River: machine learning for streaming data in Python

Jacob Montiel∗ JACOB.MONTIEL@WAIKATO.AC.NZ
AI Institute, University of Waikato, Hamilton, New Zealand
LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France

Max Halford∗ MAX.HALFORD@ALAN.EU
Alan, Paris, France

Saulo Martiello Mastelini MASTELINI@USP.BR
Institute of Mathematics and Computer Sciences, University of São Paulo, São Carlos, Brazil

Geoffrey Bolmier GEOFFREY.BOLMIER@VOLVOCARS.COM
Volvo Car Corporation, Göteborg, Sweden

Raphael Sourty RAPHAEL.SOURTY@IRIT.FR
IRIT, Université Paul Sabatier, Toulouse, France
Renault, Paris, France

Robin Vaysse ROBIN.VAYSSE@IRIT.FR
IRIT, Université Paul Sabatier, Toulouse, France
Octogone Lordat, Université Jean-Jaures, Toulouse, France

Adil Zouitine ADIL.ZOUITINE@IRT-SAINTEXUPERY.COM
IRT Saint Exupéry, Toulouse, France

Heitor Murilo Gomes HEITOR.GOMES@WAIKATO.AC.NZ
AI Institute, University of Waikato, Hamilton, New Zealand

Jesse Read JESSE.READ@POLYTECHNIQUE.EDU
LIX, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Talel Abdessalem TALEL.ABDESSALEM@TELECOM-PARIS.FR
LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France

Albert Bifet ABIFET@WAIKATO.AC.NZ
AI Institute, University of Waikato, Hamilton, New Zealand
LTCI, Télécom Paris, Institut Polytechnique de Paris, Palaiseau, France

Editor: Andreas Mueller

Abstract

River is a machine learning library for dynamic data streams and continual learning. It provides multiple state-of-the-art learning methods, data generators/transformers, performance metrics and evaluators for different stream learning problems. It is the result from the merger of two popular packages for stream learning in Python: Creme and scikit-multiflow. River introduces a revamped architecture based on the lessons learnt from the seminal packages. River’s ambition is to be the go-to library for doing machine learning on streaming data. Additionally, this open source package brings under the same um-

∗. Co-first authors.


License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/ Attribution requirements are provided at http://jmlr.org/papers/v22/20-1380.html
brella a large community of practitioners and researchers. The source code is available at https://github.com/online-ml/river.

**Keywords:** stream learning, online learning, data stream, concept drift, supervised learning, unsupervised learning, Python.

1. **Introduction**

In machine learning, the conventional approach is to process data in batches or chunks. Batch learning models assume that all the data is available at once. When a new batch of data is available, these models have to be retrained from scratch. The assumption of data availability is a hard constraint for the application of machine learning in multiple real-world applications where data is continuously generated. Additionally, keeping historical data requires dedicated storage and processing resources, which in some cases might be impractical, e.g. storing the network logs from a data center. A different approach is to treat data as a stream, in other words, as an infinite sequence of items; data is not stored and models continuously learn one data sample at a time (Bifet et al., 2018).

Creme (Halford et al., 2019) and scikit-multiflow (Montiel et al., 2018) are two open-source libraries to perform machine learning in the stream setting. River is the merger of these projects, combining their strengths while leveraging the lessons learnt during their development. More than a simple merge of code, River includes a revamped architecture and expands functionality, e.g. support for mini-batches, processing time improvements, more metrics for classification, regression and clustering, more clustering methods, etc. River supersedes its parent packages and unifies continuous development under a single project. River is mainly written in Python, with some core elements written in Cython (Behnel et al., 2011) for performance. Supported applications are generally as diverse as those found in traditional batch settings, including: classification, regression, clustering, representation learning, multi-label and multi-output learning, forecasting, and anomaly detection.

2. **Architecture**

River’s architecture is the result from the lessons learned during the development of its parent packages Creme and scikit-multiflow. Machine learning models in River are extended classes of specialized mixins that mirror the different type of learning tasks, e.g. classification, regression, clustering, etc. This ensures compatibility across the library and eases the extension/modification of existing models, as well as the creation of new models compatible with the rest of the API.

All predictive models perform two core functions: learn (also referred to as training or fitting) and predict. Learning takes place via the learn_one method (updates the internal state of the model). Depending on the learning task, models provide predictions via the predict_one (classification, regression, and clustering), predict_proba_one (classification), and score_one (anomaly detection) methods. Note that River also contains transformers, which are stateful objects that transform an input via the transform_one method. The suffix *_one indicates that the input is a single data sample.

In the following example, we show a complete machine learning task (learning, prediction and performance measurement) easily implemented in a couple lines of code:
from river import evaluate, metrics, synth, tree

stream = synth.Waveform(seed=42).take(1000)
model = tree.HoeffdingTreeClassifier()
metric = metrics.Accuracy()
evaluate.progressive_val_score(stream, model, metric)

# >>> Accuracy: 77.58%

2.1 Data structure

The de facto container for multidimensional, homogeneous arrays of fixed-size items in Python is the numpy.ndarray (van der Walt et al., 2011). However, in the stream setting, data is available one sample at a time. Accordingly, dictionaries are the default data structure in River as they efficiently store one-dimensional data with O(1) lookup and insertion (Gorelick and Ozsvárd, 2020)

Additional advantages of dictionaries include:
1. Accessing data by name rather than by position is convenient from a user perspective.
2. The ability to store different data types. For instance, the categories of a nominal feature can be encoded as strings alongside numeric features.
3. The flexibility to handle new features that might appear in the stream (feature evolution) and sparse data.

River provides an efficient Cython-based extension of dictionary structures that supports operations commonly applied to unidimensional arrays. These operations include, for instance, the four basic algebraic operations, exponentiation, and the dot product.

2.2 Pipelines

Pipelines are an integral part of River. They are a convenient and elegant way to “chain” a sequence of operations and warrant reproducibility. A pipeline is essentially a list of estimators that are applied in sequence. The only requirement is that the first \( n - 1 \) steps are transformers. The last step can be a regressor, a classifier, a clusterer, a transformer, etc. For example, some models such as logistic regression are sensitive to the scale of the data. A best practice is to scale the data before feeding it to a linear model. We can chain the scaler transformer with a logistic regression model via a `|` (pipe) operator as follows:

from river import linear_model, preprocessing

model = (preprocessing.StandardScaler() |
          linear_model.LogisticRegression())

2.3 Instance-incremental and batch-incremental

Instance-incremental methods update their internal state one sample at a time. Another approach is to use mini-batches of data, known as batch-incremental learning. River offers some limited support for batch-incremental learning. Some models have dedicated meth-

---

1. The actual performance of this operations can be affected by the size of the data to store. We assume that samples from a data stream are relatively small.
Table 1: Benchmark accuracy (%) for the Elec2 data set.

<table>
<thead>
<tr>
<th>model</th>
<th>scikit-learn</th>
<th>Creme</th>
<th>scikit-multiflow</th>
<th>River</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNB</td>
<td>73.22</td>
<td>72.87</td>
<td><strong>73.30</strong></td>
<td>72.87</td>
</tr>
<tr>
<td>LR</td>
<td><strong>68.01</strong></td>
<td>67.97</td>
<td>NA</td>
<td>67.97</td>
</tr>
<tr>
<td>HT</td>
<td>NA</td>
<td>74.48</td>
<td><strong>75.82</strong></td>
<td>75.55</td>
</tr>
</tbody>
</table>

Table 2: Benchmark processing time (seconds) for the Elec2 data set.

<table>
<thead>
<tr>
<th>model</th>
<th>scikit-learn</th>
<th>Creme</th>
<th>scikit-multiflow</th>
<th>River</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>learn</td>
<td>predict</td>
<td>learn</td>
<td>predict</td>
</tr>
<tr>
<td>GNB</td>
<td>10.94</td>
<td>±0.26</td>
<td>5.43</td>
<td>±0.10</td>
</tr>
<tr>
<td>LR</td>
<td>8.72</td>
<td>±0.14</td>
<td>3.15</td>
<td>±0.06</td>
</tr>
<tr>
<td>HT</td>
<td>NA</td>
<td>NA</td>
<td>2.66</td>
<td>±0.06</td>
</tr>
</tbody>
</table>

ods to process data in mini-batches, designated by the suffix _many instead of _one, e.g. learn_one() — learn_many(). These methods expect pandas.DataFrame (pandas development team, 2020) as input, a flexible data structure with labeled axes. This in turn allows a uniform interface for instance-incremental and batch-incremental learning.

3. Benchmark

We benchmark the implementation of 3 algorithms\(^2\) available in scikit-learn (Pedregosa et al., 2011), Creme and scikit-multiflow: Gaussian Naive Bayes (GNB), Logistic Regression (LR) (Hastie et al., 2009), and Hoeffding Tree (HT) (Hulten et al., 2001). Table 1 shows similar accuracy between implementations (as expected) for all models. Table 2 shows the processing time (learn and predict). River models perform at least as fast but overall faster than the rest. Tests are performed on the Elec2 data set (Harries and Wales, 1999) which has 45312 samples with 8 numerical features. Reported processing time is the average of running the experiment 7 times on a system with a 2.4 GHz Quad-Core Intel Core i5 processor and 16GB of RAM. Additional benchmarks for other data sets, machine learning tasks and packages are available in the project’s repository.

4. Summary

River is a machine learning package for data streams in Python. It is the merger of Creme and scikit-multiflow and supersedes said packages. The architecture is designed for both flexibility and ease of use, with the goal of facilitating the deployment of stream learning in diverse domains, both in industrial applications and in academic research. One of our next steps is to propose a canonical way to deploy online models in production. This will most likely result in another open source library, which we plan to work on in parallel of River’s development.

\(^2\) These methods are selected for illustrative purposes only; scikit-learn has many other batch learning methods. On the other hand, River has a substantial set of streaming learning methods including those available in Creme and scikit-multiflow.
References


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


