Machine Learning for Streaming Data

IJCAI Tutorial 2024

Heitor Murilo Gomes[1*], Nuwan Gunasekara[2]
Albert Bifet[2,3], Bernhard Pfahringer[2]

* Corresponding author: heitor.gomes@vuw.ac.nz

https://nuwangunasekara.github.io/ijcai2024/

[1] Victoria University of Wellington, New Zealand, [2] University of Waikato, New Zealand,
[3] TELECOM Paris, LCTI, France.

Heitor Murilo Gomes

Senior lecturer at the Victoria University of Wellington (VuW) in New Zealand. Before joining VuW, Heitor was co-director of the AI Institute at the University of Waikato.
PI for a few research projects ranging from applied to fundamental research (i.e. ML for energy distribution, novel SSL approaches for DS, …).

Leads the ***capymoa*** open source library for data stream learning, and provide support for **MOA** (Massive On-line Analysis).

https://heitorgomes.com/

Nuwan Gunasekara

Completed his Ph.D. from University of Waikato in 2023. Currently works as a researcher at the AI Institute at the University of Waikato. Nuwan's research includes the development of new algorithms such as the Streaming Gradient Boosted Trees (SGBT), recurrent concept drifts and the intersection of Stream and Continual learning.

Nuwan is a core developer of ***capymoa*** and also provides support to **MOA**

nuwangunasekara.github.io/

Albert Bifet

Professor of AI and the Director of the AI Institute at University of Waikato, and Professor of Big Data at Data, Intelligence and Graphs (DIG) LTCI, Télécom Paris, IP Paris. Co-chair of the NZ AI Researchers Association. Leads the TAIAO Environmental Data Science project and co-author of the book "Machine Learning from Data Streams" published at MIT Press.

Co-leads the open source project **MOA** Massive On-line Analysis, and provide advice/support for several other projects such as **capymoa**

https://albertbifet.com/

Bernhard Pfahringer

Professor with the Department of Computer Science, and a co-director for the AI Institute, at the University of Waikato in New Zealand. Co-author of the book "Machine Learning from Data Streams" published at MIT Press. His research span a range of data mining and machine learning sub-fields, with a focus on streaming, randomization, and complex data.

Co-leads ***MOA*** and provide advice/support for ***capymoa***

www.cs.waikato.ac.nz/~bernhard/

About this tutorial

- **Our goal:** Introduce attendees to diverse machine-learning tasks for streaming data applications

About this tutorial

- **Our goal:** Introduce attendees to diverse machine-learning tasks for streaming data applications
    - Classification, regression, ensemble learning, prediction intervals, concept drifts, partially and delayed streams, clustering, anomaly detection, …

About this tutorial

- **Our goal:** Introduce attendees to diverse machine-learning tasks for streaming data applications
    - Classification, regression, ensemble learning, prediction intervals, concept drifts, partially and delayed streams, clustering, anomaly detection, …
- **Beyond the introduction**: Enable attendees to apply and extend the concepts introduced using python notebooks and ***capymoa***

Outline

- **Machine Learning for Streaming Data (intro)**
  IJCAI_2024_introduction.ipynb
    - Learning cycle
    - Evaluation procedures
    - Introduction to *capymoa*
- **Concept drifts**
  IJCAI_2024_drifts.ipynb
    - Simulation, Detection & Evaluation
- **Supervised Learning**
  IJCAI_2024_supervised.ipynb
    - Classification
    - Ensemble learning
- **Supervised Learning (cont.)**
    - Regression
    - Prediction Intervals
      IJCAI_2024_prediction_intervals.ipynb
- **Advanced Topics**
  IJCAI_2024_advanced.ipynb
    - Partially and delayed labeled streams
    - Clustering
    - Anomaly detection
    - More *capymoa* functionalities

Notebooks: https://nuwangunasekara.github.io/ijcai2024/ →

Machine Learning for Streaming Data

Stream Learning

What are data streams?

Sequences of items, possibly infinite, each item having a timestamp, and so a temporal order

Stream Learning

What are data streams?

<u>Sequences of items</u>, possibly infinite, each item having a timestamp, and so a <u>temporal order</u>

Machine learning for streaming data (or Stream learning)

Data items arrive one by one, and we would like to **build and <u>maintain</u> models**, such as patterns or predictors, of these items in real time (or near real time)

Stream Learning: Examples

Sensor data (IoT): energy demand prediction, environmental monitoring, traffic flow

Marketing and e-commerce: product recommendation, click stream analysis, sentiment analysis (social networks)

Cybersecurity: malware detection, spam detection, intrusion detection

And many more!*

* Not every problem should be treated as a stream learning problem!

Stream Learning

When should we abstract the data as a continuous stream?

Stream Learning

When should we abstract the data as a continuous stream?

can't store all the data; or

shouldn't store all the data

Stream Learning: can't store

Storing all the data may **exceed the available storage** capacity or cause practical limitations

The **volume or velocity** of incoming data may be too high to store and process in its entirety

Stream Learning: shouldn’t store

Storing all the data may not be desirable due to **privacy concerns**, **compliance requirements**, or **the nature of the problem**

For example, if we are only interested in **real-time analysis** or **immediate decision-making**

Stream Learning

Using a stream abstraction, we can process the data incrementally, **focusing** on the **most recent** or **relevant** data points, and **discard** or **aggregate** the older data as needed

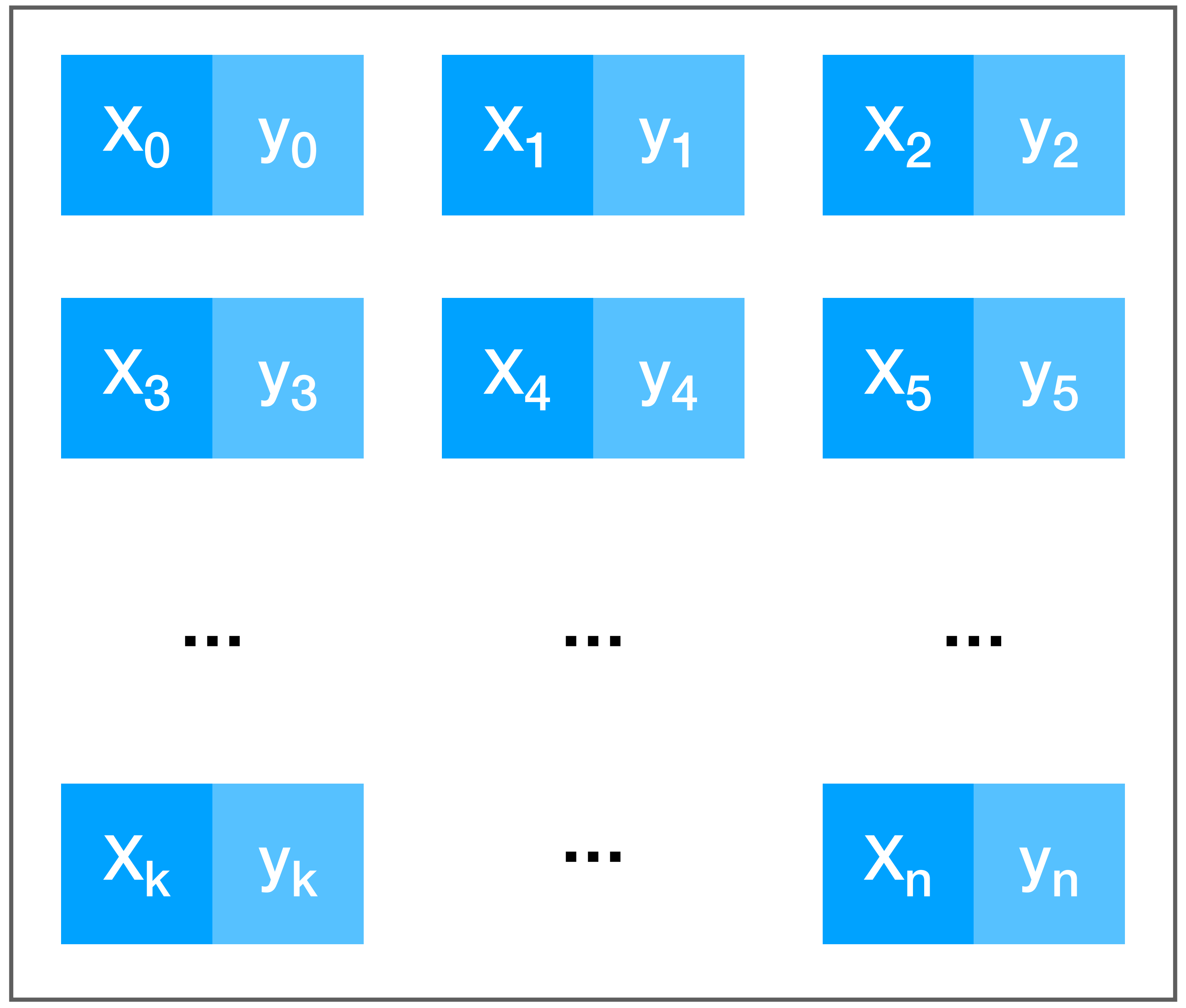
Stream Learning

ML for **Batch (“static”)** data

vs.

ML for **Streaming (“online”)** data

ML for Batch data



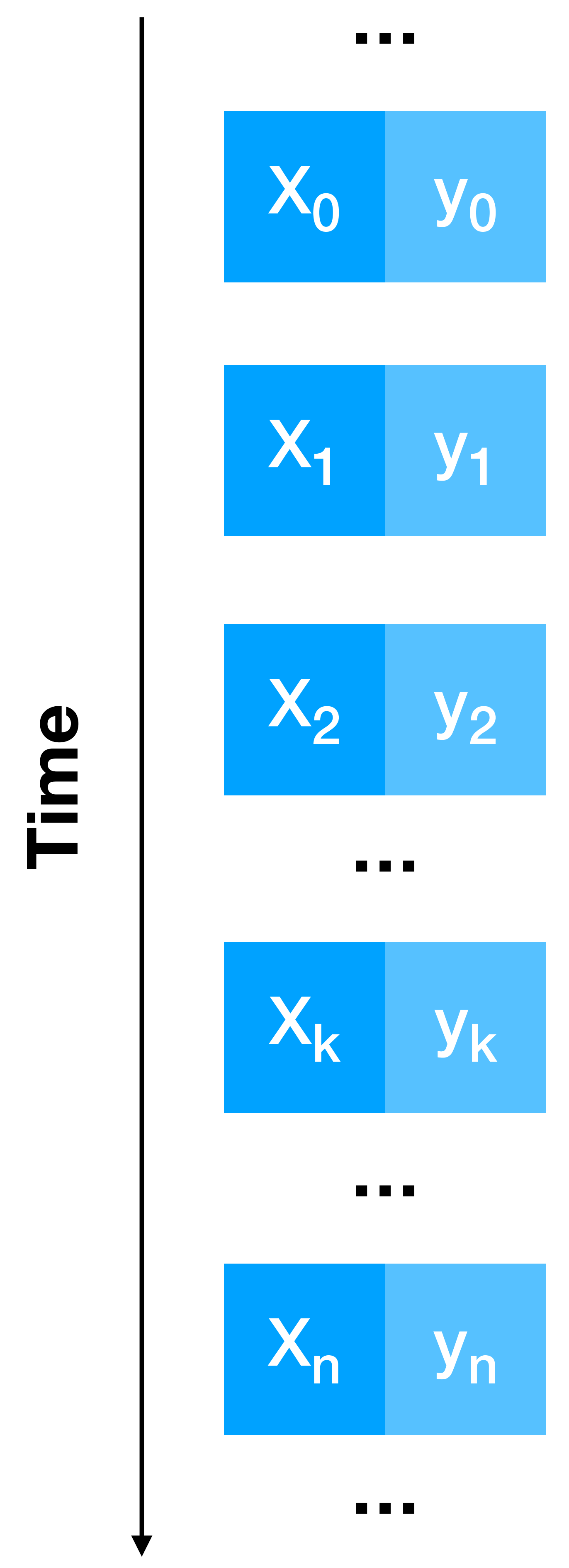
Fixed size dataset

Random access to any instance

Well-defined phases (Train, Validation, Test)

Challenges
noise, missing data, imbalance, high dimensionality, …

ML for Streaming data



Continuous flow of data

Limited time to inspect data points

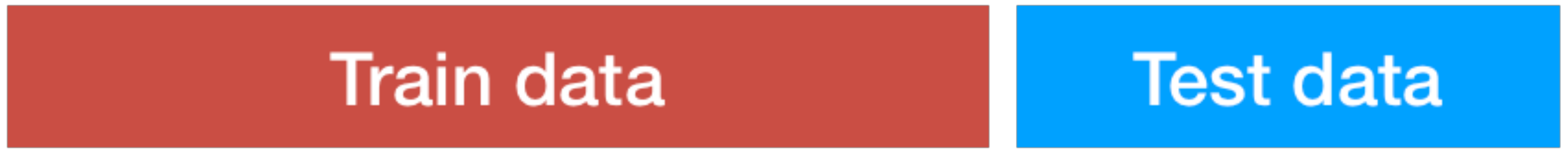
Interleaved phases (Train, Validation, Test)

Challenges

Concept drifts, concept evolution, strict memory/processing requirements, may more and…
inherit all those from batch

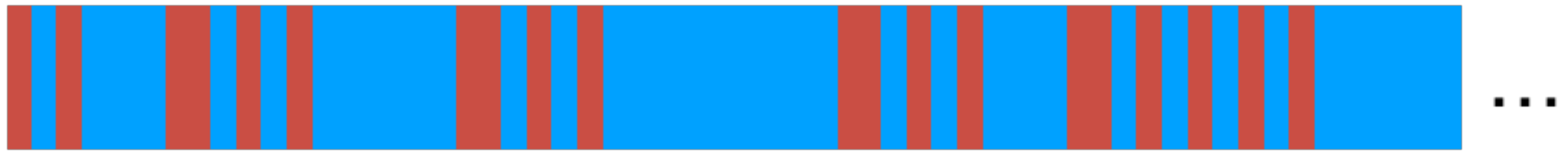
Batch vs. Streaming

Batch data



The output is a **trained model**

Streaming data



The output is a <u>**trainable**</u> **model**

The Learning Cycle

Batch vs. Stream

Batch data

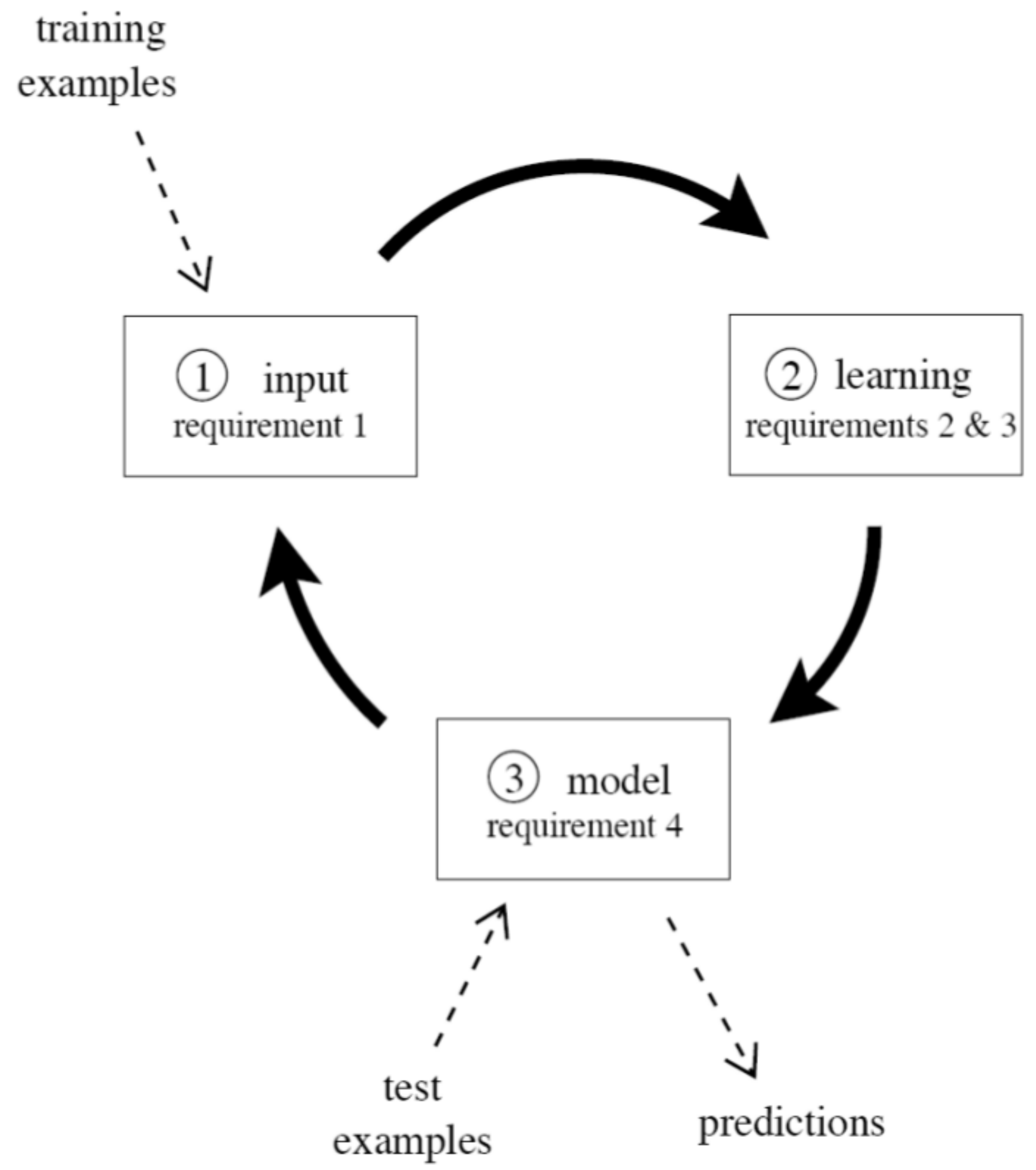
The model is updated through **<u>several passes</u>** over the training data

Streaming data

The model is updated after observing **<u>every</u>** new data point

The Learning Cycle

1. Process an example at a time, and **inspect it only once** (at most)
2. Use a **limited** amount of **memory**
3. Work in a **limited** amount of **time**
4. Be **ready to predict at any point**



Bifet, A., Gavalda, R., Holmes, G., & Pfahringer, B. (2017). *Machine learning for data streams: with practical examples in MOA*. MIT press.

Evaluation

Evaluation overview

Aspects concerning **predictive performance** evaluation:

- **Evaluation metrics**. How errors are considered?

- **Evaluation framework**. How past predictions influence the current metric?

Other measurements (e.g. wall-clock time, CPU time, …)

Evaluation Framework

(Periodic) Holdout

- Interleave test and train subsets
- After testing in one window (or chunk), we observe the average over that window. We use the following window for training.

Evaluation Framework

Interleaved test-then-train (or cumulative)

- Every instance is used for testing first, then for training
- At any point during execution, we observe the average over all instances seen so far

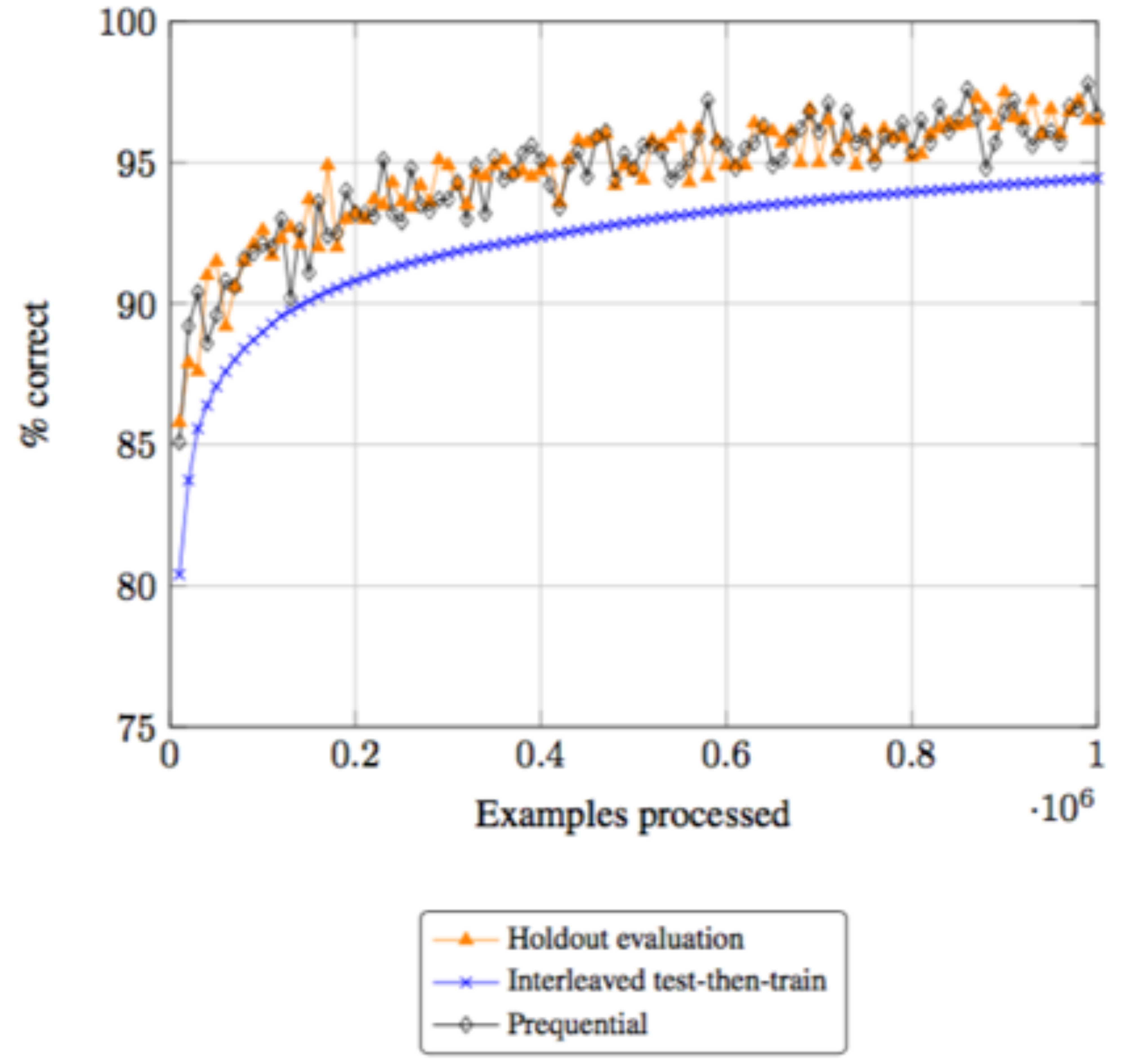
Evaluation Framework

Prequential evaluation

- Similar to interleaved test-then-train, but we observe the metrics over a sliding window of the latest instances (optionally, we can use a fading factor)

Gama, J., Sebastiao, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. Machine learning, 90, 317-346.

Evaluation Framework



<u>Figure from</u>: Bifet, A., Gavalda, R., Holmes, G., & Pfahringer, B. (2017). *Machine learning for data streams: with practical examples in MOA*. MIT press

CapyMOA

Machine learning for data streams

https://capymoa.org/

https://github.com/adaptive-machine-learning/CapyMOA

CapyMOA

A machine learning library for streaming data based on four pillars:

- **Efficiency**
- **Interoperability**
- **Accessibility**
- **Flexibility**

Other frameworks: **MOA** (java)[1], **river** (python)[2] and **scikit-multiflow** (python)[3]

[1] Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., & Seidl, T. (2010). Moa: Massive online analysis, a framework for stream classification and clustering. In *Workshop on applications of pattern analysis* (pp. 44-50). PMLR.

[2] Montiel, J., Halford, M., Mastelini, S.M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H.M., Read, J., Abdessalem, T. and Bifet, A., 2021. River: machine learning for streaming data in python. *Journal of Machine Learning Research*, *22*(110), pp.1-8.

[3] Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, *19*(72), 1-5.

Why another one?

Efficiency

A key aspect of stream learning are **efficient** implementations; they should learn from thousands of instances as quickly as possible (near real-time)

Interoperability

Use algorithms from MOA, scikit-learn and PyTorch through an unified streaming API

Accessibility

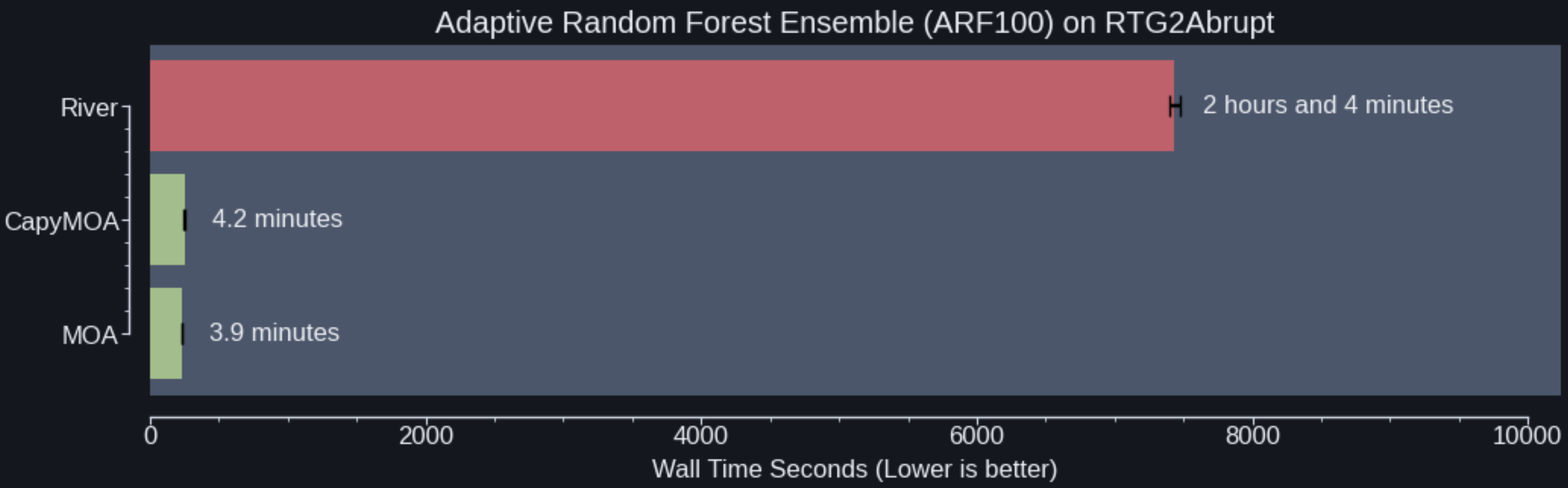
Must be easy to prototype experiments and extend functionality

Flexibility

Advanced APIs for stream learning (concept drift, evaluation, visualisation, …)

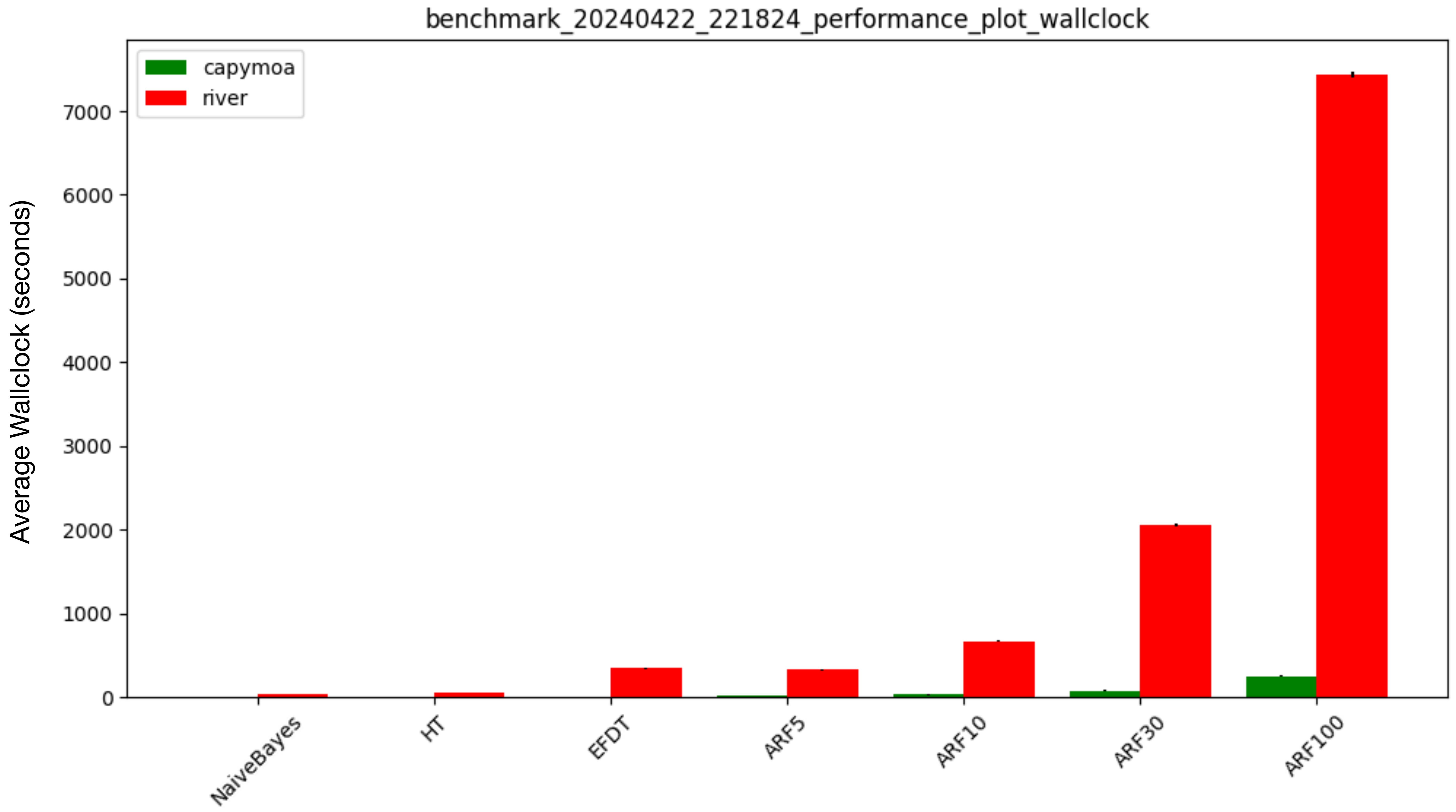
Code in either Python or Java*

Why? Efficiency



Reproducibility: https://github.com/adaptive-machine-learning/CapyMOA/blob/main/notebooks/benchmarking.py

Why? Efficiency



Dataset: RandomTreeGenerator with 2 abrupt drifts, 100k instances, 30 attributes, and 5 classes.
Experiment: 5 repetitions

Algorithms: NaiveBayes, HoeffdingTree (HT), Extremely Fast Decision Tree (EFDT), and AdaptiveRandomForest (the suffix indicates the number of base learners)

Reproducibility: https://github.com/adaptive-machine-learning/CapyMOA/blob/main/notebooks/benchmarking.py

Why? Accessibility

Make it easier to configure and execute complex experiments

MOA's learning curve tends to be steeper as it is implemented in Java

CapyMOA allow users to code in Python, while allowing advanced users to take advantage from MOA objects directly from Python

```
from capymoa.stream.generator import SEA
from capymoa.stream.drift import DriftStream, AbruptDrift,
GradualDrift
from capymoa.classifier import AdaptiveRandomForestClassifier
from capymoa.evaluation import prequential_evaluation
from capymoa.evaluation.visualization import plot_windowed_results

SEA3drifts = DriftStream(stream=[SEA(1),
                                  AbruptDrift(10000),
                                  SEA(2),
                                  GradualDrift(start=20000,
                                               end=25000),
                                  SEA(3),
                                  AbruptDrift(45000),
                                  SEA(1)])
arf =
AdaptiveRandomForestClassifier(schema=SEA3drifts.get_schema(),
                                  ensemble_size=100,
                                  number_of_jobs=4)
results = prequential_evaluation(stream=SEA3drifts,
                                 learner=arf,
                                 window_size=1000,
                                 max_instances=50000)
print(f"Cumulative accuracy = {results['cumulative'].accuracy()}")
print(f"wallclock = {results['wallclock']} seconds")
display(results['windowed'].metrics_per_window())
plot_windowed_results(results, ylabel='Accuracy')
```

Why? Interoperability

Easy access to **MOA**, **PyTorch** and **Scikit-learn** learners

Wraps and expose MOA's API through a modern and simple API

Standardise evaluation and benchmarking without losing **flexibility**

Why? Flexibility

CapyMOA facilitates preparing and executing experiments by incorporating:

- High-level evaluation functions
- Build-in visualisation tools
- Advanced Concept Drift API; and more
- We will show several examples throughout this tutorial

CapyMOA team

- Heitor Murilo Gomes (project leader)[1]
- Anton Lee[1]
- Nuwan Gunasekara[2]
- Yibin Sun[2]
- Guilherme Cassales[2]
- Marco Heyden[3]
- Justin Liu[2]
- Jesse Read[4]
- Maroua Bahri[5]
- Marcus Botacin[6]
- Vitor Cerqueira[7]
- Albert Bifet[2,9]
- Bernhard Pfahringer[2]
- Yun Sing Koh[8]

And many other individual contributors

[1] Victoria University of Wellington, New Zealand
[2] University of Waikato, New Zealand
[3] KIT, Germany
[4] École polytechnique, IP Paris, France
[5] INRIA Paris, France
[6] Texas A&M Engineering, USA
[7] Porto University, Portugal
[8] University of Auckland, New Zealand
[9] Télécom Paris, IP Paris, France

CapyMOA summary

- Allows access to existing and future MOA implementations
- Code in Python or Java, or combine both (e.g. Python using MOA objects)
- Minimal overhead in comparison to executing native MOA
- Integration with PyTorch and scikit-learn
- Streams, learners and evaluation are designed to interoperate with visualisation
- 3 releases (Mar/24, May/24, July/24)
- 20 classifiers, 8 regressors, 11 drift detectors, 3 anomaly detectors, … as of 0.6.0

www.capymoa.org

Practical examples

IJCAI_2024_introduction.ipynb