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

© 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_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 $(x_{t-k}, x_{t-k+1}, \ldots, x_{t-1})$, 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.

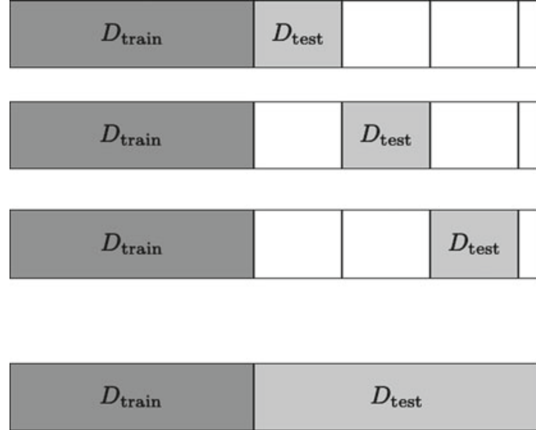

Fig. 5.1 Batch method with a prediction horizon. The training data set D_{train} is used once. The model M_{bml} trained on D_{train} is tested on the individual partitions of the test data set D_{test} one after the other. The lower figure shows (as a special case) the data sets when a classical holdout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

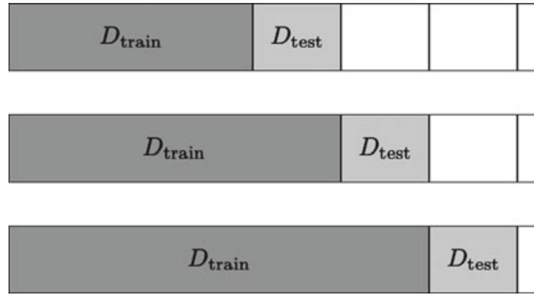

Fig. 5.2 Landmark batch method with an prediction horizon

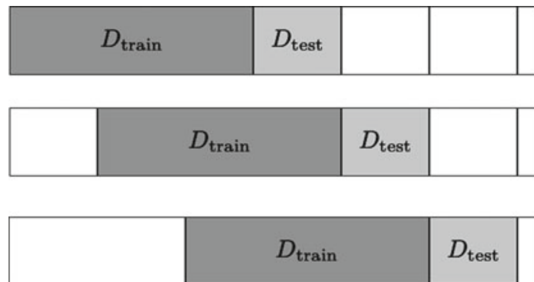

Fig. 5.3 Window-batch method with a prediction horizon. This division of the training and test data set ensures that the size of the training data set s_{train} remains unchanged and that a prediction horizon h of the same size is always used

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 $M^{(1)}_{bml}$ 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.

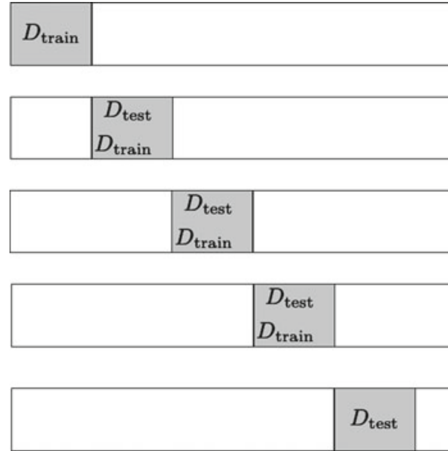

Fig. 5.4 Iterative OML method. If the window size h is one, then an example is used for testing and then for training (updating) the OML algorithm. If $h > 1$, then the calculations are performed h times and the average of these h results is calculated

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

```
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.

```
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

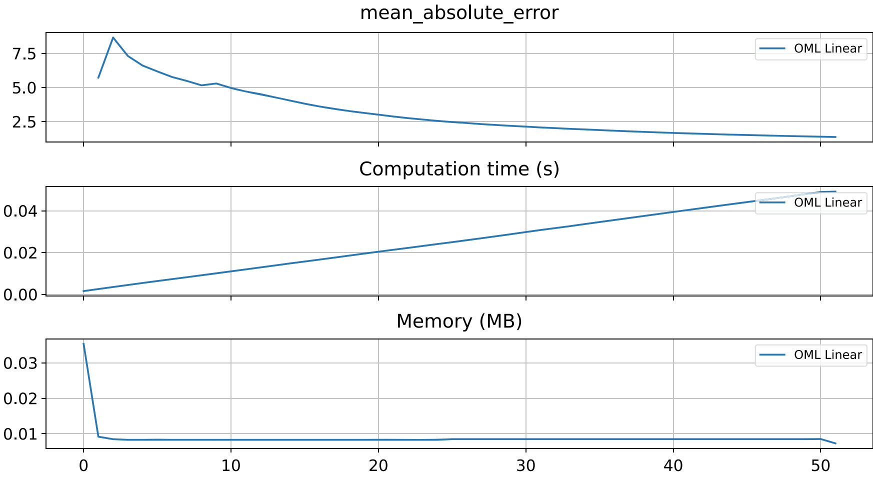

Fig. 5.5 Results of the method `plot_bml_oml_horizon_metrics`. Performance (here: MAE, computation time and memory consumption) of an OML linear model

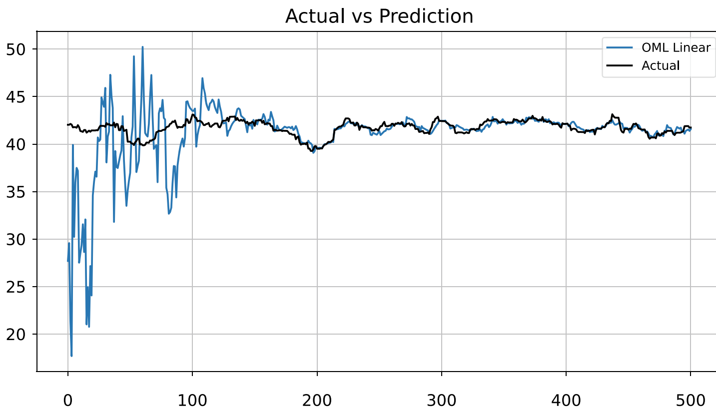

Fig. 5.6 Results of the method `plot_bml_oml_horizon_predictions`: Representation of the values predicted by the model and the ground truth ("Actual"). It becomes clear how the OML model approaches the ground truth over time and learns the underlying relationship

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`

```
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)
```

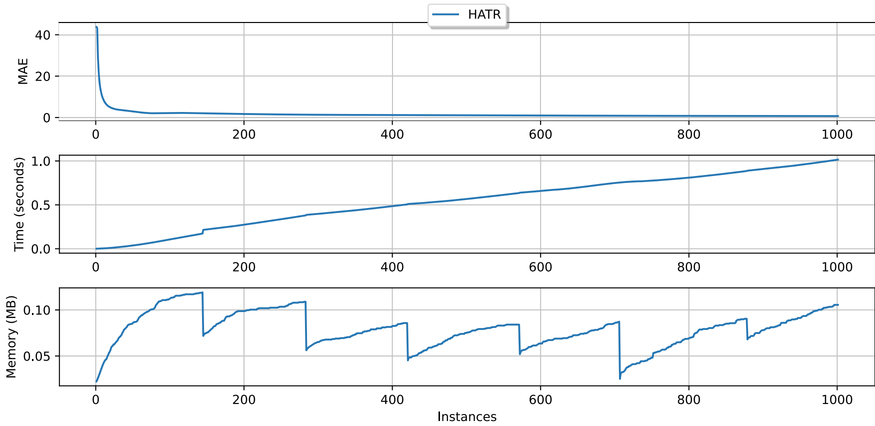

Fig. 5.7 Results of the method `plot_oml_iter_progressive`. The memory management of the HATR model is clearly visible

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.

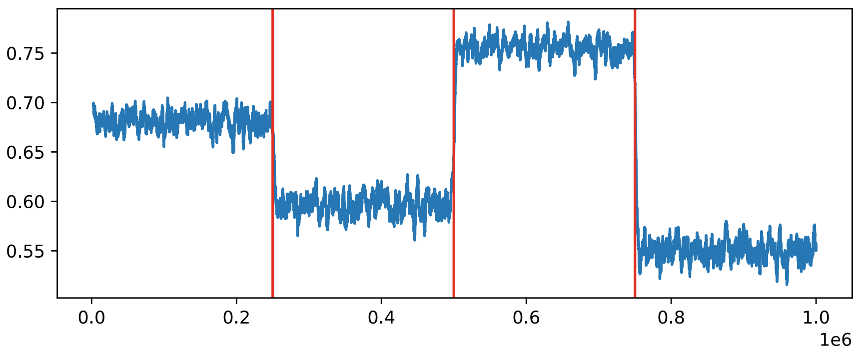

Fig. 5.8 SEA-Data with drift. Concept changes occur after 250, 000 steps

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.