrive and complete the current production step.

Continuous pre-releases and nowcasting: Nowcasting is a forecasting methodology that generates predictions for the present or recent past. Using already available observations, an unavailable variable is inferred. For example, the Gross Domestic Product (GDP) might not yet be finalized, but data correlated with it is already available as the underlying time period has elapsed. OML can be used here to create frequently updated nowcasts. This could help provide information quicker, enabling (policy) decisions to be made based on the most recent data.

In summary, OML offers considerable potential for official statistics, enabling statistical authorities to process larger volumes of data more rapidly and efficiently. OML is interesting both for process automation and for predictive methods in the form of nowcasts.

Nowcasting

Nowcasting is a method that generates predictions for the present or the recent past that have not yet been documented (Andreini et al., 2023). It stands apart from traditional forecasting techniques. Unlike classical forecasting, nowcasting does seldom project into the future, but instead focuses on estimates up to the edge of the present. In some instances, nowcasting projections do extend into the immediate future in the form of short-term forecasts.

In the field of economics, there are predictive models for real-time economic analyses that, for example, estimate the German GDP using nowcasting, even prior to the publication of actual quarterly results for the current quarter. This predictive model is employed by entities such as the German Federal Ministry for Economic Affairs and Climate Action (BMWK) and plays a crucial role in short-term economic analyses. It bridges time gaps resulting from the fact that crucial indicators such as GDP can only be provided with a significant delay. Typically, GDP figures for a quarter are not available until several weeks after the end of the quarter. Nowcasting estimates the GDP of the present in near real time.

7.1.1.3 Challenges of OML in Official Statistics

When implementing OML in official statistics, both general and specific problems can arise. A generic difficulty is that an OML model often does not deliver the same prediction accuracy as a BML model. It is also more challenging to integrate OML into the existing IT infrastructure, and there are fewer software solutions available, which are also less mature compared to their BML counterparts. However, there exist also challenges that are, at least in their manifestation, specifically relevant to official statistics.

These are the specific problems when trying to use OML in official statistics:

Compatibility with quality criteria: In order to foster comparability and preserve trust in official statistics, stringent methodological principles and criteria are employed. There are agreed-upon quality principles between the statistical producers, e.g., between the statistical offices of the German federal states, between EU member states, and also, to some extent, on an international level. As discussed in Sect. 7.1.2, OML complicates the fulfillment of some of these criteria.

Integration into existing processes: Maintaining consistently high quality necessitates defined and formalized processes. For official statistics, there already exists an official, time-tested framework that delineates individual process steps. As outlined in Sect. 7.1.3, OML, with its continuous online approach, partly follows a paradigm that is not readily compatible with existing processes described in the existing framework.

On the one hand, these challenges are characteristic for official statistics; on the other hand, there are other practical users who have these problems in a similar form. In industry, for example, a similarly conservative approach is often taken in the production of safety-critical components, and effects on defined quality principles are extensively evaluated.

7.1.2 Compatibility with Quality Criteria

Official statistics form an integral part of a society's information system and provide essential data to parliament, government and administration, the judiciary, the economy, and the public that later serve as the foundation for decision-making. To maintain trust in official statistics, extreme emphasis is put on the quality of publications. To ensure comparability and standardization, this is done in conjunction with national and international partners. As part of this standardization, quality frameworks have been defined at the EU level, which are further developed at the national level in the form of quality manuals (Quality Assurance Framework of the European Statistical System, 2019; Qualitätshandbuch der Statistischen Ämter des Bundes und der Länder, 2021).

This is a comprehensive catalog that defines, among other things, quality principles for processes and products. It also defines guidelines for quality control and organization.

ML itself is not explicitly mentioned in the quality manuals; for this, there are separate documents, coordinated in work groups and committees, that have derived requirements for ML solutions from the overarching quality requirements.

For example, in Yung et al. (2022), a document resulting from the UNECE High--Level Group for the Modernisation of Official Statistics (HLG-MOS), the requirements for ML algorithms are defined as follows:

Explainability: Explainability is defined as the ability to understand the logic underlying the algorithm used in prediction or analysis, as well as the resulting outcomes. Explainability is greatly facilitated by illustrating the relationship between input and output variables and providing necessary information about the methodology underpinning the algorithm.

Accuracy: The accuracy of statistical information refers to the extent to which it accurately describes the phenomena it is intended to measure. That is, it represents the proximity of computations or estimates to the exact or true values that the statistics are supposed to measure.

Reproducibility: At a fundamental level, reproducibility is defined as the ability to replicate results using the same data and the original algorithm. This is referred to as "method reproducibility". At a higher level, it is defined as achieving consistent results from new studies using the same experimental methods ("reproducibility of results") or similar results using different study designs, experimental methods, or analytical decisions ("inferential reproducibility").

Timeliness and punctuality: Timeliness is defined here as the time required to create a result, from conceptualization to the creation of the algorithm, processing, and production. A distinction should be made between the timeliness of development and production, with the former generally taking longer than the latter.

Cost-effectiveness: Cost-effectiveness is defined as the degree of effectiveness of the results relative to their costs. It should be noted that the total costs for carrying out the work, including fixed costs such as infrastructure and staff training, and running costs such as production costs, should be taken into account.

Saidani et al. (2023) add robustness as another dimension.

One positive point that can surely be attributed to OML is its *timeliness*. OML can potentially deliver results more promptly than BML procedures.

The point of *cost-effectiveness* can fall both in favor of and against OML. By dividing the computational steps over a longer period of time, it might become possible to work with hardware with lower maximum computational capacity. However, the sum of the required computational time over the entire process (for a comparable model class) is higher. In terms of personnel, arguments can be made in both directions. On the one hand, OML models are more complex to operate and test. On the other hand, they may need to be adjusted less frequently due to their ability to react to concept drift. Monitoring the models requires more effort for OML, as they must be continuously monitored, whereas for BML only the final run needs to be checked. There are certainly other cost factors, but in general, the costs are very problem-specific and can hardly be generalized.

Explainability is worse in OML than in conventional BML. This is partly because the fundamental process (for the same model class) is more complex to understand. It is also due to the fact that there is not as much tooling available for OML as there is for BML and there is a lack of evaluation packages.

Accuracy must also be compromised. Looking only at predictions for a consolidated final data set alone, the expected accuracy in OML is likely to be lower. However, this should always be checked on a case-by-case basis.

Reproducibility is another issue—in order to be able to reproduce the results of the OML algorithm again, not only the original data but also the order in which they entered the OML algorithm would have to be stored.

Many of the problems can be addressed with measures or could be significantly mitigated with better available supplementary software for OML (for instance, in terms of reproducibility and explainability). However, it also shows that the software and package ecosystem around OML is not yet as extensive.

Essentially, no hard exclusion criterion has been found that would make OML incompatible with the quality guidelines of official statistics. OML has, in principle, similar (albeit sometimes more pronounced) challenges regarding compliance with quality guidelines as BML. When testing the criteria, it is important to consider the individual case, i.e., to check each new BML or OML application separately.

Summing up across all ML quality dimensions relevant to official statistics, OML performs slightly worse overall than pure BML. This underscores our finding from the beginning of the chapter that OML is by no means a standard BML substitute. OML should be considered for applications where it can provide its specific advantages, such as nowcasting.

7.1.3 Embedding in the Statistics Production Process

In order to achieve consistent quality and produce comparable results, defined and formalized processes are required. The statistical production process at the Federal Statistical Office of Germany is oriented on the Generic Statistical Business Process Model (2019), a standard framework developed by the United Nations Economic Commission for Europe (UNECE). The GSBPM describes statistical production in a process-oriented manner. It describes the complete statistical production process, from concept design to evaluation.

In a similar way to the quality frameworks, it provides the framework within the European Statistical System and is adapted by the National Statistical Institutes to their needs. For Germany, the GSBPM is concretized in the more detailed GMAS—Geschäftsprozessmodell Amtliche Statistik (Blumöhr et al., 2017).

In a survey conducted in 2017/2018 that investigated the use of ML methods across several statistical institutions, Beck et al. (2018a) identified the primary sub-processes of the GSBPM where ML solutions are extensively employed. As anticipated, these are predominantly the Collect, Process, and Analysis processes, wherein data processing that can be partially automated occurs. In general, the overall process is mostly designed in such a way that one sub-process has to be completed before the next sub-process can start. This has clear advantages in terms of quality control because it is possible to check step-by-step whether the outputs of the sub-process meet the specific quality criteria. However, it is a challenge to integrate OML processes into this evolved framework (Fig. 7.1).

When data primarily progresses in batches from one sub-process to another, OML cannot maximize its potential, apart from instances of out-of-core computing.

Advocating for a seamless transition between sub-processes, wherein data points from the same statistical survey can be in different sub-processes at a given point in

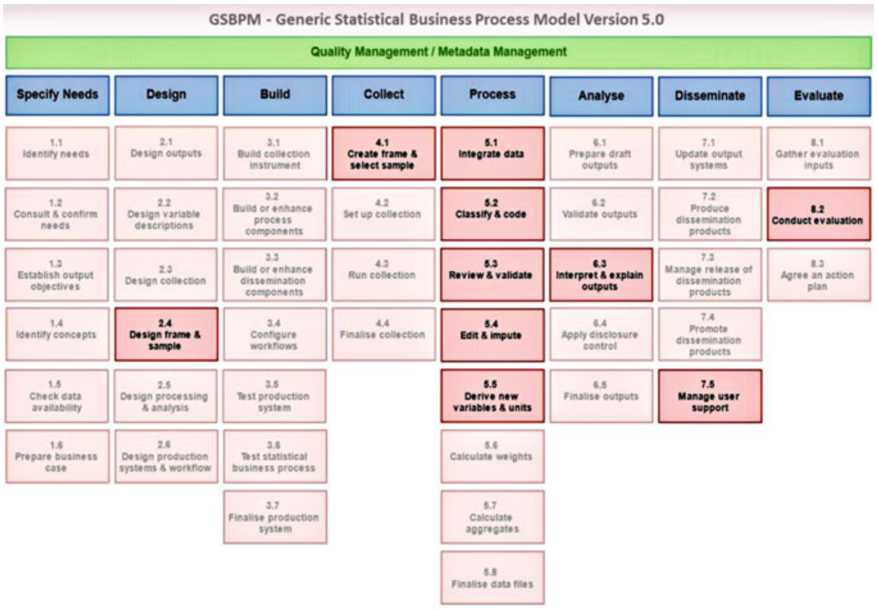

GSBPM - Generic Statistical Business Process Model Version 5.0
Quality Management / Metadata Management
Specify Needs
Design
Build
Collect
Process
Analyse
Disseminate
Evaluate
1.1 Identify needs
1.2 Consult & confirm needs
1.3 Establish output objectives
1.4 Identify concepts
1.5 Check data availability
1.6 Prepare business case
2.1 Design outputs
2.2 Design variable descriptions
2.3 Design collection
2.4 Design frame & sample
2.5 Design processing & analysis
2.6 Design production systems & workflow
3.1 Build collection instrument
3.2 Build or enhance process components
3.3 Build or enhance dissemination components
3.4 Configure workflows
3.5 Test production system
3.6 Test statistical business process
3.7 Finalise production system
4.1 Create frame & select sample
4.2 Set up collection
4.3 Run collection
4.4 Finalise collection
5.1 Integrate data
5.2 Classify & code
5.3 Review & validate
5.4 Edit & impute
5.5 Derive new variables & units
5.6 Calculate weights
5.7 Calculate aggregates
5.8 Finalise data files
6.1 Prepare draft outputs
6.2 Validate outputs
6.3 Interpret & explain outputs
6.4 Apply disclosure control
6.5 Finalise outputs
7.1 Update output systems
7.2 Produce dissemination products
7.3 Manage release of dissemination products
7.4 Promote dissemination products
7.5 Manage user support
8.1 Gather evaluation inputs
8.2 Conduct evaluation
8.3 Agree an action plan

Fig. 7.1 Use of ML algorithms in sub-processes of GSBPM (grayed out red means no ML use in that sub-process). *Source* Beck et al. (2018a). Based on Generic Statistical Business Process Model (2019). This work is licensed under the Creative Commons Attribution 4.0 International License. Attributed to: *United Nations Economic Commission for Europe (UNECE), on behalf of the international statistical community*

time, would help to leverage the benefits of OML. However, this would also require a considerable paradigm shift in many entrenched statistical procedures. Which means some of the current quality control processes would have to be completely redesigned.

A transition in the processes, geared towards a more OML-friendly application, does not appear imminent at this time. Given the challenges mentioned in integrating OML within the existing standardized process frameworks, OML is set to persist as a specialist instrument for the near future. At present, OML can truly demonstrate its potential only in a limited number of areas where individual sub-processes permit a fluid transition to subsequent phases. Additionally, OML might prove advantageous in applications operating outside the constraints of formalized procedures, for instance, in the realm of experimental statistics and digital data products.

7.1.4 (Online) Machine Learning Applications in Statistical Institutions

The two documents by Dumpert and Beck (2017) and by Beck et al. (2018b) were used as a starting point for the research. These contain an inventory of the use of ML procedures in the Federal Statistical Office of Germany and in national and selected international statistical institutions in 2017.

The documents reveal an extensive implementation of batch learning procedures across the surveyed institutions. Nevertheless, it is noteworthy that none of the projects incorporated online learning procedures. A subsequent internet investigation did not disclose any signs on the websites of the surveyed institutions that suggest any shift in this practice.

A literature review also failed to produce any evidence concerning the application of online learning procedures in relation to the participants of the 2017 study.

Table 7.1 incorporates all the institutions that implemented BML in at least five projects (at different stages from conceptualization to production at the time of the survey). These statistical institutions could thus be characterized as “early adopters” of ML methodologies. If a subsequent survey were to be conducted, the aforementioned institutions should be the primary contacts.

In summary, we are not aware of any example from official statistics so far in which OML is used in the classical statistics production process.

As described in Sect. 7.1.5, there are some applications that are at least related to official statistics and, depending on the task definition, could certainly also be performed by a National Statistical Institute.

As expounded upon in Sect. 7.1.3, incorporating OML into the traditional statistical production process is not a straightforward task. Particularly in the field of nowcasting, OML holds considerable potential, offering the ability to continuously integrate new data into the model and to make predictions at any given point in time based on the current state of training. OML could also provide added value in less formalized processes involving new digital data or real-time indicators. In this con-

Table 7.1 Institutions with at least five BML projects in 2017

Country	Institution	Number of BML projects 2017
Canada	Statistics Canada	36
Germany	Federal Statistical Office of Germany	31
Germany	GESIS	16
USA	U.S. Bureau of Labor Statistics	11
New Zeeland	Stats NZ	9
Germany	IAB	8
USA	U.S. Department of Agriculture NAA	7
Switzerland	Swiss Federal Statistical Office	6
Australia	Australian Bureau of Statistics	6
Germany	Deutsche Bundesbank	5
France	INSEE	5

text, OML could expedite processing, provided that the constraint of data primarily transitioning between sub-processes in batch form can be eliminated.

Conclusion: OML Applications in Official Statistics
OML does have potential in the area of official statistics, but this is severely limited by the fact that current processes in statistics production are based on batch processing, for understandable reasons. Thus, the two main challenges are: 1. integration into existing processes, and 2. compatibility with quality criteria.

So far, we are not aware of any examples from official statistics where OML is used in the classical statistics production process. In our opinion, the greatest short-term application potential for OML is hidden in the marginal areas of classical official statistics. Conceivable applications are real-time indicators, nowcasting and novel applications with experimental digital data.

7.1.5 *Other Applications with Reference to Official Statistics*

In a literature search on applications of online-learning methods related to official statistics, the studies summarized in this section were found. While not all of these use cases have been implemented with classical OML methods, they are basically problems for which OML would be very well suited.

Alvarez et al. (2022) describe a tree-based incremental learning model to estimate house prices using publicly available information on geography, city characteristics, transportation, and properties for sale. Previous ML models capture the marginal impact of property characteristics and location on prices by using large data sets for training. In contrast, this scenario is limited to small data sets that become available on a daily basis. Therefore, the OML model learns from daily city data and uses incremental learning to provide accurate price estimates every day. The results show that house prices are strongly influenced by the characteristics of the city and its infrastructure, and that incremental models efficiently adapt to the nature of the house price estimation task.

Pandemic forecasting is another application area where OML techniques can be used. Especially during the COVID-19 pandemic, predictions were important to derive political decisions and policies. Suárez-Cetrulo et al. (2021) compare algorithms for BML, such as Long Short-Term Memory (LSTM), with online incremental ML algorithms to adapt to daily changes in disease spread and predict future COVID-19 cases. The study conducted three experiments, where the first two used a static hold-out approach and the third used a prior evaluation. The results showed that incremental methods are a promising approach to adapt to changes in disease over time, as they are always up-to-date and have a much lower computational cost than other techniques, such as LSTMs. Kimura et al. (2022) introduce EpiCast, a data mining and forecasting method based on nonlinear differential equations. EpiCast is effective, adaptive, and scalable, working with large epidemiologic data streams to capture global trends and site-specific patterns. The method incrementally monitors current dynamic patterns and identifies abrupt changes in data streams. In experiments on real data sets, EpiCast outperformed existing state-of-the-art methods in accuracy and execution speed.

Incremental learning methods can be applied to sentiment prediction for elections. Chatterjee and Gupta (2021) developed a scalable, real-time, REST API-based system that uses bilingual, emoji-based, multi-class sentiment classification. The system includes an incremental learning framework to facilitate phased learning when system accuracy decreases.

In crisis situations, quickly available information about economic development is crucial for decision-making. Senftleben and Strohsal (2019) describe nowcasting as a real-time indicator for business cycle analysis, with the BMWK's nowcast model producing daily technical forecasts for the current quarter. Andreini et al. (2023) develop a nowcasting model for the German economy that outperforms alternatives and produces forecasts for GDP and other key variables. Steinberg et al. (2021) note that many relevant economic data are published with a lag, so higher-frequency indicators are evaluated to assess current developments. These include monthly statistics on production or the order situation, as well as survey-based sentiment indicators. Other tools for timely evaluation are econometric nowcasting models or flash estimates, which automatically evaluate current indicators and can use them to calculate a purely technical forecast of economic development in the current quarter.

Advances in Artificial Intelligence (AI) and digitization have led to new opportunities for macroeconomic analysis, with the German Federal Ministry for Economic

Affairs and Climate Action (BMWK) exploring the potential of statistical methods for observing and projecting economic development in Germany. For example, special statistical methods can be used to extract valuable new information from large, initially often unstructured data sources (Big Data). Aparicio and Bertolotto (2020) introduce online price indexes to forecast the consumer price index, significantly outperforming Bloomberg surveys and benchmark statistical forecasts. Online price indexes can anticipate changes in official inflation trends more than one month in advance. The authors' baseline forecasts specification also outperforms benchmark statistical forecasts for Australia, Canada, France, Germany, Greece, Ireland, Italy, the Netherlands, the United Kingdom, and the United States (U.S.). Similarly, the quarterly forecast for the U.S. inflation rate calculated in Aparicio and Bertolotto (2020) consistently outperforms the Survey of Professional Forecasters (SPF).

7.1.6 Summary: OML in Official Statistics

In general, OML has been relatively underexplored in the realm of official statistics production organizations. As illustrated in Sect. 7.1.5, there exist a few promising applications in domains closely related to official statistics. Yet, it is clear that OML provides notable advantages such as accelerated model updating or simplified handling of substantial volumes of incoming data. However, the use case must also be appropriate for an OML application. As delineated in this chapter, only a subset of problems that until now have been resolved with ML are genuinely suitable for an OML implementation. Moreover, beyond the apparent predictive quality trade-off, practitioners are faced with numerous other challenges.

OML is not just a new, alternative ML algorithm that simply replaces previous algorithms. Rather, it is a paradigm shift—from batch processing to data stream processing—a shift that offers new possibilities, but also requires a corresponding realignment of processes. This introduces a range of difficulties and is likely the reason why we observe so few example applications in productive use.

Making changes to a mature, established process structure, which is mostly based on step-by-step processing of complete batches, is not easily possible and rather a long-term effort. Furthermore, OML is difficult to integrate into the existing IT landscapes. Currently available standard software is often not designed for online processes and furthermore, the required OML algorithms are often just not available.

These are difficulties that basically all official statistics institutions have to overcome. For existing applications, a changeover is therefore difficult and costly. OML applications will likely emerge in newly conceived applications, provided the application scenario is appropriate. Nevertheless, we believe that OML will increasingly be considered among the possible solutions to evaluate in future processes. We particularly see the potential for OML applications in official statistics for applications with high-frequency, new digital data, real-time indicators, and nowcasting.

7.2 Industrial Application of OML in the Context of Hot Rolling

7.2.1 Hot Rolling

Hot rolling refers to a process where heated material is plastically deformed between two or more rolls in order to produce a thinner product. Today's rolling mills require a huge variety of different materials and geometries to be processed within close tolerances. An overview of different plant layouts and a detailed process explanation are given in Ginzburg and Ballas (2000), Hinkfoth (2003), Hensel and Spittel (1978), and Weber (1973). The process is shown in Fig. 7.2.

In order to reach these tolerances, the prediction of the process behavior and control of process parameters should be as accurate as possible. Process prediction is done with various physical-based software models that are interconnected. These models have the task of predicting important process parameters. The term "models" hereby refers to software that describes the physical process as accurately as possible. This software is normally developed by implementing the physics behind the process. Some assumptions made in the software are simplifications, the result of Finite Element Method (FEM) studies, or data-driven approaches.

All of those simplifications are made because, even with the power of today's CPUs, the calculation would take too long. The calculations for a single product may change depending on the current status of the product and the available information.

Fig. 7.2 Hot rolling steel. Attribution: Erik Charlton from Menlo Park, USA, CC BY 2.0 https://creativecommons.org/licenses/by/2.0, via Wikimedia Commons

The accuracy of the physical models that describe the rolling process is of major importance. According to their predictions, the mill will be operated. Minimizing any residual error therefore is the main task for the online models.

During the production process, several measurements are collected, and the predictions of the models are updated. Typical sources of these measurements are pressure transducers, drive measurements, optical measurements, or even isotopic measurements. Some of these predictions will directly influence the further processing of the same product.

A typical example is a width measurement, which takes place at multiple process steps. The measurement is used by the process models to incrementally update the prediction. This new prediction will then be used within the production process. Therefore, the update should be done as quickly as possible. A typical requirement would be to have the updated prediction available within one second or even less.

Other predictions, like, e.g., geometry or temperature measurements on the final product, are used to update the prediction for the next products. Here, the requirement would be to have an updated prediction available whenever the new prediction can be used for the next product. This depends on the plant configuration and may range from a few seconds up to a few minutes.

7.2.2 Machine Learning in Hot Rolling

Since the accuracy of the prediction is directly correlated to the quality of the product, it is important to use every measurement that is available and which may help to reduce any prediction error. As mentioned in Sect. 1.2, the BML has multiple disadvantages. Especially important in the hot rolling process are drift, memory limitations, and the occurrence of new data.

All material grades that are rolled are described by their corresponding chemical elements. Nowadays, hundreds of different material grades are known. The physical models are using databases for the description of material properties like resistance, density, heat transfer, or heat conductivity. This is done separately for each material grade. Since the prediction of the material is a key element for the process, most residual errors of the process models show a dependency on the rolled material grade. A gentle introduction to the material model and its task is given in Jung (2019). New material grades are continuously developed and may be introduced at any time. Although the initial prediction of those materials might be wrong, it should automatically improve over time. A typical solution for this problem is to use online learning models separately for each material grade.

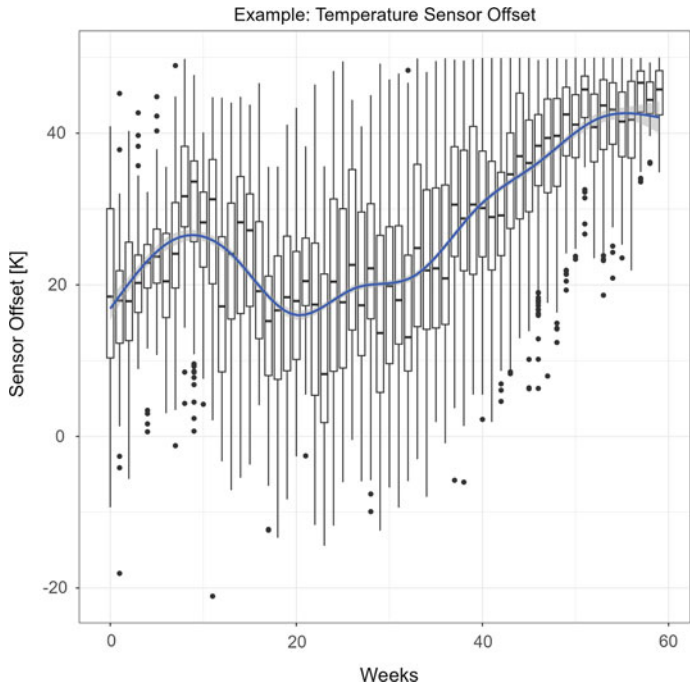


Fig. 7.3 Example of the occurrence of drift caused by bad maintenance within hot rolling. Sensor offset plotted over time. The sensor offset shows some dynamic behavior and starts increasing continuously after a certain period

7.2.3 Drift in Hot Rolling

Dynamic changes occur inherently in real-world applications. Changes can affect the distribution over time of the dependent variables, the output variable, or both. The drift may occur from various sources, like sensor drift, wrong calibration, or bad maintenance. New and unknown materials might be introduced to the process without any prior knowledge. Therefore, the process models are currently using OML wherever possible to reduce the residual errors.

Figures 7.3 and 7.4 show typical examples of drift that may occur in the hot rolling process:

- In Fig. 7.3 a temperature measurement done with an optical camera system is compared to the contact temperature measurement after rolling in the context of aluminum rolling. The difference between both measurement systems is shown as a boxplot over some production weeks. It can be seen that over several weeks, the deviation starts to increase. After the maintenance was done, the deviation immediately decreased to a normal range (not shown in the plot).

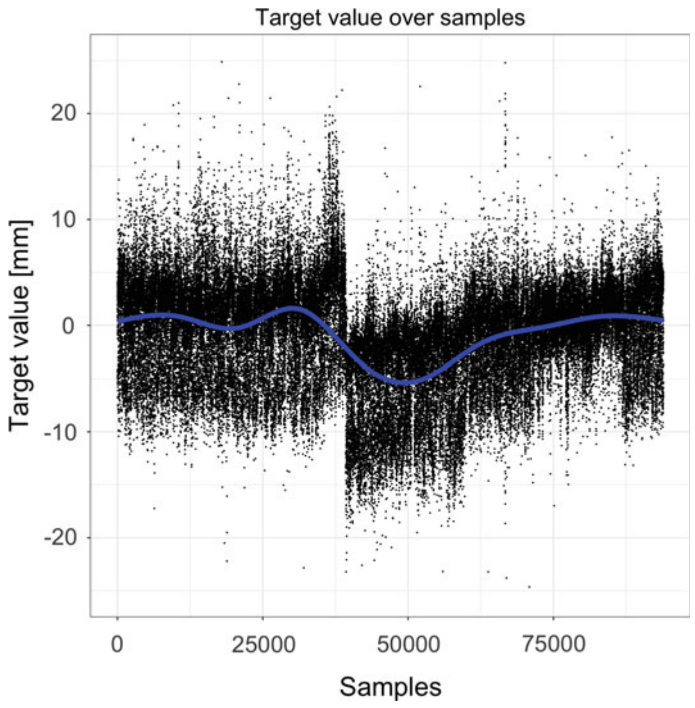


Fig. 7.4 Example for the occurrence of event-based drift within hot rolling. Target value (roll diameter) plotted over time. The roll diameter was measured incorrectly. This resulted in a prediction offset at the time when this roll was mounted in the mill

- Figure 7.4 shows another common problem that was observed in a so-called roughing mill. The plot shows the deviation of the measured width and the target width. At sample 40,000 an abrupt drift can be observed. This could, e.g., be caused by a wrongly measured roll diameter that is used in the automation system, or also due to maintenance or other reasons.

While the first example can be classified as regular label drift, the second example manifests as concept drift, see Sect. 1.2.2. The spread of the rolling process is strongly dependent on the width reduction and on the occurrence of drift; the same width reduction will result in a different spread because of the error within the roll diameter or calibration.

7.2.4 Application of OML in Hot Rolling

The complex hot-rolling process cannot be adequately described without models that are continuously updated. There are various process models available for the

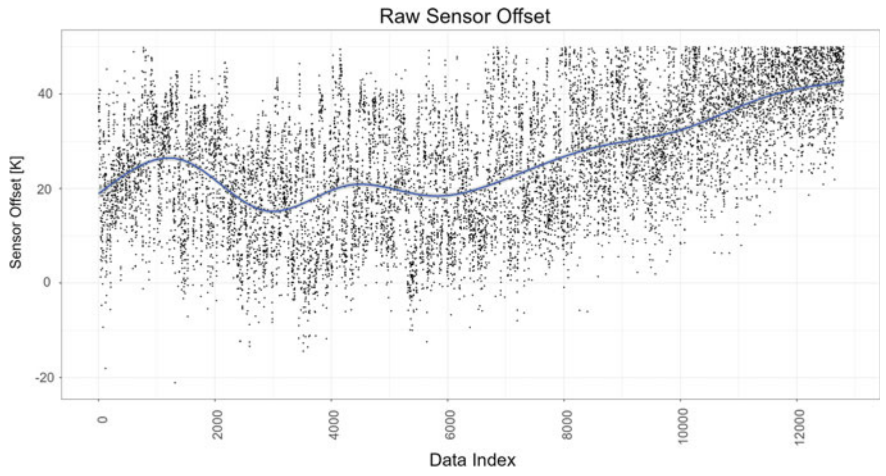


Fig. 7.5 Performance of a typical online algorithm that is used for compensation of drift. For each material grade in the data set a sliding average is used. Shown is the MAE over the whole data set. The first 4,000 data points were used to initialize the algorithm and are not used for the determination of prediction performance

description of the process. Only a subset of them typically learn incrementally. The most important prediction parameters are the following:

1. Roll Force
2. Roll Torque
3. Temperature
4. Width
5. Thickness

As shown in Fig. 7.3, the temperature sensor may be affected by drift. Figure 7.5 shows the raw data of the sensor, and Fig. 7.6 shows the performance of a simple online algorithm that compensates this drift. Together with the raw sensor drift, two categorical variables and three numerical variables are used by the online algorithm. The raw sensor data was limited to values between -50 and 50 and the drift starts approximately at sample number 6,000.

The first 4,000 data points were used for training and are therefore neglected in the determination of the Mean Absolute Error (MAE). It can be seen that the online algorithm is compensating this drift quite well, i.e., the performance in terms of MAE is stable. If the raw sensor data had not been limited to a maximum of 50 K then the drift would be more clearly visible and even higher. Additionally, an even better online correction of the drift would be expected.

Figure 7.7 shows the difference between the measured and the target width and represents a subset of the data shown in Fig. 7.4. This prediction error, together with multiple objective variables, is used to reduce the residual error. In Fig. 7.8 the performance in terms of MAE of a similar algorithm that was used within the real-

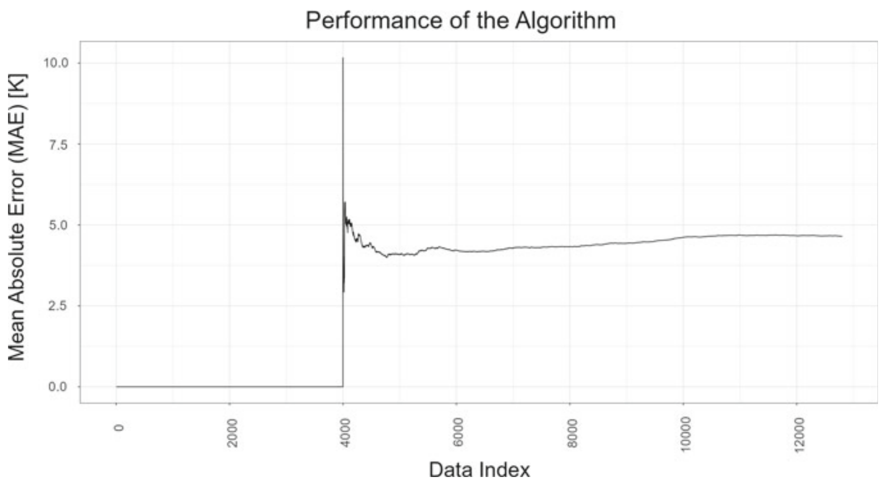


Fig. 7.6 Performance of a typical online algorithm which is used for compensation of drift. For each material grade in the data set a sliding average is used. Shown is the MAE over the whole data set. The first 4,000 data points were used to initialize the algorithm and are not used for the determination of prediction performance

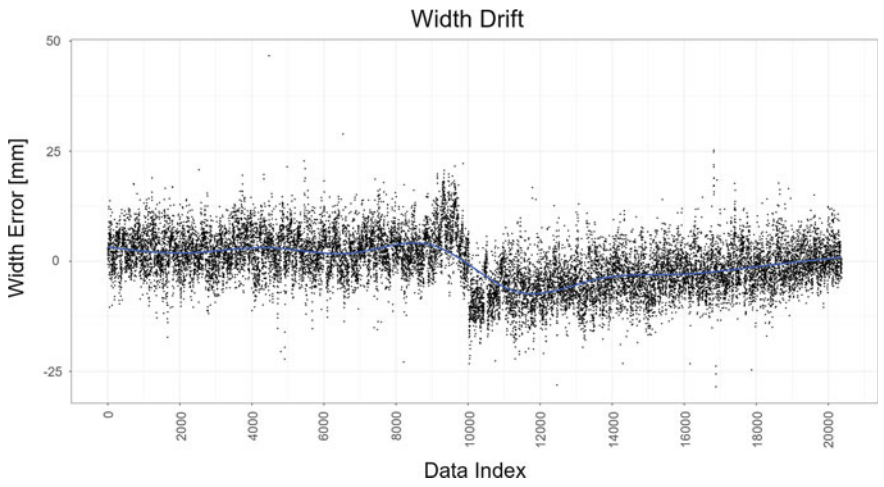


Fig. 7.7 Width prediction error over time. The data represents a subset of the data shown in Fig. 7.4. They are used to train a simple online algorithm, which should reduce the residual errors

world process is shown. The algorithm was initialized using the first 7,000 samples, and the corresponding MAE was set to 0 until sample number 7,001.

The drift that can be seen at approximately sample number 8,000 was compensated quite well, and the performance almost remained constant within some limits. This is important for the real process since the plant is still able to produce sellable products.

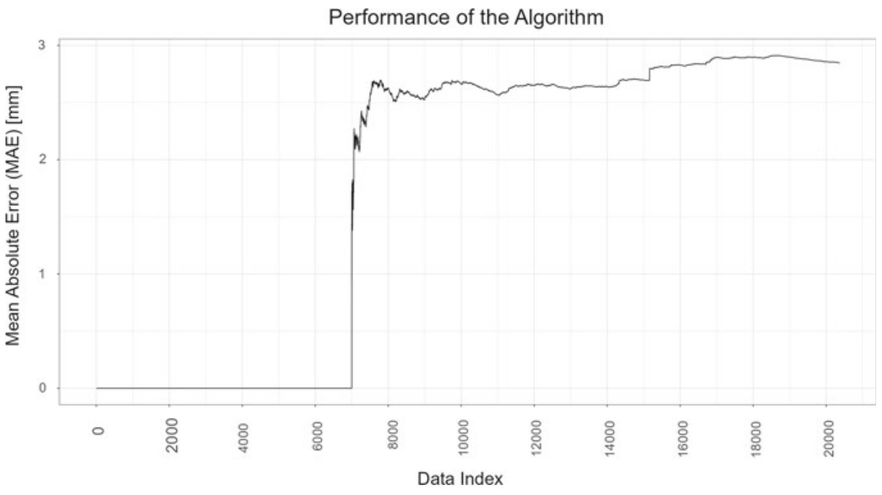


Fig. 7.8 The performance of the online algorithm, which was initialized with the first 7,000 samples, is shown in terms of MAE

7.2.5 Summary: OML in Hot Rolling

Several problems may arise within the hot rolling process that require adjustments to prediction algorithms. Therefore, online algorithms are playing an important role. Batch learning may be possible for some scenarios, but due to the fast update requirement, iterative learning, i.e., OML, should be used and is already established.

Most often, ML algorithms in hot rolling have to deal with categorical and continuous variables. A common approach is to use a separate model for each categorical variable or a dummy coding strategy for those variables. Other strategies for handling categorical variables are described in Jung (2019).

Currently, only simple algorithms like linear regression or support vector regression are used. The reason for this is the requirement for having a fast and robust prediction available as soon as possible. If more complex algorithms are developed and can be used for OML, it would be interesting to see their performance in comparison to the traditional algorithms.

7.3 Summary: Aspects of OML Implementation in Practice

Insights into the practical deployment of OML processes remain scarce, rendering the juxtaposition with BML methods a challenging endeavor. This challenge is further amplified due to the unique nature of each application, which can vary significantly across different companies or application scenarios. Despite these complexities, this comparison is of substantial interest. Moving forward, we delve into various aspects of implementing OML methods in practice, taking into account the distinctive characteristics of each application.

7.3.1 Recommendations for the Implementation Process

When considering the OML implementation process, the problems described in Sect. 1.2 that arise when using BML procedures on online data provide a good starting point.

First, it is important to determine whether OML procedures should be considered at all. Is there a need because of one or more of the following problems?

1. Memory requirements
2. Drift
3. Unknown data
4. Accessibility of the data.

If this need exists, consideration should then be given to how the OML process can be implemented and whether a successful outcome is realistic. To do this, the following points should be considered:

1. Requirements for the outcome
2. Estimation of resources
3. Properties of the data
4. Properties of the algorithms (hyperparameters).

These four points are explained in the following: Regarding the accuracy of the results, it should be noted that OML methods achieve approximate results. If the data is completely available and can be processed in its entirety, BML methods provide better results. Even if OML methods theoretically need fewer resources (memory, runtime) than BML methods, this advantage is not easy to achieve in practice, because, e.g., the trees can become arbitrarily large. Thus, a clever choice of hyperparameters is crucial for the use of OML methods. Moreover, in many cases, with online data, data quality is of even greater importance for the algorithms than with offline data. Appropriate data preprocessing is essential. For a true online deployment, data pre-processing must, of course, happen online, which brings additional challenges. Currently, there are also only a few freely available OML software packages (e.g., the River package is very promising, but as of December 2023 it is still only in version 0.21.0). Also, continuous evaluation of the model's performance and monitoring system behavior are crucial in an online learning setup.

Considerations for the Implementation Process

- Clarification: Are OML procedures really necessary?
- How do BML procedures perform?
- How are the OML hyperparameters set?
- Can testing be initially conducted on small data sets?
- What does an experimental design look like?
- How should quality be monitored during ongoing operations?

7.3.2 Expenditure for Implementation and Maintenance

Considering the diverse spectrum of use cases, each characterized by its unique challenges and parameters, the absence of precise data in the academic literature comparing the resource and time demands associated with model updating in the context of OML and BML is not surprising.

Typically, the implementation of OML methods demands a relatively high level of effort. This is because, in contrast to their BML counterparts, OML models seldom offer an "out-of-the-box" solution and often require customization to suit specific problems. In addition to these implementation costs, it is crucial to factor in the ongoing expenses for maintenance and regular oversight of the models when performing a profitability analysis of OML methods. Although certain OML models exhibit the capacity to respond to concept drift to some degree, the quality of predictions should be routinely scrutinized, and the models should be adjusted if the deviations become excessively large. Broadly, the necessity for fundamental adjustments to models in practical applications is frequently underestimated. For example, it is not rare to observe evolution in the data foundation over time, with variables undergoing addition, modification, or complete removal.

7.3.3 Application and Diffusion in Practice

The implementation of OML in practical use cases ultimately hinges on whether the benefits of its use outweigh the potential downsides, and if it generates sufficient overall added value.

In the two use cases examined in this chapter, the situation was quite different:

- In the domain of official statistics, it emerged that there are still several legacy challenges to overcome, which is why there were hardly any applications found

within the core area of official statistics. However, in closely related fields, some applications do exist, as they do not encounter these challenges to the same extent.
- The situation is entirely different in the hot rolling use case, where, due to specific requirements such as, e.g., adaptation to drift, a shift towards OML solutions appears to be inevitable.

This is also somehow in accordance with our overall findings, which suggest that the implementation of OML is only sensible where its advantages can be fully leveraged. OML is currently predominantly used in areas, where this is the case. Consequently, these are typically applications where large data streams are generated, and swift evaluations of these data are required. Typical examples that can be found online of such use cases include sensor data, real-time applications, warning systems, and systems within the realm of the Internet of Things (IoT). Specific examples would be, e.g., production lines in industry (Soto et al., 2019), real-time occupancy detection in smart buildings (Elkhoukhi et al., 2022), real-time cyber-power event and intrusion classification (Adhikari et al., 2018), smart traffic management (Nallaperuma et al., 2019), or sentiment analysis on Twitter (Rezaei & Jalali, 2017).

However, caution should be exercised when interpreting these reports from the academic world. The extent to which these methods are actually being utilized productively in regular companies and institutions is not necessarily apparent from these rather academic reports. Overall, despite the documented use cases in the aforementioned fields, OML does not seem to have been extensively adopted or achieved significant breakthroughs in other areas, as can also be inferred from the fact that the OML software landscape is clearly still underdeveloped.

7.3.4 Overall Conclusions

The practical implementation of OML involves numerous challenges and entails a multitude of considerations. One such crucial contemplation is discerning whether its application is indeed beneficial or advantageous within a given context. OML demonstrates significant potential in scenarios favoring its unique characteristics–specifically, those featuring massive data streams or necessitating real-time evaluations.

As demonstrated in Sect. 7.2, in scenarios that are perfectly suited for it, the application of OML seems almost intuitive. This is because OML offers effective solutions to requirements such as adapting to drift and managing massive volumes of incoming data.

Although some applications in official statistics seem well-suited for OML, our exploration revealed that other factors, not directly related to performance, significantly influence its implementation. Integrating OML into existing processes can indeed present a formidable challenge. Additionally, quality aspects such as reduced accuracy or complicated reproducibility and interpretability pose significant hurdles. The current state-of-the-art in OML, which is still evolving, often cannot yet provide comprehensive solutions to these multifaceted problems, thereby raising pertinent questions about its readiness for widespread adoption.

Interestingly, the decision to implement OML instead of BML involves considering more nuanced factors. These factors include the availability of hardware resources, existing expertise, and the compatibility of OML software products with the current IT infrastructure. For instance, if an organization has ample computational resources, it might be more feasible to attempt running everything as BML, even if it is resource-intensive.

It is also worth noting that the trade-off between classification performance and speed is not a simple binary choice between BML and OML. Even within batch processing, various degrees of subtlety exist based on factors such as the training window size, the number of considered training data, the number of considered variables and observations, and the type of model used. There is always an option to adjust some of these parameters to achieve faster batch processing, albeit at the cost of classification performance. For instance, one could opt for a faster BML model instead of a slower one, say, Support Vector Machine (SVM).

This is also a trend frequently observed in OML application publications available online, where comparisons among various methods and models are often presented. Furthermore, it is not unlikely that in many scenarios, even though they are well-suited for OML, the decision ultimately leans towards BML.

In general, based on the literature, it can be asserted that OML applications are predominantly found in domains where the unique strengths of OML perfectly align with the requirements of the problem. In these instances, the implementation of OML emerges as a beneficial choice, underscoring its significant potential within a carefully chosen set of application scenarios.

The ongoing proliferation and adoption of OML are likely to be influenced by the continuing trend of increasing data volumes. Larger and more frequent data streams can strain the capacity of existing infrastructure. Nevertheless, this trend is mitigated by the simultaneous advent of decreasing computational costs, expanded storage availability, and continuous optimization of BML methodologies.

Therefore, the usage of OML, at this stage, can be seen as a carefully considered choice rather than an automatic selection, one that involves a detailed assessment of both the problem at hand and the available resources and expertise. Deploying OML should ideally be a strategic decision, enacted only when truly necessary. As OML continues to evolve and mature, its practical applications are likely to expand, but its role in practice will remain tightly bound to the unique needs and constraints of each individual use case.

References

Adhikari, U., Morris, T. H., & Pan, S. (2018). Applying Hoeffding adaptive trees for real-time cyber-power event and intrusion classification. *IEEE Transactions on Smart Grid, 9*(5), 4049–4060. https://doi.org/10.1109/TSG.2017.2647778

Alvarez, F., Roman-Rangel, E., & Montiel, L. V. (2022). Incremental learning for property price estimation using location-based services and open data. *Engineering Applications of Artificial Intelligence, 107*, 104513.

Andreini, P., et al. (2023). Nowcasting German GDP: Foreign factors, financial markets, and model averaging. *International Journal of Forecasting, 39*(1), 298–313. ISSN: 0169-2070. https://doi.org/10.1016/jijforecast.2021.11.009

Aparicio, D., & Bertolotto, M. I. (2020). Forecasting inflation with online prices. *International Journal of Forecasting, 36*(2), 232–247.

Beck, M., Dumpert, F., & Feuerhake, J. (2018a). *Machine learning in official statistics*. arXiv:1812.10422

Beck, M., Dumpert, F., & Feuerhake, J. (2018). *Proof of concept machine learning - Abschlussbericht*. Technical Report, Statistisches Bundesamt (Destatis), Wiesbaden.

Blumöhr, T., Teichmann, C., & Noack, A. (2017). Standardisierung der Prozesse: 14 Jahre AG SteP. *WISTA - Wirtschaft und Statistik, 5*, 58–75. https://www.destatis.de/DE/Methoden/WISTA-Wirtschaft-und-Statistik/2017/05/standardisierung-prozesse-052017.html

Chatterjee, S., & Gupta, S. (2021). Incremental real-time learning framework for sentiment classification: Indian general election 2019, a case study. In *2021 IEEE 6th International Conference on Big Data Analytics, ICBDA 2021* (pp. 198–203). https://doi.org/10.1109/ICBDA51983.2021.9402992

Dumpert, F., & Beck, M. (2017). Einsatz von Machine-Learning-Verfahren in amtlichen Unternehmensstatistiken. *AStA Wirtschafts- und Sozialstatistisches Archiv, 11*(2), 83–106.

Elkhoukhi, H., et al. (2022). Using stream data processing for real-time occupancy detection in smart buildings. *Sensors, 22*(6). ISSN: 1424-8220. https://doi.org/10.3390/s22062371. https://www.mdpi.com/1424-8220/22/6/2371

Generic Statistical Business Process Model - GSBPM (2019). https://www.statswiki.unece.org/display/GSBPM/GSBPM+v5.1

Ginzburg, V., & Ballas, R. (2000). Flat rolling fundamentals. In *Manufacturing engineering and materials processing*. Taylor & Francis.

Hensel, A., & Spittel, T. (1978). *Kraft- und Arbeitsbedarf bildsamer Formgebungsverfahren*. Verlag Grundstoffindustrie.

Hinkfoth, R. (2003). *Massivumformung*. Wissenschaftsverlag.

Jung, C. (2019). Data-driven optimization of hot rolling processess. https://katalog.ub.tu-dortmund.de/id/ir01388a:ubd.lobid:990365227020206441

Kimura, T., et al. (2022). Fast mining and forecasting of co-evolving epidemiological data streams. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD'22, New York, NY, USA* (pp. 3157–3167). Association for Computing Machinery.

Nallaperuma, D., et al. (2019). Online incremental machine learning platform for big data-driven smart traffic management. *IEEE Transactions on Intelligent Transportation Systems, 20*(12), 4679–4690. https://doi.org/10.1109/TITS.2019.2924883

Qualitätshandbuch der Statistischen Ämter des Bundes und der Länder. (2021). 1(21). https://www.destatis.de/DE/Methoden/Qualitaet/qualitaetshandbuch.pdf

Quality Assurance Framework of the European Statistical System. (2019). 2.0. https://ec.europa.eu/eurostat/documents/64157/4392716/ESSQAF-V2.0-final.pdf

Radermacher, W. J. (2018). Official statistics in the era of big data opportunities and threats. *International Journal of Data Science and Analytics, 6*(3), 225–231. https://doi.org/10.1007/s41060-018-0124-z

Rezaei, Z., & Jalali, M. (2017). Sentiment analysis on Twitter using McDiarmid tree algorithm. In *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)* (pp. 33–36). https://doi.org/10.1109/ICCKE.2017.8167924

Saidani, Y., et al. (2023). Qualitätsdimensionen Maschinellen Lernens in der Amtlichen Statistik. *AStA Wirtschafts- und Sozialstatistisches Archiv, 17*, 253–303. https://doi.org/10.1007/s11943-023-00329-7

Schweinfest, S., & Jansen, R. (2021) Data science and official statistics: Toward a new data culture. *Harvard Data Science Review, 3*(4). https://doi.org/10.1162/99608f92.c1237762. https://hdsr.mitpress.mit.edu/pub/1g514ljw/release/4

Senftleben, C., & Strohsal, T. (2019). Nowcasting: Ein Echtzeit- Indikator für die Konjunkturanalyse. *Schlaglichter der Wirtschaftspolitik, 7*, 12–14. https://www.bmwk.de/Redaktion/DE/Schlaglichter-der-Wirtschaftspolitik/2019/07/kapitel-1-3-nowcasting-einechtzeit-indikator-fuer-die-konjunkturanalyse.html

Soto, J. A. C., Tavakolizadeh, F., & Gyulai, D. (2019). An online machine learning framework for early detection of product failures in an Industry 4.0 context. *International Journal of Computer Integrated Manufacturing, 32*(4–5), 452–465. https://doi.org/10.1080/0951192X.2019.1571238

Steinberg, P., Börnsen, N., & Neumann, D. (2021). Digitale Ordnungspolitik -Wirtschaftspolitik daten- und evidenzbasiert weiterentwickeln. *Wirtschaftsdienst, 101*(9), 706–712.

Suárez-Cetrulo, A. L., Kumar, A., & Miralles-Pechun, L. (2021). Modelling the COVID-19 virus evolution with incremental machine learning. arXiv:2104.09325

Weber, K. (1973). *Grundlagen des Bandwalzens*. VEB Deutscher Verlag fuer Grundstoffindustrie.

Yung, W., et al. (2022). A quality framework for statistical algorithms. *Statistical Journal of the IAOS, 38*(1). 291–308 (IOS Press). https://doi.org/10.3233/SJI-210875. https://content.iospress.com/articles/statistical-journal-of-the-iaos/sji210875

Chapter 8
Open-Source Software for Online Machine Learning

Thomas Bartz-Beielstein

Abstract In contrast to Batch Machine Learning (BML), there are only a few open-source software packages for Online Machine Learning (OML). This chapter describes the availability of open-source software packages (especially in R/Python) that provide OML methods and algorithms to solve tasks such as regression, classification, clustering, or outlier detection. Section 8.1 gives an overview of the software, followed by a description of the corresponding packages. Then, Sect. 8.2 provides a comparative overview of the scope of the individual software packages. The chapter concludes with a comparison of the most important programming languages in the field of Machine Learning (ML) (Sect. 8.3).

8.1 Overview and Description of Software Packages for Online Machine Learning

Table 8.1 provides an overview of the software packages. The selection of packages consists of two R, one Python and one Java package. The Java package is included for two reasons: first, it is one of the oldest and most popular open-source software packages for data stream mining, and second, both R packages depend on it. The following section briefly describes the individual packages.

8.1.1 MOA

MOA (Bifet & Holmes, 2010) is one of the oldest and most popular open-source software packages for data stream mining. It contains a collection of streaming data algorithms for supervised (classification, regression, etc.) and unsupervised learning (clustering, etc.) in Java. MOA is developed by the University of Waikato. It provides a graphical user interface, but can also be used via the command line in the terminal

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

Table 8.1 OML open-source software

Software	Language	Last update	Remarks	References
MOA	Java	2021	Established, optional GUI	Bifet et al. (2018)
RMOA	R	2022	Focus on classification	Wijffels (2014)
Stream	R	2022	Focus on clustering	Hahsler et al. (2017b)
River	Python	2022	Current, active development, deep learning	Montiel et al. (2021)

or the Java programming interface (API). MOA received several releases per year until 2021. The last release is from April 2023.

8.1.2 RMOA

RMOA (Wijffels, 2014) provides an interface between the programming language R and the software package MOA. The package is developed by a single person at irregular intervals. The last update was released in July 2022. However, it must be taken into account that RMOA is an interface to the MOA version `2014.04`. This is an outdated MOA version from 2014, which can no longer be found in the release history of the MOA GitHub page. RMOA focuses mainly on classification models, but also includes some regression models.

8.1.3 Stream

The R package stream (Hahsler et al., 2017a) provides, in addition to the data stream mining algorithms, the possibility of simulating and modeling data streams. The focus is on clustering algorithms. Like the package RMOA, it is maintained by a main developer and has had at least one new release per year in the last 3 years. As an extension package, streamMOA exists. This extension, like RMOA, provides an interface to MOA to make the clustering algorithms from MOA available in R. The MOA release 18.06.0 (June 2018) is used for this purpose.

8.1.4 River

Recently, there has been a rapid development of OML algorithms for Python. In particular, the package River is worth mentioning, which will be discussed in more detail below.

River is a relatively new Python package for OML. It is the result of the fusion of the packages Creme and Scikit-Multiflow. The scikit-multiflow framework builds on other well-known open-source frameworks such as scikit-learn, MOA, and MEKA. Therefore, River is already a well-developed package, even though it is very new under this name. Several main developers are actively working on the package. In 2022 alone, there were six releases. River supports various ML tasks, including regression, classification, and clustering. River can also be used for ad hoc tasks, such as computing online metrics and detecting concept drift. In a Python environment, River is the most user-friendly OML package, as it works well with Python dictionaries. Therefore, it can be easily used in the context of web applications where JSON data is abundant.

Pipelines in the River package use a special approach adapted to OML, which is fundamentally different from the approach used for BML algorithms. This enables efficient and flexible data preprocessing.

Tip
The following description is valid for River version <= 0.18.0. The new behaviour is described by the River developers as follows:

Starting with version 0.19.0, calling "`learn_one` in a pipeline will update each part of the pipeline in turn. Before the unsupervised parts of the pipeline were updated during `predict_one`. This is more intuitive for new users. The old behavior, which yields better results, can be restored by calling `learn_one` with the new `compose.learn_during_predict` context manager." Details can be found on the River webpage: https://riverml.xyz/0.21.0/releases/0.19.0/ and in the Jupyter Notebooks from this book that are provided in the GitHub repository https://github.com/sn-code-inside/online-machine-learning.

The `learn_one` method only updates the model (the supervised part of the OML pipeline), but not the estimators (the unsupervised part of the pipeline). The unsupervised part, the so-called transformers, is updated when `predict_one` is called. In OML, the unsupervised parts of the processing pipeline can be updated when an observation arrives. We do not have to wait for the ground truth (the Y value) to arrive to update the unsupervised estimators, which depend only on the X values. In other words, `learn_one` updates the supervised parts, while `predict_one` updates the statistics of the unsupervised parts of the pipeline. When `transform_one` is called for a pipeline whose last step is not a transformer, the output of the last transformer (which is thus the penultimate step) is returned. This allows the data preprocessing to be observed without training the model. Table 8.2 summarizes these steps.

Table 8.2 Supervised and unsupervised steps and the related methods in River pipelines explained by the example of the classes `StandardScaler` and `LinearRegression` (valid for River version <= 0.18.0)

Class, method	Step type	Remark
`StandardScaler`		
`.preprocessing`	Unsupervised	Transformer: processes the features (X), not the target (Y). During a call to `predict_one()`, the statistics (e.g., the mean) are updated for each numerical feature
`LinearRegression`		
`.linear_model`	Supervised	Updates the model with the information of the features (X) and the target (Y) when `learn_one()` is called

Notebook: Introduction to River
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning/ provides an introduction to River. It shows the use of River using an example and introduces the most important methods.

8.2 Scope of the Software Packages

Table 8.3 provides an overview of the scope of the individual software packages. Since the documentation of the packages is not always kept up to date, it is possible that individual packages have included more up-to-date methods over time. In addition, we have limited the categories of the listed methods to the application areas that we consider to be the most relevant for OML. Therefore, the table only contains methods from the areas of classification, clustering, and regression. Table 8.3 is certainly not complete, but it can provide a good orientation.

MOA provides methods for each category, while RMOA makes methods for classification and regression accessible to R in particular. Stream is a rather small package that brings some own implementations of clustering methods. The extension of stream, streamMOA, also provides the clustering methods implemented in MOA for R.[1] River is the most comprehensive and contains the most important methods from each area. It also contains many additional and more up-to-date methods that are not available in the R packages.

8.3 Programming Languages: A Brief Comparison

When comparing Python, R, and Julia, the leading languages for data science and ML, it becomes apparent that the R community consists of experienced statisticians. On the other hand, Python has caught up with R in statistics and scientific computing with libraries such as NumPy, SciPy, and Pandas and has partly overtaken R in terms of user-friendliness. Python stands out in terms of libraries for machine learning. The following libraries are written entirely or primarily in Python:

- NumPy is a library for scientific computing in Python (Harris et al., 2020). It provides an efficient implementation of multidimensional arrays and many mathematical functions.
- Scikit-learn is written in Python and Cython[2] (Pedregosa et al., 2011). It provides implementations of a very large number of algorithms for training and evaluating models for machine learning.
- Statsmodels provides statistical tests and models such as the generalized linear model (GLM), ARMA, and many more (Seabold et al., 2010).
- Keras is used to interact with TensorFlow and other deep learning libraries (Chollet et al., 2015; Abadi et al., 2016).

The most popular frameworks for ML are also mainly written in Python or provide interfaces for Python. In addition, Python is ideal as a general-purpose programming language.

Tip
- R is ideal for statistical special questions and graphics (shiny).
- Python is the “Swiss Army Knife” in the field of data science.
- A combination of the two languages makes sense.
- The influence of Julia is only marginal compared to the two market leaders.

[1] However, this is not taken into account in the table.

[2] A Python dialect similar to the programming language C.

Table 8.3 Scope of the software packages

Category	Method	MOA	RMOA	Stream	River
Classification					
Trees	AdaHoeffdingOptionTree	✓	✓		
	ASHoeffdingTree	✓	✓		
	DecisionStump	✓	✓		
	HoeffdingAdaptiveTree	✓	✓		✓
	HoeffdingOptionTree	✓	✓		
	HoeffdingTree	✓	✓		✓
	LimAttHoeffdingTree	✓	✓		
	RandomHoeffdingTree	✓	✓		
	ExtremelyFastDecsionTree				✓
	LabelCombinationHoeffdingTree				✓
Regression	LogisticRegression				✓
Bayesian	Naive Bayes	✓	✓		✓
	Naive-Bayes Multinomial	✓	✓		✓
	Bernoulli				✓
	Complement				✓
SVM	ALMAClassifier				✓
	PAClassifier				✓
Active learning	ActiveClassifier	✓	✓		
Bagging	LeveragingBag	✓	✓		✓
	OzaBag	✓	✓		✓
	OzaBagAdwin	✓	✓		✓
	OzaBagASHT	✓	✓		
Boosting	OCBoost	✓	✓		
	OzaBoost	✓	✓		✓
	OzaBoostAdwin	✓	✓		
Stacking	LimAttClassifier	✓	✓		✓
Other	AccuracyUpdatedEnsemble	✓	✓		
	AccuracyWeightedEnsemble	✓	✓		
	ADACC	✓	✓		
	DACC	✓	✓		
	OnlineAccuracyUpdatedEnsemble	✓	✓		
	TemporallyAugmentedClassifier	✓	✓		
	WeightedMajorityAlgorithm	✓	✓		
	AdaptiveRandomForest				✓
	StreamingRandomPatches	✓			✓
	VotingClassifier				✓
	FFMClassifier				✓
	FMClassifier				✓
	FwFMClassifier				✓
	HOFMClassifier				✓
	KNNClassifier				✓
	StochasticGradientDescent				✓

(continued)

Table 8.1 (continued)

Category	Method	MOA	RMOA	Stream	River
Clustering					
	BICO			✓	
	BIRCH			✓	
	DBSTREAM			✓	✓
	DStream	✓		✓	
	evoStream			✓	
	CluStream	✓			✓
	StreamKM++	✓			✓
	ClusTree	✓			✓
	DenStream	✓			✓
	CobWeb	✓			
Regression					
	TargetMean	✓	✓		
	Perceptron	✓	✓		✓
	FIMTDD	✓	✓		
	ORTO	✓	✓		
	LinearRegression				✓
	AdaptiveRandomForest	✓			✓
	BaggingRegressor				✓
	EWARegressor				✓
	StreamingRandomPatches				✓
	FFMRegressor				✓
	FMRegressor				✓
	FwFMRegressor				✓
	HOFMRegressor				✓
	BayesianLinearRegression				✓
	PARegressor				✓
	SoftmaxRegression				✓
	KNNRegressor				✓
	MultiLayerPerceptron				✓
	HoeffdingAdaptiveTree				✓
	HoeffdingTree				✓
	iSOUPTree				✓

References

Abadi, M., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467

Bifet, A., Gavalda, R., et al. (2018). *Machine learning for data streams with practical examples in MOA*. MIT Press.

Bifet, A., Holmes, G., et al. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research, 11*, 1601–1604.

Chollet, F., et al. (2015). Keras. https://keras.io

Hahsler, M., Bolaños, M., & Forrest, J. (2017a). Introduction to stream: An extensible framework for data stream clustering research with R. *Journal of Statistical Software, 76*(14), 1–50.
Hahsler, M., Bolaños, M., & Forrest, J. (2017b). *Stream: Infrastructure for data stream mining*.
Harris, C. R., et al. (2020). Array programming with NumPy. *Nature, 585*(7825), 357–362.
Montiel, J., et al. (2021). River: Machine learning for streaming data in Python. *Journal of Machine Learning Research, 22*(1), 4945–4952. ISSN: 1532-4435.
Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.
Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. In *9th Python in Science Conference*.
Wijffels, J. (2014). RMOA: Connect R with MOA to perform streaming classifications.

Chapter 9
An Experimental Comparison of Batch and Online Machine Learning Algorithms

Thomas Bartz-Beielstein and Lukas Hans

Abstract This chapter presents the results of the experimental analyses. The first study (Sect. 9.1) examines the use of Batch Machine Learning (BML) and Online Machine Learning (OML) models for predicting the demand for bicycles at a bike-sharing station. The second study (Sect. 9.2) investigates the use of BML and OML models for prediction when very large data sets are available and drift is present. The synthetic Friedman-drift data set (see Definition 1.8) is used for this purpose. All data sets were standardized using the StandardScaler method so that the models were trained on data with mean zero and standard deviation one. In Sect. 9.3, we conducted a comprehensive investigation to evaluate the efficacy of scaling techniques in the context of drifting events. Our primary hypothesis centered on the potential benefits of scaling in handling dynamic data streams. Through rigorous experimentation and analysis, we compared various scaling methods to determine if one specific approach outperforms others in adapting to evolving data distributions.

9.1 Study: Bike Sharing

Titled “Time-related feature engineering”, scikit-learn[1] presents an example analyzing strategies for using time-related features for a bike-sharing demand regression task. In this study, bike-sharing demand data is loaded from the OpenML repository. On the OpenMl site,[2] the data are described as follows:

[1]https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html.
[2] https://www.openml.org/search?type=data&sort=runs&id=42713&status=active.

T. Bartz-Beielstein (✉) · L. Hans
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

L. Hans
e-mail: lukas.hans@th-koeln.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_9

Table 9.1 Attributes of the Bike-Sharing data set

Attribute	Description
`Season`	Season (1: Spring, 2: Summer, 3: Fall, 4: Winter)
`Yr`	Year (0: 2011, 1: 2012)
`Mnth`	Month (1–12)
`Hr`	Hour (0–23)
`Holiday`	Holiday[1]
`Weekday`	Weekday
`Workingday`	Working day (neither weekend nor holiday)
`Weathersit`[2]	Weather situation (1: Clear, few clouds, partly cloudy 2: Mist + cloudy, mist + broken clouds, Mist + few clouds, mist 3: Light snow, light rain + thunderstorm + Scattered clouds, light rain + scattered clouds 4: Heavy rain + hail + Thunderstorm + fog, snow + fog)
`Temp`	Temperature in Celsius
`Atemp`	Feeling temperature in Celsius
`Hum`	Humidity
`Windspeed`	Wind speed
`Casual`	Number of casual users
`Registered`	Number of registered users
`Count`	Target variable: Number of total rental bikes, including casual users and registered users

[1]Taken from http://dchr.dc.gov/page/holiday-schedule.
[2]Since there are only three “heavy_rain” events, we simplify the presentation by combining them with the entries in the “rain” category.

> “Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. Through these systems, user is able to easily rent a bike from a particular position and return back at another position. Currently, there are about over 500 bike-sharing programs around the world which is composed of over 500 thousands bicycles. Today, there exists great interest in these systems due to their important role in traffic, environmental and health issues. [...]
>
> Bike-sharing rental process is highly correlated to the environmental and seasonal settings. For instance, weather conditions, precipitation, day of week, season, hour of the day, etc. can affect the rental behaviors”.

The data were aggregated by Fanaee-T et al. (2014) and supplemented with the appropriate weather and seasonal information.[3] The total data set includes 17,379 observations. The size count is selected as the target variable. The data are described in Table 9.1.

[3] http://www.freemeteo.com.

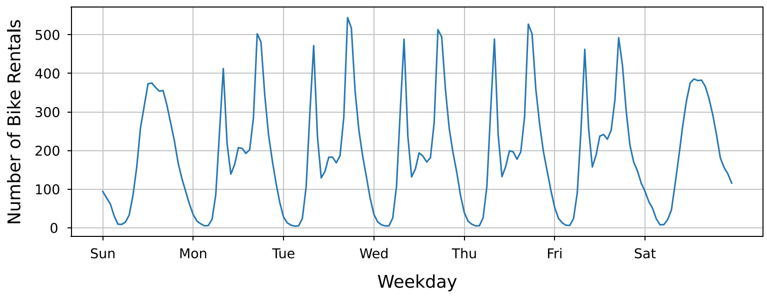
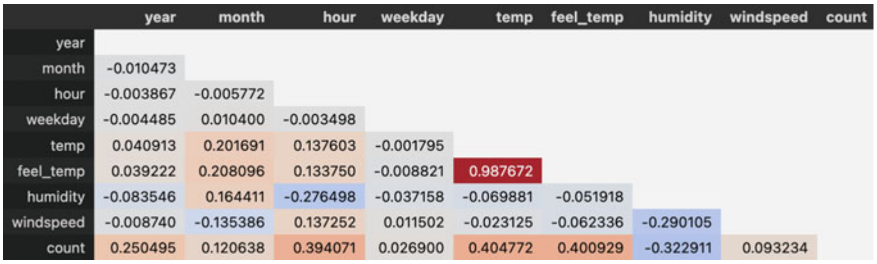


Fig. 9.1 Average number of bicycle rentals in the course of a week



	year	month	hour	weekday	temp	feel_temp	humidity	windspeed	count
year									
month	-0.010473								
hour	-0.003867	-0.005772							
weekday	-0.004485	0.010400	-0.003498						
temp	0.040913	0.201691	0.137603	-0.001795					
feel_temp	0.039222	0.208096	0.133750	-0.008821	0.987672				
humidity	-0.083546	0.164411	-0.276498	-0.037158	-0.069881	-0.051918			
windspeed	-0.008740	-0.135386	0.137252	0.011502	-0.023125	-0.062336	-0.290105		
count	0.250495	0.120638	0.394071	0.026900	0.404772	0.400929	-0.322911	0.093234	

Fig. 9.2 Bike-share correlations. There are no unusual patterns to be identified. Highly positively correlated features are colored in *red*, highly negatively correlated features are colored in *blue*. Obviously, the feeling temperature is highly correlated with the temperature. `Count` is the target variable

Figure 9.1 depicts the average demand during a week. We can clearly distinguish between morning and evening commuting on work days and recreational use of bicycles on weekends, when peak demand occurs around noon.

Figure 9.2 shows the correlations of the features with the target variable. The target of the prediction problem is the absolute number of hourly bicycle rentals for the subsequent 7 days, which is referred to as `count`. We scale the target variable to predict relative demand so that the mean absolute error can be more easily interpreted as a fraction of the maximum demand.

We split the data into 60% for training and 40% for testing. Thus, 10,427 training and 6,952 testing data sets are available. The data retain their original order.

Table 9.2 Model types, implementations, and references

Model	Implementation	Reference
Linear regression	Sklearn: `RidgeCV`	Pedregosa et al. (2011)
Gradient boosting	Sklearn: `HistGradientBoostingRegressor`	Pedregosa et al. (2011)
Linear regression	River: `LinearRegression`	Montiel et al. (2021)
Hoeffding tree	River: `HoeffdingTreeRegressor`	Montiel et al. (2021)
Hoeffding adaptive tree	River: `HoeffdingAdaptiveTreeRegressor`	Bifet et al. (2009), Montiel et al. (2021)

Note: MAE and MSE
The fit of the models minimizes the mean squared error (MSE) to estimate the conditional mean. The mean absolute error (MAE) would fit an estimator identical to the estimator of the conditional median. When we report results in the discussion, we focus instead on the MAE, which is more intuitive than the MSE. It should be noted, however, that in this study the best models for the one metric are also the best for the other.

9.1.1 Overview: Models

The BML and OML models shown in Table 9.2 were used in this study:

Regarding the selection of the models it was taken into account that for each Machine Learning (ML)-algorithm category (BML or OML) one simple and one more complex model is used. For the OML models, it was also considered that the influence of a drift detection method (here: Adaptive Windowing (ADWIN)) can be analyzed.

9.1.2 Linear Regression

As the simplest model, we use linear regression, which is generally accepted and well understood as a standard procedure in classical statistics.

Tip
For the experiments performed with River the software package Sequential Parameter Optimization Toolbox for River (spotRiver) was developed. The experiments can be found as Jupyter Notebooks on GitHub https://github.com/sn-code-inside/online-machine-learning/.

9.1.2.1 BML Linear Regression

For consistency, we scale the numerical features to the interval from zero to one using `sklearn.preprocessing`. However, in this case this does not have much impact on the results because they are already on comparable scales. We use scikit-learn: Machine Learning in Python (sklearn)'s Ridge regression with built-in Cross Validation (RidgeCV). Here (and in all other BML examples) we adopt the methods described on sklearn.[4]

```
categorical_columns = [
    "weather",
    "season",
    "holiday",
    "workingday",
]
categories = [
    ["clear", "misty", "rain"],
    ["spring", "summer", "fall", "winter"],
    ["False", "True"],
    ["False", "True"],
]
one_hot_encoder = OneHotEncoder(handle_unknown="ignore",
    sparse_output=False)
alphas = np.logspace(-6, 6, 25)
linear_pipeline = make_pipeline(
    ColumnTransformer(
        transformers=[
            ("categorical",
             one_hot_encoder,
             categorical_columns),
        ],
        remainder=MinMaxScaler(),
    ),
    RidgeCV(alphas=alphas),
)
```

[4] https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html.

9.1.2.2 OML-Linear Regression

The modeling of the OML linear regression model follows the example “Bike-sharing forecasting” from the package River.[5] In this example, the demand for bicycles is forecast at five bike stations in the city of Toulouse. The data set contains 182,470 observations.

We choose the sklearn data set for two reasons: first, the examples used by sklearn are widely available, and second, a new method (in this case, River) should be able to compare itself against the established methods (sklearn), using the standards of the established methods as a benchmark.

```
oml_linear_model = compose.Select(
    'humidity',
    'temp',
    'feel_temp',
    'windspeed')
oml_linear_model += (
    feature_extraction.TargetAgg(
        by=['hour'],
        how=stats.Mean())
)
oml_linear_model |= preprocessing.StandardScaler()
oml_linear_model |= linear_model.LinearRegression()
```

Tip: Debugging River
We can use the `debug_one` method to see what happens to a particular instance. We train the model with the first 1,000 observations and then call `debug_one` with the next one. The `debug_one` method shows what happens to an input set of features step by step. An example is shown below:

```
Input
--------
feel_temp: 22.72500 (float)
holiday: False (str)
hour: 8 (int)
humidity: 0.82000 (float)
month: 4 (int)
season: summer (str)
temp: 18.86000 (float)
```

5 https://riverml.xyz/0.15.0/examples/bike-sharing-forecasting/.

```
weather: rain (str)
weekday: 4 (int)
windspeed: 12.99800 (float)
workingday: True (str)
year: 1 (int)


LinearRegression
-------------------
Name              Value      Weight     Contribution
     Intercept    1.00000    0.24210         0.24210
y_mean_by_hour    1.23551    0.17709         0.21880
     windspeed   -0.14590   -0.01982         0.00289
          temp   -0.16870    0.00672        -0.00113
     feel_temp   -0.11330    0.01813        -0.00205
      humidity    1.08461   -0.01596        -0.01731

Prediction: 0.44329
```

9.1.2.3 Comparison of BML- and OML-Linear Models on the Bike-Sharing Data set

The experimental design compares models from two different categories, BML and OML, for each of which reference implementations are used. Therefore, standards are compared, each using a different data preparation (feature generation).

For the evaluation, the `eval_bml_horizon`, `eval_bml_landmark`, and `eval_bml_window` methods described in Chap. 5 are used, which allow a comparison of BML and OML algorithms. The Mean Absolute Error (MAE) is chosen as the error measure. The OML linear model is evaluated using the function `eval_oml_horizon`, which was also described in Chap. 5. The values for the error, the time and the memory requirements are available. The results of the comparison experiments can be summarized and visualized with the functions `plot_bml_oml_horizon_metrics` and `plot_bml_oml_horizon_predictions`.

The results can be seen in Fig. 9.3. The experiments show that the OML-linear model (represented by the *red* lines in Fig. 9.3) performs better than the BML-linear model for all metrics (error, time, and memory requirements).

Figure 9.4 compares the values predicted by the models with the actual values. This provides a microscopic view on a section of the data, revealing the strengths and weaknesses of each model in detail.

9.1.3 Gradient Boosting

In addition to the comparison of BML and OML linear models, a gradient boosting model (Friedman & Jerome, 2001) is included in the comparison. Gradient boosting is currently one of the most successful models in BML, but requires more computational time and memory compared to linear models. The "histogram-based" Gradient Boosting Regression Tree (gbrt) algorithm by sklearn[6] is used, which is faster than sklearn's `GradientBoostingRegressor` for large data sets. The gbrt implementation is inspired by LightGBM (Ke et al., 2017). For data preprocessing, similar to linear models, the state-of-the-art implementation is used, which is described on sklearn as follows[7]:

> Gradient Boosting Regression with decision trees is often flexible enough to efficiently handle heterogeneous tabular data with a mix of categorical and numerical features as long as the number of samples is large enough.
>
> Here, we do minimal ordinal encoding for the categorical variables and then let the model know that it should treat those as categorical variables by using a dedicated tree splitting rule

```
ordinal_encoder = OrdinalEncoder(categories=categories)
gbrt_pipeline = make_pipeline(
    ColumnTransformer(
```

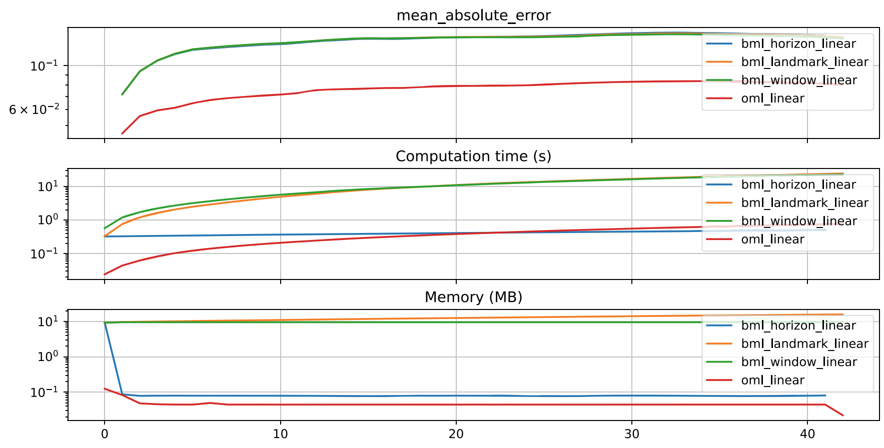


Fig. 9.3 Bike sharing. Comparison of error (MAE), time, and memory requirements of the four linear regression models. The BML-linear model is evaluated using the horizon, landmark, and window metrics. The OML-linear model is evaluated using the OML-horizon metric

[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoosting Regressor.html.

[7] https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html.

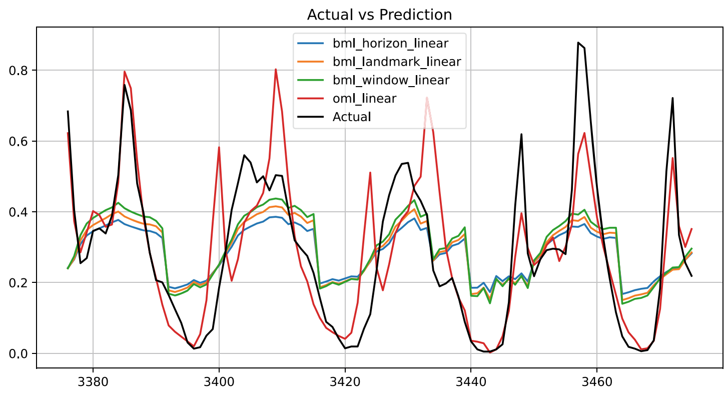


Fig. 9.4 Bike-sharing data. Actual versus predicted values. Comparison of the four linear regression models. The residuals are computed on an interval of size 100, which lies in the middle of the test data period

```
        transformers=[
            ("categorical",
             ordinal_encoder,
             categorical_columns),
        ],
        remainder="passthrough",
        verbose_feature_names_out=False,
    ),
    HistGradientBoostingRegressor(
        categorical_features=categorical_columns,
    ),
).set_output(transform="pandas")
```

9.1.3.1 Evaluation Metrics

The comparison of performance, time and memory requirements for gbrt is performed using the three evaluation functions `eval_bml_horizon`, `eval_bml_landmark`, and `eval_bml_window`, as was done for the BML-linear models in Sect. 9.1.2.

We compare the performance of gbrt with the performance of the best linear model, the OML-linear model from Sect. 9.1.2, which was evaluated using the `eval_oml_horizon` function.

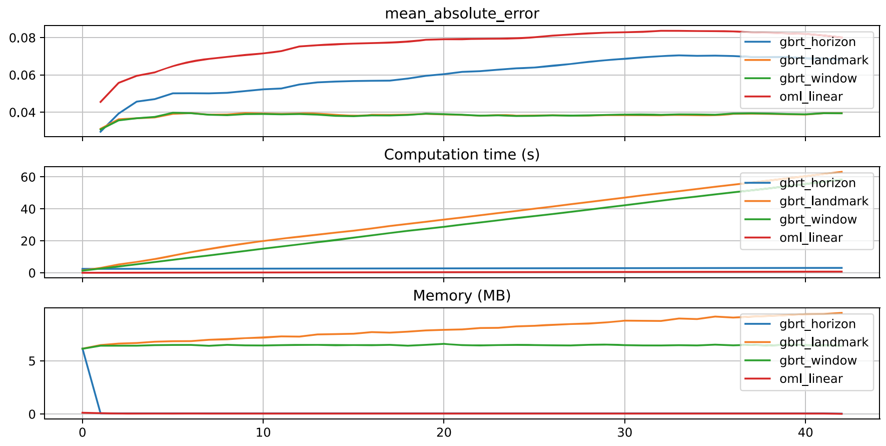


Fig. 9.5 Bike-Sharing Metrics. Three different selection methods (horizon, landmark, and window, cf. Sect. 5.1) of the BML-gradient boosting method gbrt compared with the OML-linear model from Sect. 9.1.2. Gradient boosting provides the best results in terms of the error measure, but requires more memory and time

9.1.3.2 Comparison of BML-Gradient Boosting with OML-Linear Model

Figure 9.5 compares the performance of the BML method gbrt with the performance of a simple OML-linear model. Differences can be seen within the three gbrt models: The landmark- and window-based gbrt models perform best. These reduce the MAE by about 50% compared to the OML-linear model. The landmark-based gbrt model achieves a worse MAE, which is still better than that of the OML-linear model. The time and memory requirements are higher for the landmark-based and window-based gbrt models than for the horizon-based gbrt and the OML-linear model.

These results indicate that there is a clear trade-off between modeling performance and resource requirements. The two best gbrt models require more resources than the OML-linear model, but produce better results. The worst gbrt model produces better results than the OML-linear regression model.

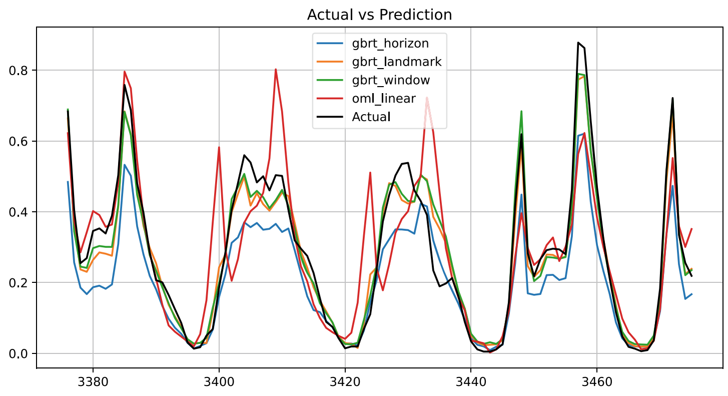


Fig. 9.6 Bike-sharing data. Actual versus predicted values on a section of the data. The black line ("actual") shows the ground truth. Shown are the predictions of the three gbrt variants (horizon, landmark, and window) of the gbrt model and, in addition, for comparison, the predictions of the OML-linear model

Warning

The memory required by the "best" BML gbrt models is not constant and is significantly higher than that required by the OML-linear models. By using a window-based model, constant time and memory requirements can also be achieved for the GBRT methods, which are comparable to the resource requirements of the OML model. However, this comes at the expense of modeling quality.

Figure 9.6 shows the corresponding residuals. The landmark-based gbrt model (shown in *orange*), which has the historical and the current (and thus the most data) available, achieves the best result, but also requires the most resources. The OML-linear model (shown in *red*) sometimes overshoots the mark. In contrast, the horizon-based gbrt model (shown in *blue*) behaves relatively conservatively. Overall, it can be observed that all models are able to fit a good model. The residuals are generally small, and the predictions are usually quite accurate.

9.1.4 Hoeffding Regression Trees

After comparing the simple OML methods with a complex BML method, we now investigate the performance of a complex OML method. For this purpose, we use the OML Hoeffding regression trees introduced in Sect. 2.1.3.1. For the Hoeffding trees, we selected Hoeffding Tree Regressor (HTR) and Hoeffding Adaptive Tree Regressor (HATR), i.e., two different OML methods (without and with the drift detection ADWIN).

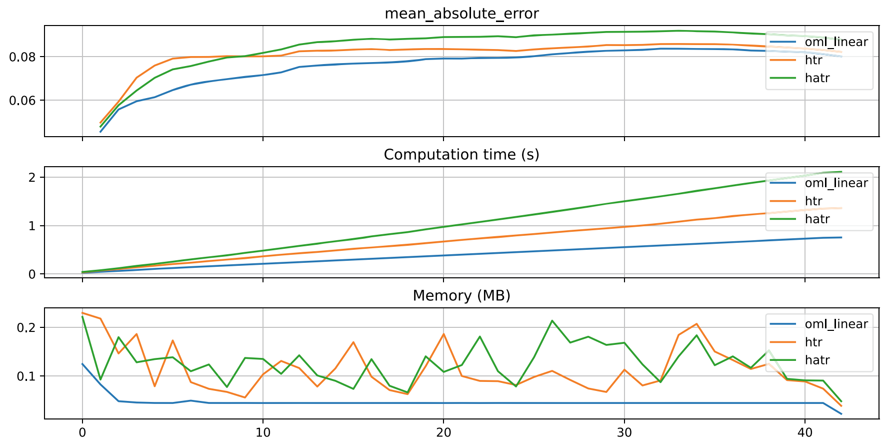


Fig. 9.7 Bike-sharing metrics. Evaluation of the OML-Hoeffding regression trees in comparison with the linear regression model

First, we compare the OML methods with each other, i.e., we compare the Hoeffding trees with the linear model from Sect. 9.1.2.2. Figure 9.7 compares the performance and resource requirements of the OML methods HTR, HATR, and the linear model.

It is noticeable that the linear model has the lowest resource requirement and provides the best results. At this point, no trade-off between modeling quality and resource requirements is observed. The HATR is the "loser" in this comparison: due to the use of ADWIN, the HATR is significantly slower than the linear model (and also compared to HTR) and delivers worse results. This result illustrates that it does not always make sense to choose an algorithm with extra features if the data situation does not require it. Already with the simple linear model a good result could be achieved.

A comparison of the actual and predicted values does not show any serious differences. The corresponding figure can be found in the notebook belonging to this chapter.

9.1.5 Final Comparison of the Bike-Sharing Experiments

Figure 9.8 compares the best methods from each of the ML categories considered in this chapter. The gradient boosting regressors with the window-based and the landmark-based approaches achieve the lowest MAE. The two OML methods (the linear model as well as the HTR) perform the worst. The gbrt with horizon is in the middle range.

These results indicate that retraining the gbrt takes a relatively large amount of resources. This can be done more efficiently by the OML methods. Looking at the

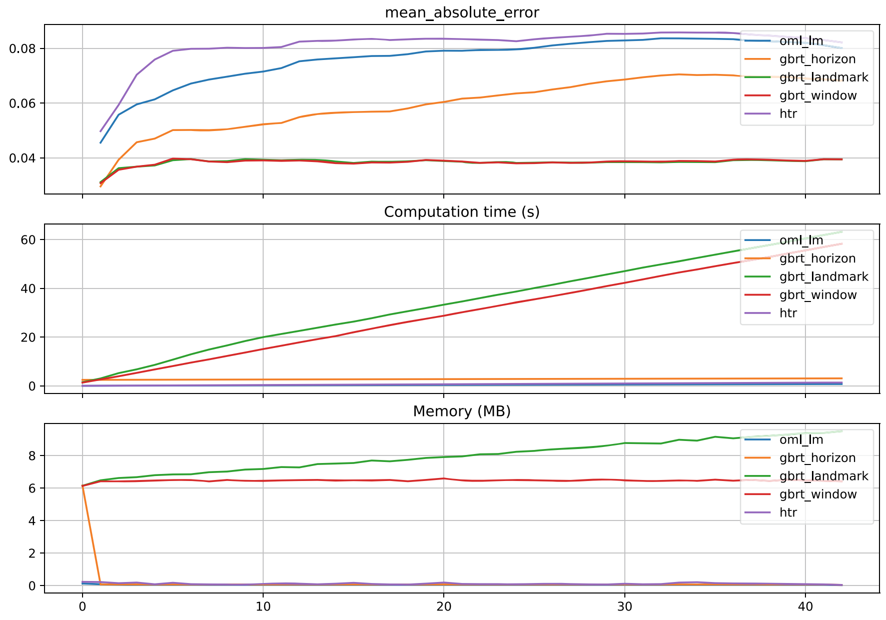


Fig. 9.8 Bike-sharing metrics. Final comparison (MAE, time and memory requirements) of the best BML and OML models. Compared are the three gbrt implementations (`gbrt_horizon`, `gbrt_landmark`, and `gbrt_window`) and two OML methods: the linear model `oml_lm` and the HTR `htr`

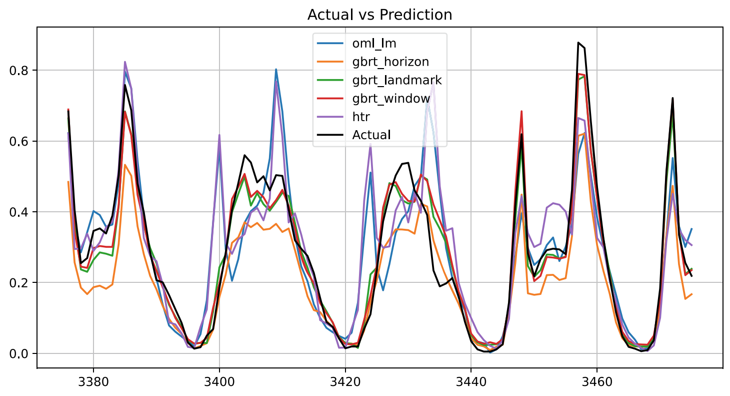


Fig. 9.9 Bike-sharing data. Actual versus predicted values. Final comparison of the best BML and OML models

residuals (Fig. 9.9), it is noticeable that all models provide a good fit. The gbrt horizon-based model behaves conservatively, while the two OML models react quickly to changes and sometimes overshoot the target.

9.1.6 Summary: Bike-Sharing Experiments

Table 9.3 shows a highly simplified summary of the results of the bike-share experiments conducted in this chapter. The least error occurs with the gbrt implementations. However, this good performance is not achieved without additional costs. OML methods do not achieve the performance of the best BML method, but they perform best in terms of resources.

It should be noted that this study used a relatively small data set with the bike-sharing data set, so the OML methods did not show their full strengths. In the following study, a very large data set is used to examine the resource requirements of the BML methods in particular.

Furthermore, it should be noted that no statistical tests or other advanced statistical methods were used. The results are therefore to be understood as qualitative statements only.

9.2 Study: Very Large Data Sets With Drift

Whether the very good performance of the BML methods observed in Sect. 9.1 is also achieved for very large online data sets with drift will be analyzed in the following study.

9.2.1 The Friedman-Drift Data Set

The synthetic Friedman data set with concept drift introduced in Definition 1.8 is used. As implementation of the drift a so-called Global Recurring Abrupt (GRA) drift is used: There are two points at which the concept (K, see Definition 1.7) changes. In this study, $n_{\text{total}} = 1{,}000{,}000$ data sets are used. The first concept change (drift)

Table 9.3 Bike-sharing experiments. Highly simplified summary of the results of the bike-share experiments

Model	Category	Error	Time	Memory
Linear regression	BML	–	–	–
Gradient boosting	BML	++	–	–
Linear regression	OML	o	–	–
Hoeffding tree	BML	–	+	+
Hoeffding adaptive tree	BML	–	–	–

Table 9.4 Models, implementations, and references

Model	Implementation	References
Linear regression	Sklearn: `RidgeCV`	Pedregosa et al. (2011)
Regression tree	Sklearn: `DecisonTreeRegressor`	Pedregosa et al. (2011)
Linear regression	River: `LinearRegression`	Montiel et al. (2021)
Hoeffding tree	River: `HoeffdingTreeRegressor`	Montiel et al. (2021)
Hoeffding adaptive tree	River: `HoeffdingAdaptive-`	
	`TreeRegressor`	Bifet et al. (2009), Montiel et al. (2021)

occurs after 250,000 samples. After 500,000 steps, the original concept is adopted again.

It is assumed that the data arrive every hour. As in the bike-sharing study (Sect. 9.1), a prediction horizon of $7 \times 24 = 168$ time steps is used.

9.2.2 *Algorithms*

Two BML and three OML algorithms are evaluated against each other, which are shown in Table 9.4. It was taken into account that one simple and one complex model from each category (BML or OML) is used. The gradient boosting procedure from the bike-sharing study is not used here because it leads to longer run times for very large data sets. Instead, a BML regression tree is used. A preprocessing pipeline is created for all algorithms so that data scaled with the method `StandardScale` is used by the algorithms. All methods are used with the default hyperparameter settings. The BML and OML models shown in Table 9.4 were used in this study.

9.2.2.1 The Evaluation Function

The BML models are trained on a training data set of size $n_{\text{train}} = 1{,}000$. The resulting model is then used to predict on approximately 6,000 (more precisely, $\lfloor(n_{\text{total}}/168\rfloor)$) mini-batches of size 168. The comparison of performance, time and memory requirements for gbrt is performed using the evaluation function `eval_bml_horizon` for the BML methods and the evaluation function `eval_oml_horizon` for the OML methods (see Sect. 5.1).

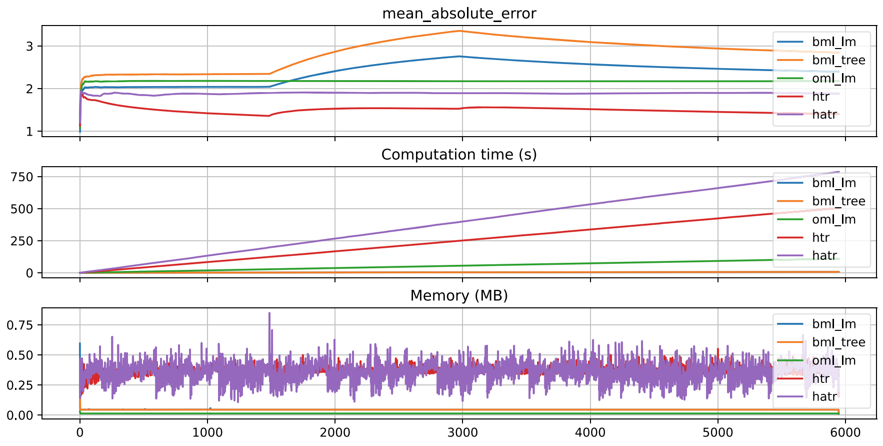


Fig. 9.10 Friedman-drift data. Metrics versus mini-batches. The one million observations were divided into batches of size 168 (7 days by 24 hours), resulting in 5,952 data points, which are plotted on the horizontal axis. The concept change after 250,000 and 500,000 instances can be clearly seen (in the figure approximately at the locations 1,500 and 3,000)

9.2.3 Results

Figure 9.10 compares the individual algorithms. Not surprisingly, the error of the BML algorithms increases when drift occurs. The prediction performance of the OML algorithms is only slightly (HTR) or not at all (HATR) affected by drift. The HTR algorithm has the lowest MAE overall.

Additional resources are needed to achieve this good result: Both Hoeffding tree methods show a continuously increasing time overhead and require the most memory. As expected, the BMLs require constant memory and time to compute the predictions. The linear regression models have the lowest memory requirements. Figure 9.11 shows the actual and the predicted values for which no anomalies can be detected.

9.3 Study: Drift Scaling in Online Machine Learning

In Sect. 3.5, as mentioned previously, we investigate the hypothesis that combining OML approaches with data scaling can effectively address drift events. To test this hypothesis, the first step involves acquiring data exhibiting drift. In this regard, we utilize the same data introduced in Figs. 7.5 and 7.7. Figure 7.5 demonstrates a label drift, while Fig. 7.7 represents the classical concept drift. For both data sets, baseline models were trained, as well as models using the data treated with scaling methods. The baseline models consist of three different models.

The first model is a static decision tree, the second model is a dynamic decision tree, and the third model is a Hoeffding Tree (HT) trained on unscaled data. The

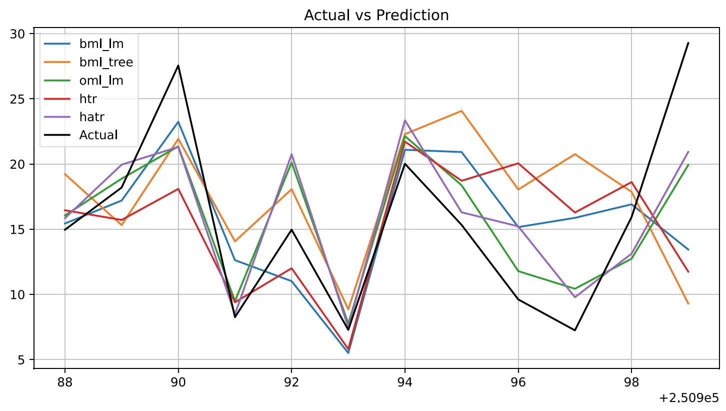


Fig. 9.11 Friedman-drift data. Actual versus predicted values. The OML-linear model adapts relatively quickly, but sometimes overshoots the target

static decision tree is trained solely on the first 2,000 data points and then used to make predictions for the remaining data points. In contrast, the dynamic decision tree differs in that it is retrained on every 2,000 data points based on the observed data up to that point. In theory, this approach should yield better results since it incorporates the data after the drift into the training process, allowing the model to recalibrate. For both models, the data is normalized using the sklearn standard scaler, which corresponds to a z-score normalization. For all OML approaches, both the unscaled and scaled Hoeffding trees, the first 2,000 data points are used for training only before making the first predictions. This is done for two reasons: first, to facilitate a direct comparison between OML approaches and BML approaches, and second, to account for the warm-up phase of many OML methods in which they perform relatively poorly. The success of the different methods is evaluated using MAE and presented in Fig. 9.12.

Figure 9.12 illustrates the performance of the different model approaches as the number of observed data points increases for the label drift problem. From Fig. 9.12, it is evident that the static BML approach is particularly affected by the label drift and experiences a significant decline in accuracy towards the end. Surprisingly, both the unscaled and mean-centered scaled Hoeffding trees underperform the dynamically trained decision tree towards the end, resulting in poorer predictions. The remaining OML approaches perform significantly better than the BML approaches, with the z-scaled approach standing out, experiencing almost no change in MAE due to label drift. In the second case, a similar procedure is followed, except that all models are trained on the first 6,000 data points. The dynamic model is still updated every 2,000 data points. The results of the different approaches are summarized in Fig. 9.13.

From Fig. 9.13, it can be observed that towards the end of the data sequence, all OML approaches outperform their BML counterparts. In this problem, the min-max transformation and the max-abs transformation emerge as clear winners. Moreover, Fig. 9.13 demonstrates a pronounced peak at the initial phase for both the min-max

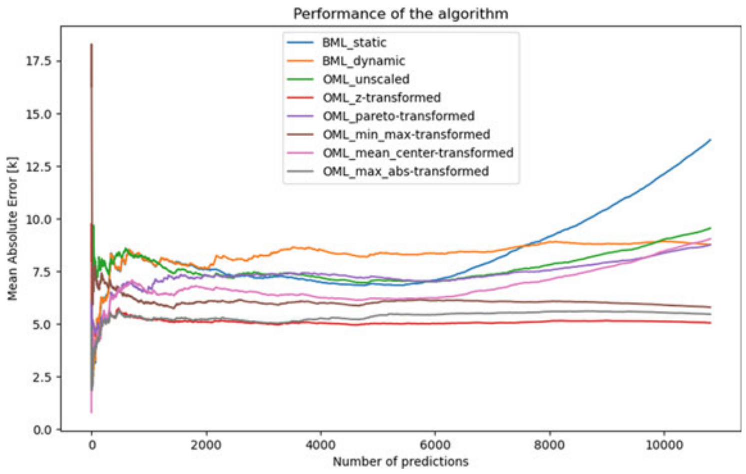


Fig. 9.12 Performance of the different model approaches as the number of observed data points increases for the label drift problem. MAE plotted against number of predictions

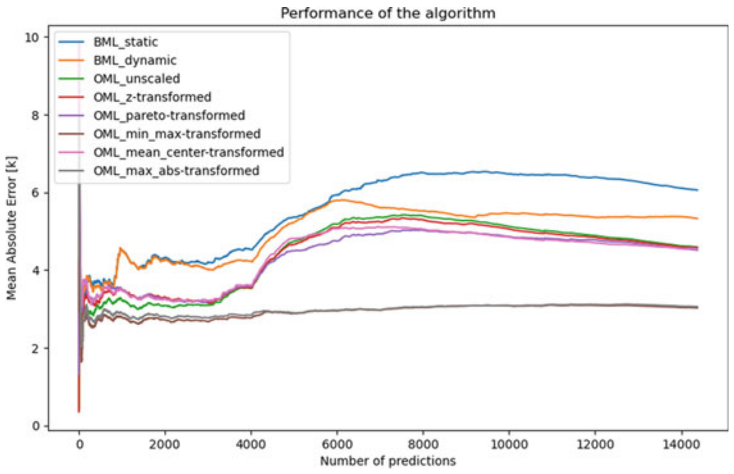


Fig. 9.13 Performance of the different model approaches when tackled with concept drift. Error plotted against the number of observed data points

and max-abs transformations. This phenomenon can be attributed to the inherent nature of these scaling methods, which tends to yield more conservative estimates, consequently resulting in elevated MAE values when encountering outliers in the data set.

9.4 Summary

To conclude this chapter, the examples in Sect. 9.3 reinforce the significance of effective data scaling in an online setting, showcasing its potential to yield enhanced results. Nevertheless, it becomes apparent that there is no universally optimal scaling method. Rather, the key lies in the comparison of diverse approaches to identify the most suitable one for a particular scenario. By carefully considering the context and requirements, data practitioners can make informed decisions to leverage the power of scaling techniques effectively.

The No-Free Lunch Theorem (Wolpert et al., 1997) also applies to OML algorithms (Haftka & Raphael, 2016). Two trade-offs could be observed:

1. For manageable data sets (e.g., 10,000 observations), BML algorithms are better but require more resources.
2. For larger data sets (e.g., 1,000,000 observations), OML algorithms are better and are not affected by drift, but require additional resources (time and memory requirements).

Figure 9.14 compares, in a highly simplified manner, the performance (error, time, and memory requirements) of the BML and OML algorithms, using flexibility as the basis. Flexibility in this context is defined as the ability of an algorithm to adapt to the data. For example, it can be interpreted as "the number of coefficients (or hyperparameters) in a model".

Thus, the model $f_2(x) = b_0 + b_1x + b_2x^2$ is more flexible than the model given by $f_1(x) = b_0 + b_1x$. This definition is not exact, but it is sufficient for the purposes of this study. It is based on the considerations of James et al. (2021).

In our context, linear regression models are on the left of the "flexibility axis", while simple regression trees are in the middle. This is followed by sophisticated methods such as gradient boosting, gbrt, or even more complex methods such as HATR which require many hyperparameters.

Figure 9.14a schematically compares the error[8] of BML and OML methods as a function of flexibility: simple BML methods are inferior to OML methods in our scenarios, as shown in Fig. 9.3. From Fig. 9.8 it became clear that the more complex BML methods like gradient boosting perform better than the OML methods.

Figure 9.14b compares the time required. Here, the simple OML methods are initially superior to the BML methods. However, the situation reverses when more complex algorithms are compared. In particular, the HATRs require more time than the BML methods, which only need to be trained once at the beginning. The OML procedures require more time because they need to be retrained for each new data point.

Figure 9.14 compares the algorithms in terms of memory requirements. Here, the simple OML procedures are significantly better than the BML procedures and similar to the time requirement, the situation reverses when more complex algorithms are

[8] The error is considered on the test data, so overfitting is visible. On the training data, the error would continuously decrease with the increase of flexibility.

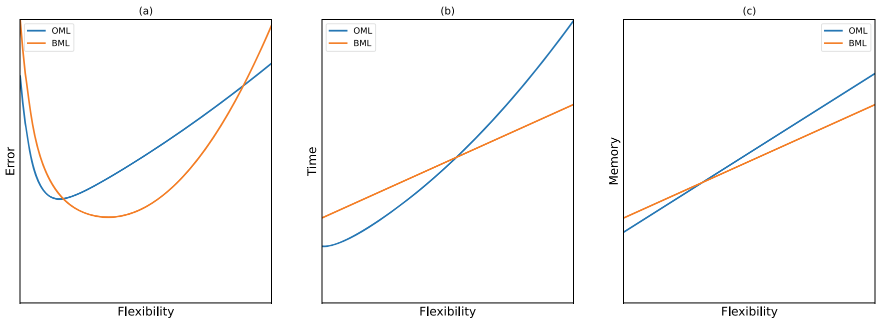


Fig. 9.14 Flexibility versus performance. Algorithms with few hyperparameters such as a simple linear regression model are less flexible than a complex algorithm such as HATR. The figures schematically compare error **a**, time requirements **b**, and memory requirements **c** for BML and OML methods with different flexibilities (in each case, smaller values are better). The comparison is based on considerations in James et al. (2021)

compared. The HATRs require more memory than the BML methods, although the difference is not as great as in the case of time requirements.

From these observations, the question arises whether the OML trees can be optimized by Hyperparameter Tuning (HPT). This will be investigated in Chap. 10.

References

Bifet, A., & Ricard, G. (2009). Adaptive Learning from Evolving Data Streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII. IDA '09* (pp. 249–260). Berlin: Springer.
Fanaee-T, H., & Joao, G. (2014). Event labeling combining ensemble detectors and background knowledge. In *Progress in Artificial Intelligence 2.2* (pp. 113–127).
Friedman, J.H. (2001). Greedy function approximation: A gradient boosting machine. In *The Annals of Statistics 29.5* (pp. 1189–1232).
Haftka, R.T. (2016). Requirements for papers focusing on new or improved global optimization algorithms. In *Structural and Multidisciplinary Optimization 54.1* (p. 1).
James, G., et al. (2021). *An introduction to statistical learning with applications in R*. (2nd ed.). Springer.
Ke, G., et al. (2017). LightGBM: A highly efficient gradient boosting decision tree. In I. Guyon et al (Eds.), *Advances in Neural Information Processing Systems, vol. 30*. Curran Associates, Inc.
Montiel, J., et al. (2021). River: Machine learning for streaming data in python. In *J. Mach. Learn. Res. 22*(1), 4945–4952. Issn: 1532-4435.
Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in python. *In Journal of Machine Learning Research, 12*, 2825–2830.
Wolpert, D.H., & William, G.M. (1997). No free lunch theorems for optimization. In *IEEE Transactions on Evolutionary Computation 1.1* (pp. 67–82).

Chapter 10
Hyperparameter Tuning

Thomas Bartz-Beielstein

Abstract The Online Machine Learning (OML) methods presented in the previous chapters require the specification of many hyperparameters. For example, a variety of "splitters" are available for Hoeffding trees to generate subtrees. There are different methods for limiting the tree size in order to keep the time and memory requirements within reasonable limits. In addition, there are many other parameters, so that a manual search for the optimal hyperparameter setting is very time-consuming and doomed to fail due to the complexity of the possible combinations. Therefore, this chapter explains how an automatic optimization (or "tuning") of the hyperparameters can be performed. In addition to the optimization of the OML procedure, Hyperparameter Tuning (HPT) performed with the Sequential Parameter Optimization Toolbox (SPOT) is also important for the explainability and interpretability of OML procedures and can lead to a more efficient and thus resource-saving algorithm ("Green IT").

10.1 Hyperparameter Tuning: An Introduction

Optimizing the hyperparameters is an important but usually difficult and computationally intensive task. The goal of HPT is to optimize the hyperparameters in a way that improves the performance of the Machine Learning (ML) model. The simplest approach, but also the most computationally expensive, uses manual search (or trial-and-error) Meignan et al. (2015). Commonly encountered is simple Random Search (RS), i.e., random and repeated selection of hyperparameters for evaluation, and grid search. In addition, methods that perform directed search and other model-free algorithms, i.e., algorithms that do not explicitly rely on a model, e.g., evolution strategies Bartz-Beielstein et al. (2014) or pattern search Lewis et al. (2000) play an important role. Also, "hyperband", i.e., a multi-armed bandit strategy that dynamically allocates resources to a set of random configurations and uses successive bisections to stop

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_10

configurations with poor performance Li et al. (2016), is very common in the HPT domain. The most sophisticated and efficient approaches are Bayesian Optimization (BO) and Surrogate Model Based Optimization (SMBO) methods, which are based on the optimization of cost functions obtained through simulations or experiments.

In the following, we consider an HPT approach based on SPOT (Bartz-Beielstein et al., 2005), which is suitable for situations in which only limited resources are available. This may be due to limited availability and the cost of hardware, or due to the fact that confidential data may only be processed locally, e.g., because of legal requirements. Furthermore, in our approach, the understanding of algorithms is seen as a key tool for transparency and explainability. This can be enabled, for example, by quantifying the contribution of ML and Deep Learning (DL) components (nodes, layers, split decisions, activation functions, etc.). As discussed in Sect. 6.6, understanding the meaning of hyperparameters and the interactions between multiple hyperparameters plays a major role in the interpretability and explainability of ML models. SPOT provides statistical tools for understanding hyperparameters and their interactions. Last but not least, it should be noted that the SPOT software code is available in the open-source Sequential Parameter Optimization Toolbox for Python (spotPython) and Sequential Parameter Optimization Toolbox for River (spotRiver) packages on GitHub,[1] allowing replicability of the results. SPOT is an established open-source software that has been maintained for more than 15 years (Bartz-Beielstein et al., 2005; Bartz et al., 2022).

10.2 The Hyperparameter-Tuning-Software SPOT

SMBO methods are common approaches in simulation and optimization. SPOT was developed because there is a great need for sound statistical analysis of simulation and optimization algorithms. SPOT includes methods for tuning based on classical regression and analysis of variance techniques, it provides tree-based models such as Classification And Regression Tree (CART) and Random Forest (RF) as well as BO (Gaussian Process Models, also known as Kriging) and combinations of different meta-modeling approaches. SPOT is implemented in Python and each model from the Python package scikit-learn: Machine Learning in Python (sklearn) can be used as a meta-model. SPOT is available in the open source packages spotPython and spotRiver on GitHub. SPOT implements key techniques such as exploratory fitness landscape analysis and sensitivity analysis. SPOT can be used to understand algorithm performance and gain insight into algorithm behavior. In addition, SPOT can be used as an optimizer and for automatic and interactive tuning. Details on SPOT and its application in practice are given by Bartz-Beielstein et al. (2021). A typical HPT process with SPOT consists of the following steps:

[1] https://github.com/sequential-parameter-optimization.

1. Loading the data (training and test data sets), see Sect. 10.3.1.
2. Specification of the preprocessing model, see Sect. 10.3.2. This model is called `prep_model` (“preparation” or preprocessing). The information required for the HPT is stored in the dictionary `fun_control`. Thus, the information needed to execute the HPT is available in a readable form.
3. Selection of the algorithm, see Sect. 10.3.3. This is called the `core_model`. Once the `core_model` is defined, then the associated hyperparameters can be stored in the `fun_control` dictionary. First, the hyperparameters of the `core_model` are initialized with the default values of the `core_model`. As default values, we use the default values contained in the spotRiver package for the algorithms of the River package, which are in JSON format.
4. For modification of the default values for the hyperparameters used in the `core_model`, see Sect. 10.3.4. This step is optional. Numeric parameters are modified by changing the limits (“bounds”), whereas categorical parameters are modified by changing the categories (“levels”).
5. Selection of the target function (loss function), see Sect. 10.3.5.
6. Calling SPOT with the corresponding parameters, see Sect. 10.3.6. The results are stored in a dictionary and are available for further evaluation.
7. Presentation, visualization and interpretation of the results, see Sect. 10.3.8.

10.3 Study: Hyperparameter Tuning of the HATR Algorithm on the Friedman-Drift Data

In this study, the hyperparameters of the Hoeffding Adaptive Tree Regressor (HATR) algorithm are tuned for predictions using the Friedman-Drift data.

Notebook: Experiments
The Jupyter Notebook in the GitHub repository https://github.com/sn-code-inside/online-machine-learning/ documents how the experiments were performed.

10.3.1 Loading the Data

At the beginning of an HPT study, an empty dictionary named `fun_control` is created. This is filled with the parameters needed to run the HPT in the next steps. It contains information about the data set, the preprocessing model, the algorithm to be tuned, the hyperparameters of the algorithm, the target function, and the weights for the target function. The `fun_control` dictionary is passed to the SPOT constructor.

This dictionary is used to store the information required for the HPT in a readable form.

We use the Friedman-Drift data set from the River[2] package. This data set was introduced in Definition 1.8 and described earlier in Sect. 9.2.1. We will consider a stream of 1,000,000 examples, each with ten features and a numerical target. Each feature value is sampled uniformly in [0, 1]. Only the first five features are relevant. The target is defined by different functions depending on the type of the drift. Global recurring abrupt drift will be used, i.e., the concept drift appears over the whole instance space. There are two points of concept drift. At the second point of drift, the old concept reoccurs. The following parameters are used to generate and handle the data set:

- horizon: The prediction horizon in hours.
- n_samples: The number of samples in the data set.
- p_1: The position of the first concept drift.
- p_2: The position of the second concept drift.
- position: The position of the concept drifts.
- n_train: The number of samples used for training.

We will use spotRiver's `convert_to_df` function to convert the River data set to a pandas data frame.

10.3.2 Specification of the Preprocessing Model

The next step is the creation of the preprocessing model ("`prep_model`"). First, the model components that do not use hyperparameters are created. This includes in particular all components with which the data preprocessing is carried out, e.g., the scaler or the selection function for the variables. This model is called `prep_model`. We use the `StandardScaler` from River to standardize the data set, i.e., it has zero mean and unit variance.

10.3.3 Selection of the Algorithm to be Tuned and the Default Hyperparameters

The next step is to select the algorithm whose hyperparameters are to be tuned. This is called the `core_model`. The spotPython hyperparameter tuning approach uses two components: a model (class) and an associated hyperparameter dictionary. Here, the River model class `HoeffdingAdaptiveTreeRegressor` is selected. The corresponding hyperparameters are loaded from the associated dictionary, which is

[2] https://riverml.xyz/0.15.0/api/datasets/synth/FriedmanDrift/.

Table 10.1 Hyperparameters of the HTR Algorithm

Parameter	Description
`grace_period`	Number of instances a leaf should observe between splitting trials
`max_depth`	Maximum depth a tree can reach
`delta`	Significance level for calculating the Hoeffding bound. The significance level is given by 1–delta. Values closer to zero imply longer delays in the split decision
`tau`	Threshold below which splitting is forced to break ties ("equality")
`leaf_prediction`	Prediction mechanism used with leaves. `mean`: average, `model`: uses the model defined in `leaf_model`, and `adaptive`: dynamically chooses between `mean` and `model`
`leaf_model`	Regression model used to provide answers when `leaf_prediction = model`
`model_selector_decay`	Exponential decay factor applied to the squared errors of the learning models monitored when `leaf_prediction='adaptive'`. It must be between 0 and 1. The closer it is to 1, the more importance is given to past observations. On the other hand, if the value approaches 0, the most recently observed errors have a greater influence on the final decision.
`splitter`	The splitter or attribute observer is used to monitor class statistics of numerical features and perform splits
`min_samples_split`	The minimum number of samples for performing a split
`binary_split`	If `True`, only binary splits will be performed
`max_size`	The maximum size of the tree, in megabytes (MB)

stored as a JSON file.[3] The JSON file contains hyperparameter type information, names, and bounds. The method `add_core_model_to_fun_control` adds the model and the hyperparameter dictionary to the `fun_control` dictionary.

Since the HATR is implemented as an extension of the Hoeffding Tree Regressor (HTR), the default values of the hyperparameters of the HTR are presented first. The corresponding hyperparameters are listed in Table 10.1.

The hyperparameters used by the HATR in addition to the hyperparameters of the HTR, which provide further functionalities for adaptive drift detection, are shown in Table 10.2.

The `prep_model` is combined with the `core_model` model: The tuner creates the full model by combining the `prep_model` with the `core_model` in a pipeline. Thus, the hyperparameters can be passed to the model at runtime (during tuning). Finally, the model is assembled in spotRiver. Afterwards, the default

[3] https://github.com/sequential-parameter-optimization/spotRiver/blob/main/src/spotRiver/data/river_hyper_dict.json.

Table 10.2 Hyperparameters of the HATR algorithm, which are used in addition to the hyperparameters of the HTR algorithm from Table 10.1

Parameter	Description
`bootstrap_sampling`	If `True`, bootstrap sampling is performed in the leaf nodes
`drift_window_threshold`	Minimum number of examples that an alternative tree must observe before it can be considered as a potential replacement for the current tree
`switch_significance`	Significance level to assess whether the alternative subtrees are significantly better than their main sub-tree counterparts
`memory_estimate_period`	Interval (number of instances processed) between memory usage checks
`merit_preprune`	If `True`, merit-based prepruning is enabled. Removes parts of the tree that do not contribute to the classification
`stop_mem_management`	If `True`, stops growth, once the memory limit is reached
`remove_poor_attrs`	If `True`, "bad" attributes will be are disabled to reduce memory consumption

values for the hyperparameters are chosen according to the selected algorithm. For the algorithm HATR, the corresponding hyperparameters[4] are determined (including type information, names and bounds). Here we use the default values contained in the package spotRiver for the algorithms of the package River. The value ranges of the hyperparameters are shown in Table 10.4.

10.3.4 Modification of the Default Values for the Hyperparameters

The hyperparameters used in the `core_model` can be modified. This step is optional.

1. Numeric parameters are modified by changing the limits ("bounds"). This also includes Boolean parameters whose levels are coded as 0 for False and 1 for True.
2. Categorical parameters are modified by changing the categories ("levels").

If identical values are specified for the limits, the value is interpreted as a constant parameter. This parameter is no longer considered in the tuning process.

[4] https://riverml.xyz/0.15.0/api/tree/HoeffdingAdaptiveTreeRegressor/.

10.3.5 Selection of the Target Function (Loss Function)

The HPT simultaneously considers the error (Mean Absolute Error (MAE), y_1), the time (seconds, y_2) and the memory requirement (MB, y_3). The three values are combined in a weighted manner, whereby the weights are specified via the parameter `weights` in the `fun_control` dictionary:

$$y = \sum_{i=1}^{3} w_i \times y_i .$$

In our example, the weights $w_1 = 1$ and $w_2 = w_3 = 1/1000$ are chosen, since the error reduction is in the foreground. This weighting is called "vertical weighting".

Tip
In a pre-experiment, reasonable orders of magnitude for the weights can be determined.

The parameter `weight_coeff` allows a weighting along the time axis, which is described as "horizontal weighting". Current values can be given a greater weight than older values. If the value zero is chosen for `weight_coeff`, all values are weighted equally. If a value greater than zero is chosen, the values are weighted exponentially.

10.3.6 Calling the Hyperparameter Tuner SPOT

An instance of the class Spot is created. Finally, the run method is called, which starts the hypertuning process.

10.3.7 Visualization with TensorBoard

Now we can start TensorBoard in the background with the following command, where './runs' is the default directory for the TensorBoard log files:

```
tensorboard --logdir="./runs" --port=6006 &
```

The TensorBoard web server can be accessed with the following URL: https://localhost:6006/. The TensorBoard visualization of the hyperparameter tuning process is shown in Fig. 10.1.

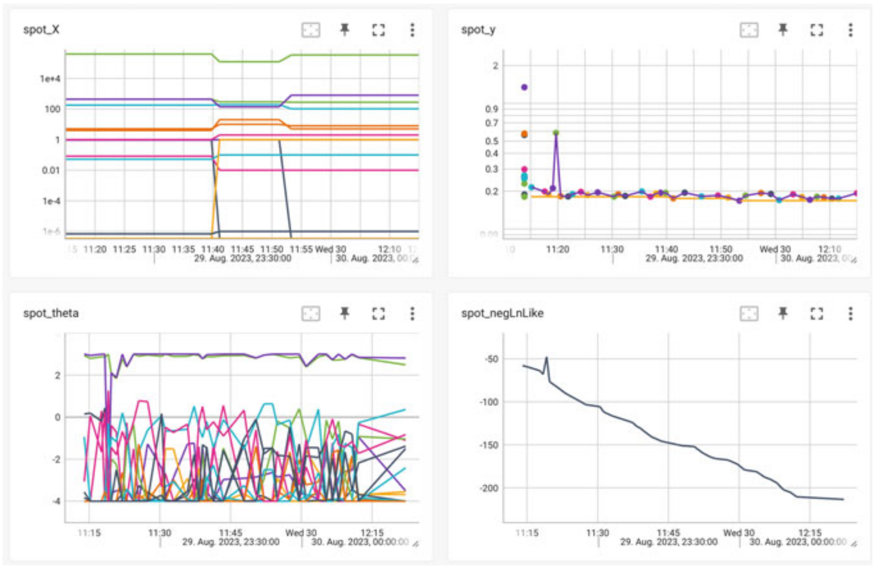


Fig. 10.1 TensorBoard visualization of the hyperparameter tuning process. A detailed description of the TensorBoard visualization is given in Bartz-Beielstein (2023). Updates of the hyperparameter tuning cookbook are available at https://sequential-parameter-optimization.github.io/Hyperparameter-Tuning-Cookbook/

10.3.8 HATR Tuning Results

Figure 10.2 shows the progress of the hyperparameter tuning.[5] The HPT algorithm was given a budget of 60 minutes. About 40 hyperparameter configurations were calculated. The values generated for the initial design of SPOT are shown as *black dots*. The *black line* represents the value of the best hyperparameter configuration found during the initial design. The first surrogate model is created using the points evaluated during the initial design. Only after the initial design has been evaluated does the tuning (optimization) start. The points generated during the HPT are shown in *red*.

The best hyperparameter configuration can be displayed using the `print_results` method. A more detailed output, which allows a direct comparison of the default settings with the optimized settings and also takes into account the relative importance, can be generated using the `gen_design_table` method. The output is shown in Table 10.4.

[5] The tuning was performed on a MacBookPro (Apple M2 Max Chip, 12-Core CPU, 38-Core GPU, 96 GB memory).

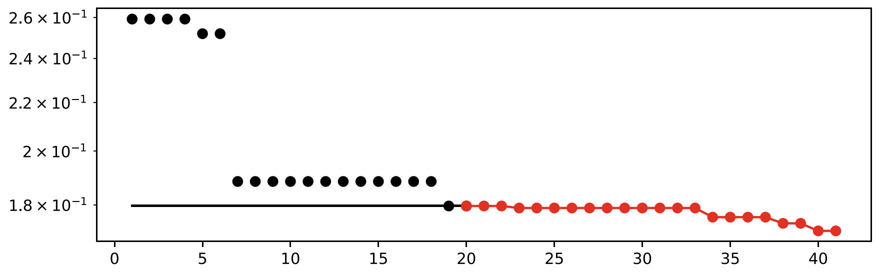


Fig. 10.2 Progress of the hyperparameter optimization with SPOT. The values of the target function are shown on the y-axis, the x-axis shows the number of evaluated hyperparameter configurations. *Black* is used for the evaluations of the initial design, *red* for the surrogate evaluations

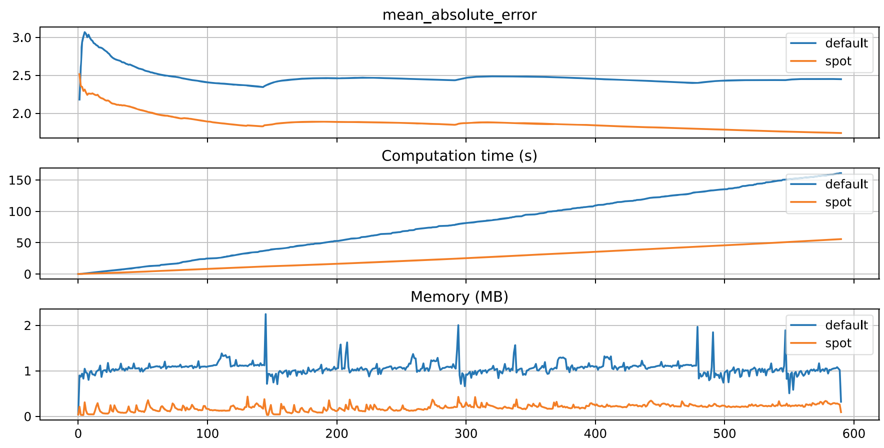


Fig. 10.3 Comparison of error, time, and memory requirement. Default (*blue*) versus SPOT (*orange*). The tuning reduced the error as well as the time and memory requirement so that an overall more efficient setting was found. This figure shows the results for 100,000 observations. The predictions for one week (7 times 24 hours) are shown, so that about 600 evaluations are available

Tip
In most experiments, a good value for the target variable was already determined during the initial design phase. The improvements achieved in the further course of the HPT are relatively small. However, in practical use, they can provide the decisive advantage.

Since the tuning was performed on the reduced data set with one hundred thousand observations, we first consider the results for this data set. The comparison of the performance (error, i.e., MAE, time and memory requirement) is shown in Fig. 10.3.

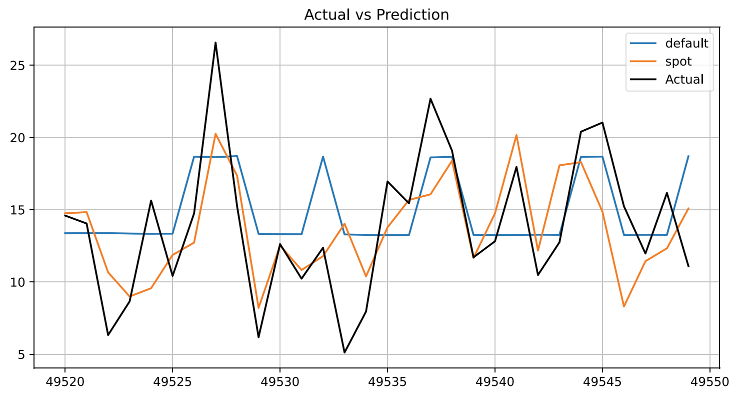


Fig. 10.4 Comparison of the residuals. The actual observations are shown by a *black* line. Compared to the HATR algorithm with default hyperparameters (shown in *blue*), the tuned algorithm (*orange*) reacts more flexibly to changes in the data stream due to the optimized hyperparameters

The MAE was reduced by SPOT compared to the default setting from about 2.5 to values that are consistently smaller than 2.0. An improvement was achieved here. Similar results are also obtained for the time and memory requirements. While the evaluation time of the HATR model with default settings after 100,000 observations takes about 150 seconds, the algorithm optimized with SPOT only needs 50 seconds. The memory requirement for the default algorithm is about 1 MB, while the algorithm optimized with SPOT needs less than half. It should be noted that the HATR algorithm uses internal memory management algorithms, which can lead to fluctuations in the memory requirements. The fluctuations are clearly visible in the figures.

A comparison of the actual and the predicted values is shown in Fig. 10.4. They illustrate that the settings found with SPOT improve the flexibility of the HATR algorithm: While the default settings lead to a conservative, mean-approximating behavior, the optimized algorithm is more risk-taking and better reflects jumps in the data stream.

The improvement of the HATR algorithm with SPOT can also be seen while comparing the residuals. Fig. 10.5 shows the residuals of the HATR algorithm with default hyperparameters, whereas Fig. 10.6 shows the residuals of the tuned algorithm. The residuals of the tuned algorithm are smaller than the residuals of the default algorithm.

It is very interesting to answer the question of whether the results found with a reduced data set can be transferred to a larger data set. The evaluation of the HATR algorithm with default and SPOT-optimized hyperparameters for one million observations is shown in Fig. 10.7. In general, a similar picture emerges as with the reduced data set: The hyperparameter tuning was able to visibly reduce errors, time, and memory requirements.

However, the visualization of the memory requirement of the optimized HATR algorithm in the interval between 5,000 and 6,000 observations, i.e., towards the

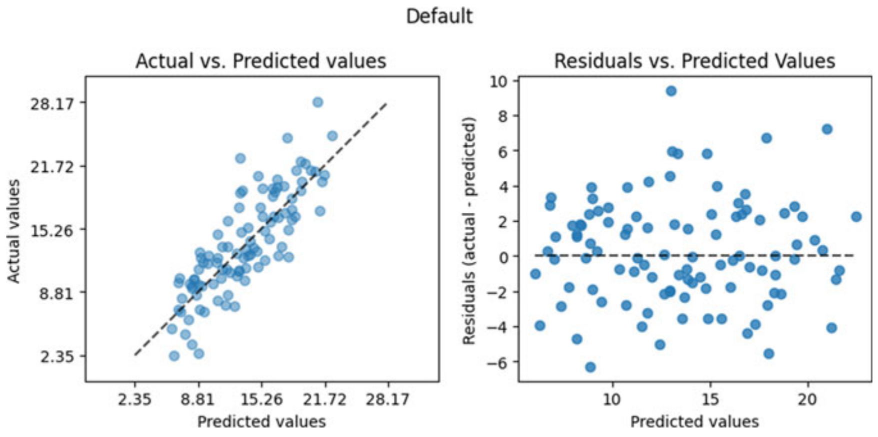


Fig. 10.5 Residuals of the HATR-algorithm with default hyperparameters

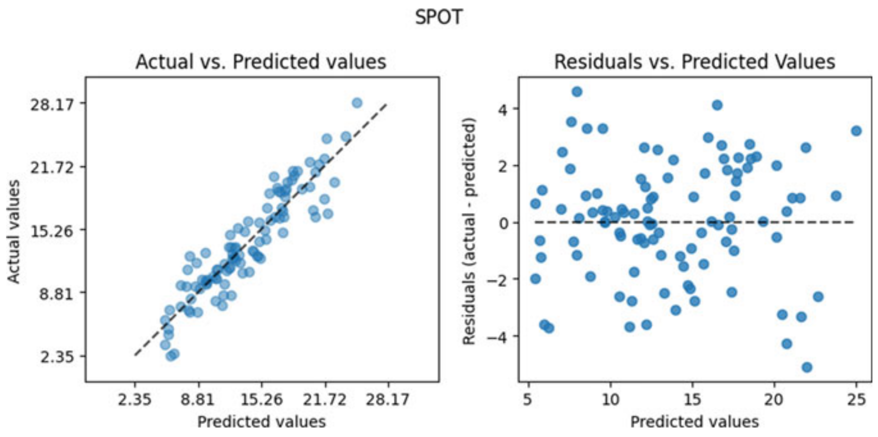


Fig. 10.6 Residuals of the HATR-algorithm with tuned hyperparameters. A comparison with Fig. 10.5 shows that the residuals of the tuned algorithm are smaller than the residuals of the default algorithm

end of the observation period, shows an anomaly. A single peak occurs, indicating a memory requirement of about 60 MB. In further investigations, it has to be clarified whether this is a single case caused by the operating system and not by the HATR algorithm, or whether it is a bug in the HATR algorithm.

The analysis of the residuals for one million observations showed no differences from the already discussed analysis of the reduced data set (see Fig. 10.4). Therefore, the corresponding figure is not shown. Details of the analyses can be found in the notebooks belonging to this chapter.

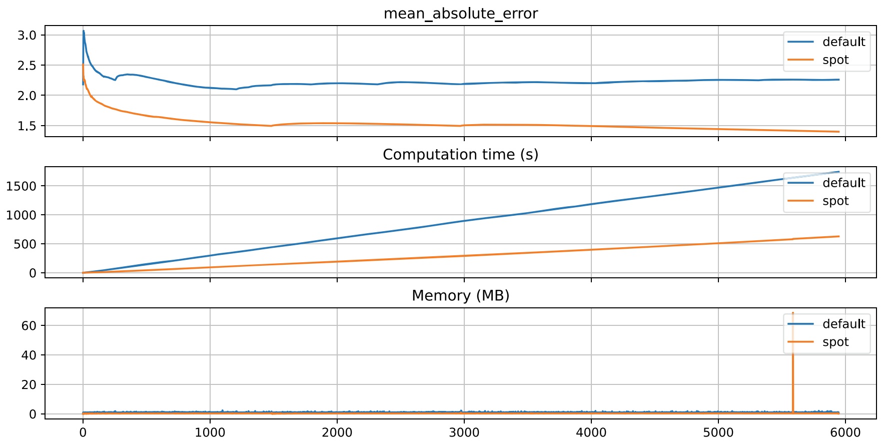


Fig. 10.7 Comparison of error, time, and memory requirement for 1 million observations. Default (*blue*) versus SPOT (*orange*). The figure illustrates the transferability of the results found on a reduced data set (see Fig. 10.3) to a larger data set. An anomaly in the memory requirement is noticeable towards the end of the observation period

Tip
The transferability of the results found on a reduced data set to a larger data set was investigated. The results show that the settings found with SPOT can also be transferred to a larger data set.

10.3.9 Explainability and Understanding

To calculate the importance or effect of individual hyperparameters, SPOT provides a method that uses the "Activity" or "Width" of the parameters of the surrogate model (Forrester et al., 2008; Bartz et al., 2022). The symbol θ is often used in the literature for these parameters (Forrester et al., 2008). The relative importance of a hyperparameter is calculated as a relation to the importance of the most important hyperparameter. Accordingly, the most important parameter has a relative importance of 100%. The relative importance of individual hyperparameters can be displayed by calling the method `spot_hatr.print_importance`. These values are visualized in Fig. 10.8 and are also shown in Table 10.4.

In our study, the parameters `leaf_prediction` and `leaf_model` have the greatest effect. These are followed by `delta` and `bootstrap_sampling`, whose effects were only drawn in for comparison.

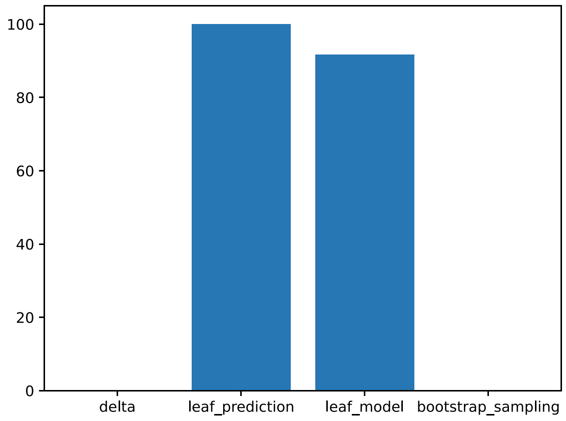


Fig. 10.8 Relative importance of the hyperparameters. `leaf_prediction` and `leaf_model` are by far the most important parameters

Table 10.3 Coding of categorical hyperparameters

Name	Level 0	Level 1	Level 2
`leaf_prediction`	`mean`	`model`	`adaptive`
`leaf_model`	`LinearRegression`	`PARegressor`	`Perceptron`
`splitter`	`EBSTSplitter`	`TEBSTSplitter`	`QOSplitter`

SPOT uses a numerical coding for the levels of the categorical hyperparameters. The categorical values are coded as shown in Table 10.3. The setting `leaf_prediction` with the value `mean` corresponds to the numerical value `0`, the setting `model` corresponds to the numerical value `1` and the setting `adaptive` corresponds to the numerical value `2`. Table 10.4 shows the hyperparameter values of the HATR algorithm.

It is also interesting to examine the structure of the internal Hoeffding tree models. Table 10.5 compares the attributes of the regression trees of the default and the HATR model optimized with spotPython. The values shown represent a snapshot of the activities in the individual elements (nodes and branches of the HATR). They indicate that the tuning reduces the complexity of the trees. However, this question still needs further investigation. For example, the representation of the course of the number of active and inactive leaves is an interesting analysis, which has not yet been fully carried out.

Figure 10.9 visualizes the interaction of the two most important hyperparameters, `leaf_prediction` and `leaf_model` of the surrogate model used to optimize the hyperparameters. Since both hyperparameters take categorical values (with three levels each), a step-like fitness landscape (or “response surface”) is generated. The hyperparameter configuration `leaf_prediction = 1` and `leaf_model = 2` leads to poor results. Therefore, SPOT recommends the setting `leaf_prediction = 1 (model)` and `leaf_model = 0 (Linear Regression)`. These results are in accordance with the results already discussed.

Table 10.4 Hyperparameter of the HATR algorithm, see Table 10.1. The relative importance is shown using the symbols: "***" very important, "**" important, "*" less important, "." slightly important. For the hyperparameter `max_depth` a 2^x transformation is performed, so that, e.g., the value `10` corresponds to the depth `1024`. The hyperparameters `leaf_prediction`, `leaf_model` and `splitter` are categorical hyperparameters. The levels are coded in Table 10.3

Hyperparameter	Typ	Default	Lower	Upper	Tuned	Importance	Stars
grace_period	int	200	10	1000	758	0.00	
max_depth	int	20	2	20	19	0.00	
delta	float	1e–07	1e–10	1e–06	1e–06	0.00	
tau	float	0.05	0.01	0.1	0.1	0.00	
leaf_prediction	factor	0	0	2	1	100.00	***
leaf_model	factor	0	0	2	0	91.75	**
model_selector_decay	float	0.95	0.9	0.99	0.9	0.00	
splitter	factor	0	0	2	1	0.00	
min_samples_split	int	5	2	10	8	0.00	
bootstrap_sampling	factor	0	0	1	0	0.02	
drift_window_threshold	int	300	100	500	101	0.00	
switch_significance	float	0.05	0.01	0.1	0.1	0.00	
binary_split	factor	0	0	1	0	0.00	
max_size	float	500.0	100.0	1000.0	789.80	0.00	
memory_estimate_period	int	1e6	1e6	1e6	938,558	0.00	
stop_mem_management	factor	0	0	1	1	0.00	
remove_poor_attrs	factor	0	0	1	0	0.00	
merit_preprune	factor	0	0	0	0		

Table 10.5 Comparison of the parameters of the default and the HATR model optimized with SPOT

Parameter	Default	Spot
`n_nodes`	151	149
`n_branches`	75	74
`n_leaves`	76	75
`n_active_leaves`	210	58
`n_inactive_leaves`	0	0
`height`	12	12
`total_observed_weight`	1e5	1e5
`n_alternate_trees`	33	34
`n_pruned_alternate_trees`	8	29
`n_switch_alternate_trees`	1	3

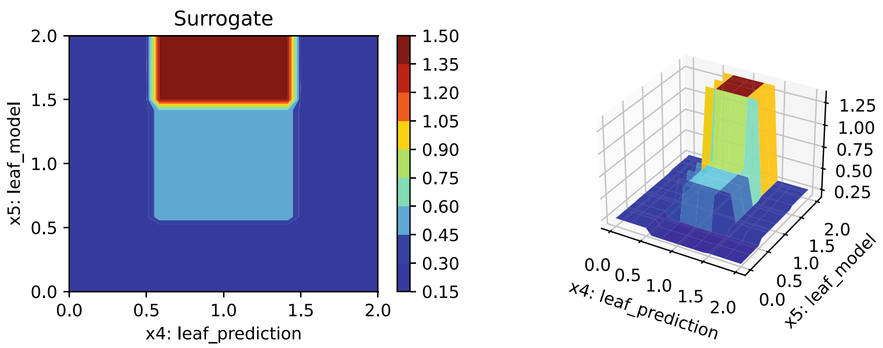


Fig. 10.9 Surrogate model. The figure on the *left* shows the influence of the hyperparameters `leaf_prediction` and `leaf_model` on the performance of the HATR algorithm. The figure on the *right* shows the same relationship as a 3D plot. Bad settings are shown in *dark red*

SPOT plots the interactions of the most important hyperparameters by default. All interactions can also be visualized. For this purpose, we refer again to the accompanying material to this chapter in the notebooks.

10.4 Summary

In this chapter, the HATR algorithm was analyzed and optimized using the HPT software SPOT. The HPT provides important insights into the importance of the individual hyperparameters. These analyses provide elementary building blocks for the understanding of complex algorithms in the field of OML and are therefore relevant for the explainability of OML algorithms.

It has been shown that results found on a reduced data set (or limited data stream) can be transferred to a larger data stream. This is an important aspect, since the optimization of the hyperparameters usually has to be performed on a reduced data set in order to reduce the computation time and memory requirements. For the tuning of the 17 hyperparameters of the HATR algorithm, 60 minutes were available on a standard notebook. At this time, about 40 configurations could be evaluated using the reduced data set (one tenth of the total volume). The error as well as the time and memory requirement could be reduced. However, when presenting the results, it should be noted that only individual runs were compared. In future analyses, a comparison of the results of several runs should therefore be performed.

The greatest improvement is achieved by selecting the best hyperparameter configuration of the initial design. A further improvement is possible with the help of the surrogate model. Here, the relatively large number of different hyperparameters and the small number of evaluations must be taken into account, so that the HPT is more similar to a screening than to an optimization. Nevertheless, important hints for the selection of suitable hyperparameter settings can be obtained. In particular, unfavorable settings can be detected.

Besides the algorithmic advantages of OML, there are additional ecological and economic advantages: By reducing the memory requirements and the computing time, the costs for providing OML algorithms can be reduced. This is of particular interest for Small and Medium-sized enterprises (SMEs), which often have only limited resources. By reducing the computing time, the algorithms can also be used in real time. An important, positive ecological effect results directly from the lower memory and time requirements, so that an improved ecological footprint can be achieved and the label "Green IT" can be used with a clear conscience.

References

Bartz, E., et al. (2022). *Hyperparameter tuning for machine and deep learning with R—A practical guide*. Springer. https://doi.org/10.1007/978-981-19-5170-1

Bartz-Beielstein, T. (2023). Hyperparameter tuning cookbook: A guide for scikit-learn, PyTorch, river, and spotPython. In *arXiv e-prints* arXiv:2307.10262. https://doi.org/10.48550/arXiv.2307.10262. [cs.LG].

Bartz-Beielstein, T., & Jürgen, B., et al. (2014). Evolutionary algorithms. In *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 4.3* (pp. 178–195).

Bartz-Beielstein, T., Lasarczyk, C., Preuss, M., et al. (2005). Sequential parameter optimization. In B. McKay (Ed.), *Proceedings 2005 Congress on Evolutionary Computation (CEC-05), Edinburgh, Scotland* (pp. 773–780). Piscataway NJ: IEEE Press.

Bartz-Beielstein, T., Martin, Z., & Frederik, R. (2021). In a nutshell—The sequential parameter optimization toolbox. In *arXiv e-prints* arXiv:1712.04076.

Forrester, A., Sóbester, A., & Keane, A. (2008). *Engineering design via surrogate modelling*. Wiley.

Lewis, R.M., Virginia, T., & Michael, W.T. (2000). Direct search methods: Then and now. In *Journal of Computational and Applied Mathematics 124.1–2* (pp. 191–207).

Li, L., et al. (2016). Hyperband: A novel bandit-based approach to hyperparameter optimization. In *arXiv e-prints* arXiv:1603.06560.

Meignan, D., et al. (2015). A review and taxonomy of interactive optimization methods in operations research. In *ACM Transactions on Interactive Intelligent Systems*.

Chapter 11
Summary and Outlook

Thomas Bartz-Beielstein and Eva Bartz

Abstract This chapter presents an assessment of the potential of Online Machine Learning (OML) for practitioners. The results of the studies are summarized and discussed and concrete recommendations for OML practice are given. The importance of a suitable comparison methodology for Batch Machine Learning (BML) and OML methods is highlighted to avoid "comparing apples to oranges". We also point out the great potential of OML that is available through the development of the open-source software River.

11.1 Necessity for OML Methods

The necessity for the use of OML methods is undisputed, since BML is no longer sufficient in many areas. However, the application of OML methods is not trivial. The advantages of OML must be reproducible, transparent, and reliably demonstrable. "How can BML and OML be compared?" is a central question that has been answered in this book.

For the comparison of BML and OML, novel, problem-adapted and promising methods have been developed and applied. Currently, there are only a few publications that compare BML, mini-batch, and OML methods. Therefore, this book is a pioneering work. For the first time, quality (performance), time, and memory requirements are compared for these three classes of algorithms and experimentally analyzed.

The consideration on how to select training and test data is central, since the procedure for BML and OML differs fundamentally. In this book, open-source software tools for the evaluation of batch, mini-batch, and online learning methods were

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering, and Analytics, TH Köln, Gummersbach, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz
Bartz & Bartz GmbH, Gummersbach, Germany
e-mail: eva.bartz@bartzundbartz.de

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*,
Machine Learning: Foundations, Methodologies, and Applications,
https://doi.org/10.1007/978-981-99-7007-0_11

described and applied, with which BML and OML algorithms for regression and classification in the context of batch learning, mini-batch learning, and online learning can be compared. For this purpose, the visualization of the following three criteria for the entire data stream is central:

- quality (in particular accuracy and Mean Absolute Error (MAE)),
- time and
- memory requirements.

The package Sequential Parameter Optimization Toolbox for River (spotRiver) provides tools to experimentally analyze arbitrary BML and OML methods with suitable quality criteria on different data sets.

After a basic introduction to OML, state-of-the-art methods for batch, mini-batch, and online learning were introduced. The focus lies on practical applicability. Therefore, the methods in this book were presented and discussed using real-world examples. Two studies and a hyperparameter tuning were carried out using publicly and freely available data sets. The quality, runtime, and memory requirements were calculated as comparison metrics for the individual BML and OML models.

11.2 Recommendations for Using OML in Practice

Based on the studies and experiments presented in this book, the following recommendations for OML practice can be derived:

- Start simple: In the case of classification, the simplest logistic regression model should be used first, and in the case of regression, the simplest linear regression model. This approach allows an assessment of whether OML algorithms are suitable. The influence of data preprocessing ("preprocessing") should not be underestimated. Good feature generation can significantly improve the quality of the OML methods.
- Consider hyperparameters: The influence of the hyperparameters should be investigated. In particular, the adaptive methods developed for drift treatment should be considered for the tree-based OML methods.
- Question comparisons: The results of the BML and OML algorithms are only partially directly comparable, even if the same quality measures are used, since the comparisons use different data sets. The selection of suitable data and the selection of quality measures are therefore two important steps that were described in Chap. 5.
- Consider OML peculiarities in comparisons: The poor performance of the OML methods compared to the BML method may need to be reconsidered, depending on the task. It can reflect a desired effect of the OML methods: They are able to adapt to the current data situation. This can be the case if the OML method is trained on the current data, but the entire data set is used to evaluate it. If drift occurs in this case, the quality of the OML learner on the entire data set can be

worse than the quality of the BML method, which has the entire data set available for learning (or at least a relatively large training data set). On the other hand, if only the current period is used to evaluate the OML method, the quality of the OML method can be much better. Therefore, in the further course of the study, in addition to the comparisons of the batch and online methods, we also included the comparison of the mini-batch methods, see also Definition 1.11.

- Estimate memory and time requirements: Hoeffding trees become relatively large and therefore require a lot of memory. Thus, a detailed investigation of the methods for limiting the tree size is required. The preparation of an experimental plan (using Design of Experiments (DOE) or Design and Analysis of Computer Experiments (DACE)) is recommended to avoid excessively long run times for the experiments (Montgomery, 2017; Santner et al., 2003). Hyperparameter tuning is also a useful approach here.
- Expect surprises: In some cases, unexpected events occurred only after a long period of time. The importance and effects are difficult to estimate for practical use, since the software used is also still in a developmental stage. Therefore, it is not clear whether incorrect results or non-ending program runs and crashes were caused by the algorithm or by a faulty implementation.
- Observe software development: It is becoming apparent that the River package as state-of-the-art software will replace the previous OML software packages. River has great potential, since the package is actively developed further. It is currently available in version `0.21.0`. In particular, an interface to hyperparameter tuning methods such as Sequential Parameter Optimization Toolbox (SPOT) is useful (Bartz et al., 2022). In parallel with the further development of River, the spotRiver, and Sequential Parameter Optimization Toolbox for Python (spotPython) packages are being developed, which enable or simplify hyperparameter tuning for OML.

Note: OML Software

- Right now, there is no current OML software available that can be taken "off the shelf" and used immediately.
- In the foreseeable future, the field of OML could provide interesting solutions, especially for updating very large Machine Learning (ML) models.

References

Bartz, E., et al. (2022). *Hyperparameter tuning for machine and deep learning with R—A practical guide*. Springer. https://doi.org/10.1007/978-981-19-5170-1.

Montgomery, D. C. (2017). *Design and analysis of experiments*, 9th ed. Wiley.

Santner, T. J., Williams, B. J., & Notz, W. I. (2003). *The design and analysis of computer experiments*. Springer.

Appendix A
Definitions and Explanations

A.1 Gradient Descent

Definition A.1 Gradient descent. The gradient descent algorithm performs the following steps, where δ is the step size, λ is the learning rate, x_t are the old, x_{t+1} the new parameters, and ∇ is the gradient:

1. Initialization of the parameters x_t with random values.
2. Calculation of the gradient ∇ of the objective function f.
3. Calculation of the step size: $\delta = \lambda \cdot \nabla$.
4. Calculation of the new parameters: $x_{t+1} = x_t - \delta$.
5. Repeat steps 2–4 until the gradient is close to zero.

The learning rate is a hyperparameter that directly affects the convergence of the algorithm. With a very small learning rate, it takes a long time for the gradient descent algorithm to converge. A very large value of the learning rate causes the algorithm to choose large steps, so that it may miss the optimum.

A.2 Bayes' Theorem

Let $P(A)$ denote the (a-priori) probability of event A and $P(A \mid B)$ the (conditional) probability of event A under the condition that event B has occurred. If $P(B) > 0$, then

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}.$$

Thus, the probability of A under the condition that B has occurred can be calculated by the probability of B under the condition that A has occurred. This statement provides the basis for Bayes' theorem.

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-99-7007-0

Theorem A.1 *Bayes' theorem. For a finite number of events, the following applies: If A_i, $i = 1, \ldots, N$ is a partition (decomposition) of the result set into disjoint events, the a posteriori probability $P(A_i \mid B)$ is*

$$P(A_i \mid B) = \frac{P(B \mid A_i) \cdot P(A_i)}{P(B)} = \frac{P(B \mid A_i) \cdot P(A_i)}{\sum_{j=1}^{N} P\left(B \mid A_j\right) \cdot P(A_j)}.$$

A.3 Hoeffding Bound

Theorem A.2 *Hoeffding bound. For all $\epsilon \in (0, 1)$, it holds that*

$$P(|X - E(X)| > \epsilon) < 2\exp(-2\epsilon^2 n).$$

As a confidence interval for estimating the entropy in a node, it was proposed to use

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

where

- R is the range of values of the variable,
- δ is the desired probability that the estimated value X is not within the ϵ-neighborhood of the expected value $E(X)$ and
- n is the number of samples in the nodes.

If the information gain is chosen as the splitting criterion using the entropy, the range of values R of the entropy is between $[0, \ldots, \log(n_c)]$, if the class n_c can take different values.

A.4 Kappa Statistics

Definition A.2 Kappa statistic. Let p_0 be the prequential accuracy of the classifier and p_c ("c" = chance) the probability that a "chance" classifier, which randomly assigns instances to the classes, using the same percentages as the actual classifier, makes a correct prediction. Then the κ-statistic is defined as follows:

$$\kappa = \frac{p_0 - p_c}{1 - p_c}.$$

If the classifier is always correct, then $\kappa = 1$, if it is as "bad" as the chance classifier, then $\kappa = 0$.

Definition A.3 Kappa-M-Statistic. The Kappa-M-Statistic is defined as follows:

$$\kappa_m = \frac{p_0 - p_m}{1 - p_m},$$

where p_m is the probability that a majority classifier makes a correct prediction.

The majority classifier yields better values than the classifier if the class distribution of the predicted class differs strongly from the current class distribution.

Definition A.4 Kappa-Temporal-Statistic. The Kappa-Temporal-Statistic is defined as follows:

$$\kappa_{\text{per}} = \frac{p_0 - p'_e}{1 - p'_e},$$

where p'_e is the probability that a "no-change" classifier makes a correct prediction.

The Kappa-M and the Kappa-Temporal-Statistic are orthogonal measures: κ_{per} can detect sequences of equal data ("bursts") well (i.e., no changes), while κ_m detects changes.

Appendix B
Supplementary Materials

B.1 Notebooks

In addition to this book, interactive Jupyter Notebooks are provided in the GitHub repository https://github.com/sn-code-inside/online-machine-learning. These notebooks are organized by chapter. Table B.1 provides an overview. The repository is continuously maintained, so the notebooks may change over time.

Table B.1 Notebooks

Chapter	Notebook	Content
Chapter 1	`ch01.ipynb`	Introduction: From Batch to Online Machine Learning
Chapter 2	`ch02.ipynb`	Supervised Learning: Classification and Regression
Chapter 3	`ch03.ipynb`	Drift Detection and Handling
Chapter 4	`ch04.ipynb`	Initial Selection and Subsequent Update of OML Models
Chapter 5	`ch05.ipynb`	Evaluation and Performance Measurement
Chapter 6	`ch06.ipynb`	Special Requirements for OML Methods
Chapter 8	`ch08.ipynb`	Short introduction to River
Chapter 9	`ch09_bike.ipynb`	Bike-Sharing
Chapter 9	`ch09_friedman.ipynb`	Friedman Drift
Chapter 10	`ch10_friedman-hpt.ipynb`	Hyperparameter Tuning (HPT)

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-99-7007-0

B.2 Software

- The source code for the open-source software package `spotPython` can be found on GitHub: https://github.com/sequential-parameter-optimization/spotPython.
- The source code for the open-source software package `spotRiver` can be found on GitHub: https://github.com/sequential-parameter-optimization/spotRiver.

Glossary

ADWIN	Adaptive Windowing
AI	Artificial Intelligence
ALMA	Approximative Large-Margin-Algorithms
BML	Batch Machine Learning
BMWK	German Federal Ministry for Economic Affairs and Climate Action
BO	Bayesian Optimization
CART	Classification And Regression Tree
CV	Cross Validation
CVFDT	Concept-adapting Very Fast Decision Tree
DACE	Design and Analysis of Computer Experiments
DDM	Drift Detection Method
DL	Deep Learning
DOE	Design of Experiments
EFDT	Extremely Fast Decision Tree
FEM	Finite Element Method
gbrt	Gradient Boosting Regression Tree
GDP	Gross Domestic Product
GRA	Global Recurring Abrupt
HAT	Hoeffding Adaptive Tree
HATC	Hoeffding Adaptive Tree Classifier
HATR	Hoeffding Adaptive Tree Regressor
HPT	Hyperparameter Tuning
HT	Hoeffding Tree
HTC	Hoeffding Tree Classifier
HTR	Hoeffding Tree Regressor
IoT	Internet of Things
KPI	Key Performance Indicator
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MOA	Massive Online Analysis
MSE	Mean Squared Error

E. Bartz and T. Bartz-Beielstein (eds.), *Online Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-99-7007-0

NN	Neural Network
OML	Online Machine Learning
PA	Passive-Aggressive
RAM	Random Access Memory
RF	Random Forest
River	
RMOA	Massive Online Analysis in R
ROC, AUC	Area Under The Curve, Receiver Operating Characteristics
RS	Random Search
SEA	SEA synthetic dataset
SGD	Stochastic Gradient Descent
sklearn	scikit-learn: Machine Learning in Python
SMBO	Surrogate Model Based Optimization
SME	Small and Medium-sized enterprise
SMOTE	Synthetic Minority Oversampling Technique
SPF	Survey of Professional Forecasters
SPOT	Sequential Parameter Optimization Toolbox
spotPython	Sequential Parameter Optimization Toolbox for Python
spotRiver	Sequential Parameter Optimization Toolbox for River
SVM	Support Vector Machine
U.S.	United States
VFDT	Very Fast Decision Tree

Index