Navigating Complex Machine Learning Challenges in Streaming Data

ECML Tutorial 2024

Heitor Murilo Gomes[1*], Marco Heyden[2]
Maroua Bahri[3,4]
https://heymarco.github.io/ecml24-streamingchallenges/

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

https://capymoa.org/

[1] Victoria University of Wellington, New Zealand, [2] KIT, Germany, [3] INRIA Paris, France,
[4] Sorbonne Université, 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/

Marco Heyden

Marco is a research scientist and PhD student in the field of machine learning and data mining at Karlsruhe Institute of Technology. He focus on learning from sequential data, specifically the intersection between data stream mining and decision making under uncertainty.

Marco is a core developer of the open source project ***capymoa***, responsible for several algorithms and the pipeline API

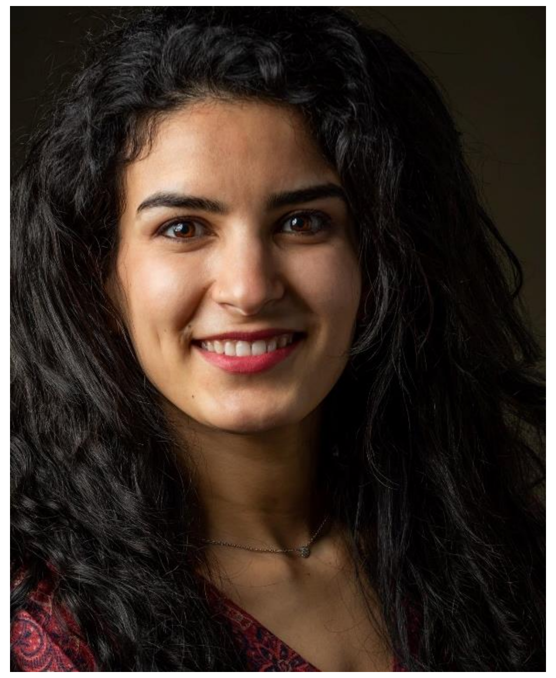
https://heymarco.github.io/

Maroua Bahri

Maroua is an Associate Professor of Computer Science at LIP6, Sorbonne Université. Until August 2024, she was a researcher with the MiMove team at INRIA Paris, where she continues to serve as a scientific collaborator. She earned her PhD in Computer Science from Télécom Paris - Institut Polytechnique de Paris. Her research interests concern machine learning, specifically data stream mining, summarization techniques, and AutoML.

Maroua is a core developer providing advice and support for ***capymoa***

https://sites.google.com/site/bahrimarouaa



Our goals

- Introduce attendees to several machine-learning tasks for streaming data, such as:

  Classification, regression, prediction intervals, concept drifts, partially and delayed streams, clustering, anomaly detection

- Discuss the challenges pertaining ***streaming pipelines*** and ***AutoML***

- Finally, enable attendees to **apply** and **extend** the concepts demonstrated using Python and ***capymoa***

Outline

- **Machine Learning for Streaming Data (intro)**
  01_ECML2024_introduction.ipynb
  - Learning cycle
  - Evaluation
  - *capymoa*
- **Supervised Learning**
  02_ECML2024_supervised.ipynb
  - Classification
  - Regression
  - Prediction Intervals
- **Concept drifts**
  03_ECML2024_drift.ipynb
  - Simulation, Detection & Evaluation
- **Streaming Pipelines**
  04_ECML2024_pipelines.ipynb
  - Challenges and application
- **AutoML**
  05_ECML_2024_automl.ipynb
  - Challenges and application
- **Other Topics**
  06_ECML_2024_other.ipynb
  - Partially and delayed labeled streams
  - Clustering
  - Anomaly detection

Notebooks:
https://heymarco.github.io/ecml24-streamingchallenges/

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?

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

Machine learning for streaming data (or Stream learning)

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

Stream Learning: Examples

<u>Sensor data (IoT):</u> energy demand prediction, environmental monitoring, traffic flow

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

<u>Cybersecurity:</u> 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

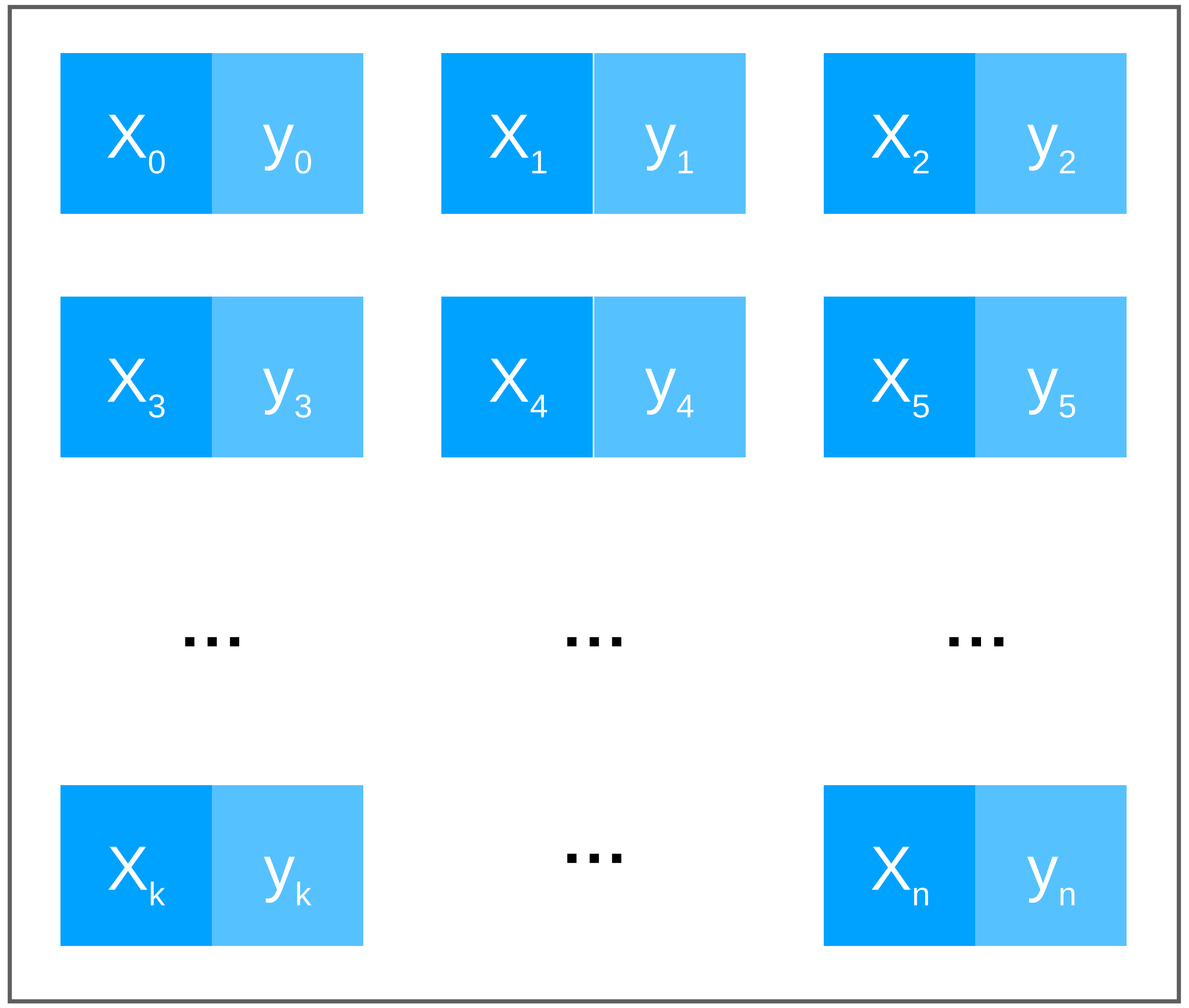
Stream Learning

ML for **Batch (“static”)** data

vs.

ML for **Streaming (“online”)** data

ML for Batch data



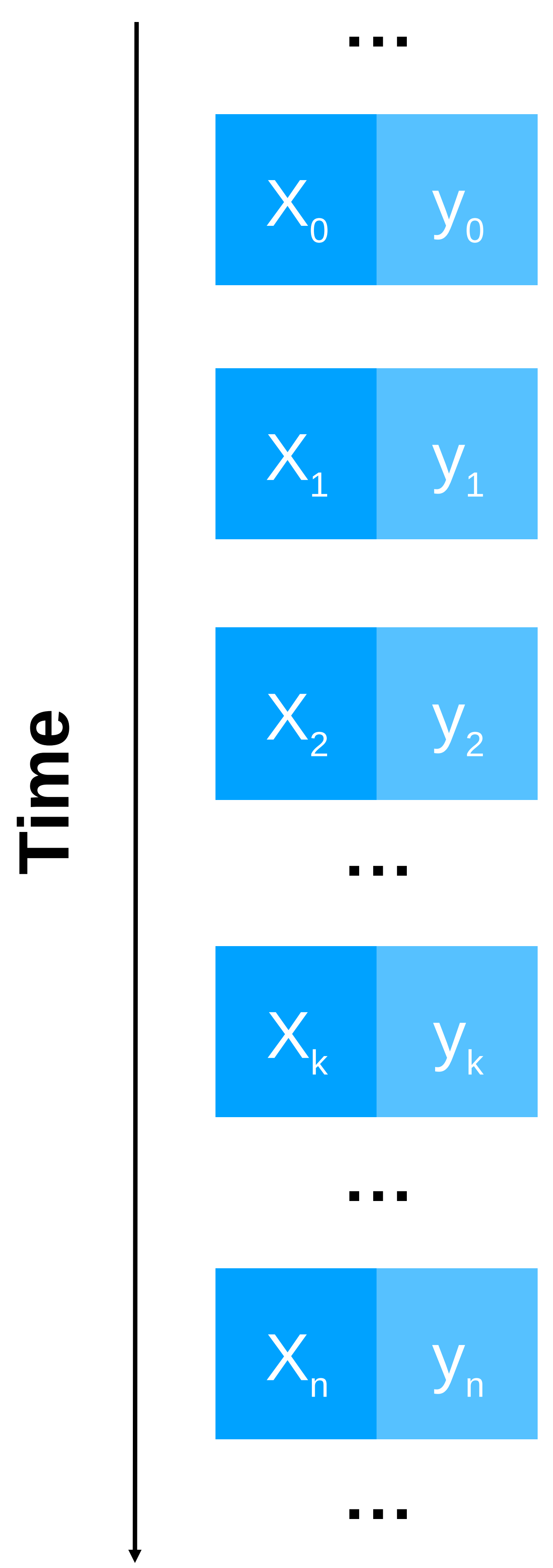
Fixed size dataset

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

<u>Interleaved</u> 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

…

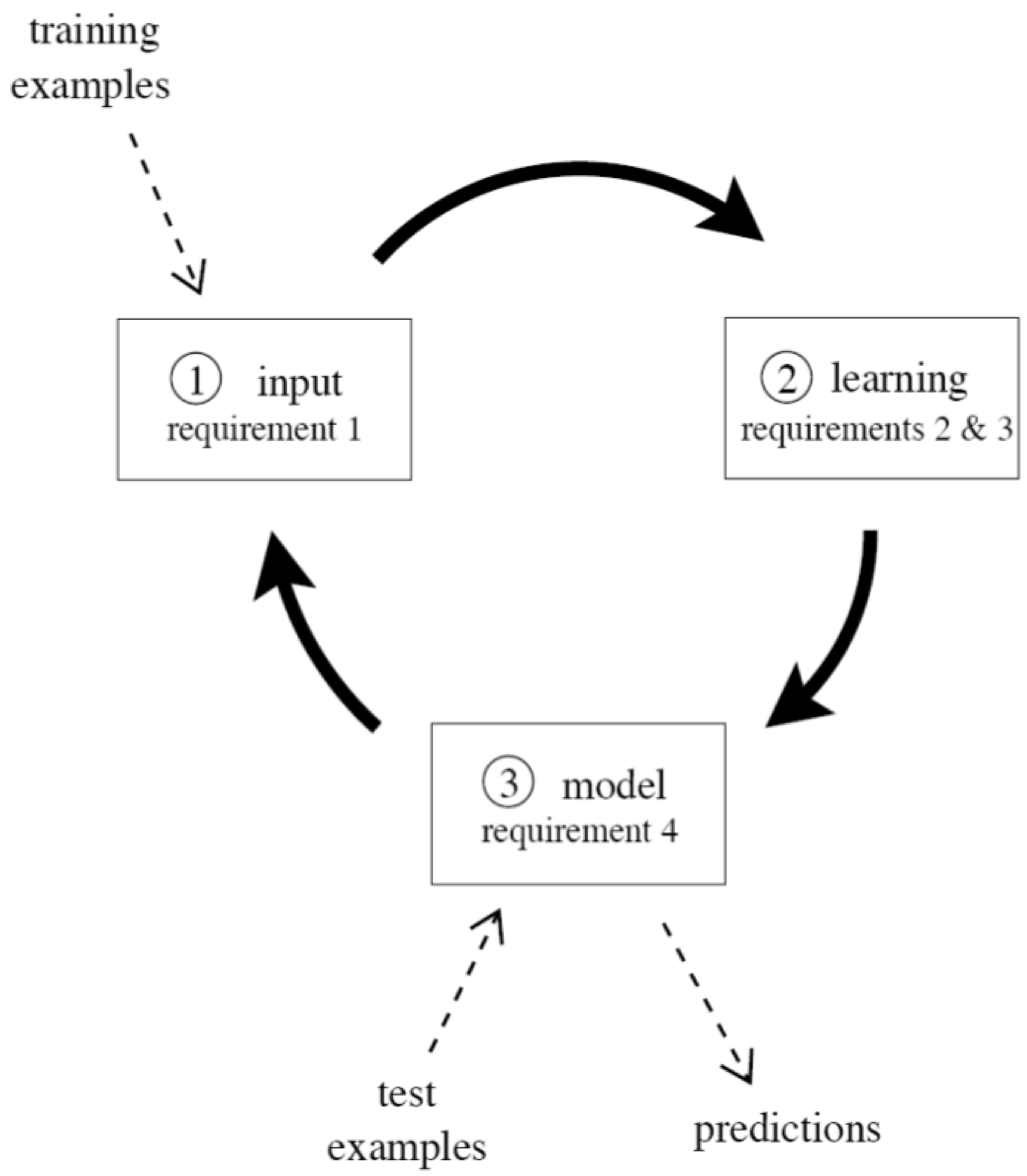
The output is a **<u>trainable</u> model**

The Learning Cycle

The Learning Cycle

Requirements

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**

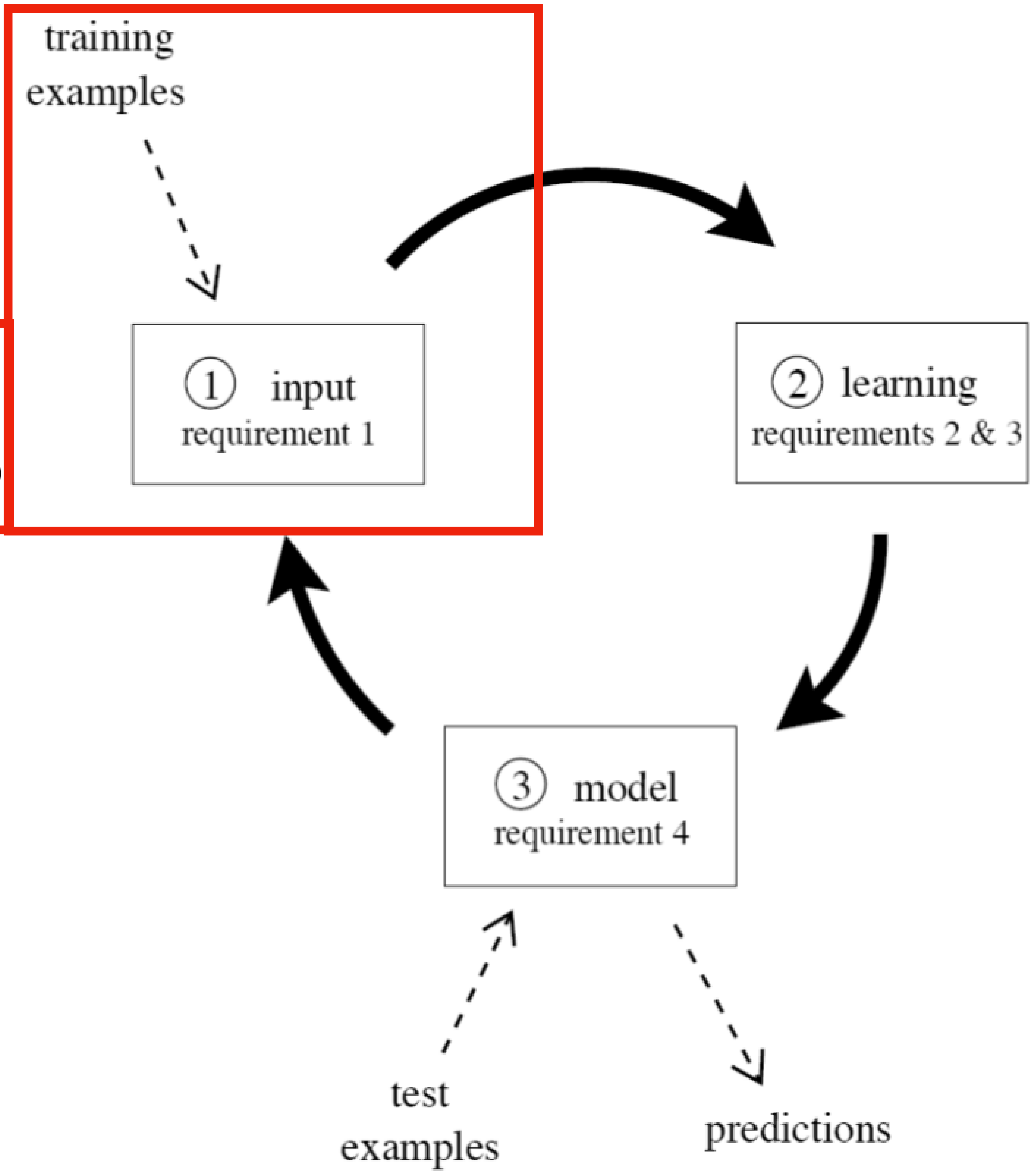


Bifet, A., Gavalda, R., Holmes, G., & Pfahringer, B. (2017). *Machine learning for data streams: with practical examples in MOA*. MIT press.

The Learning Cycle

Requirements

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**

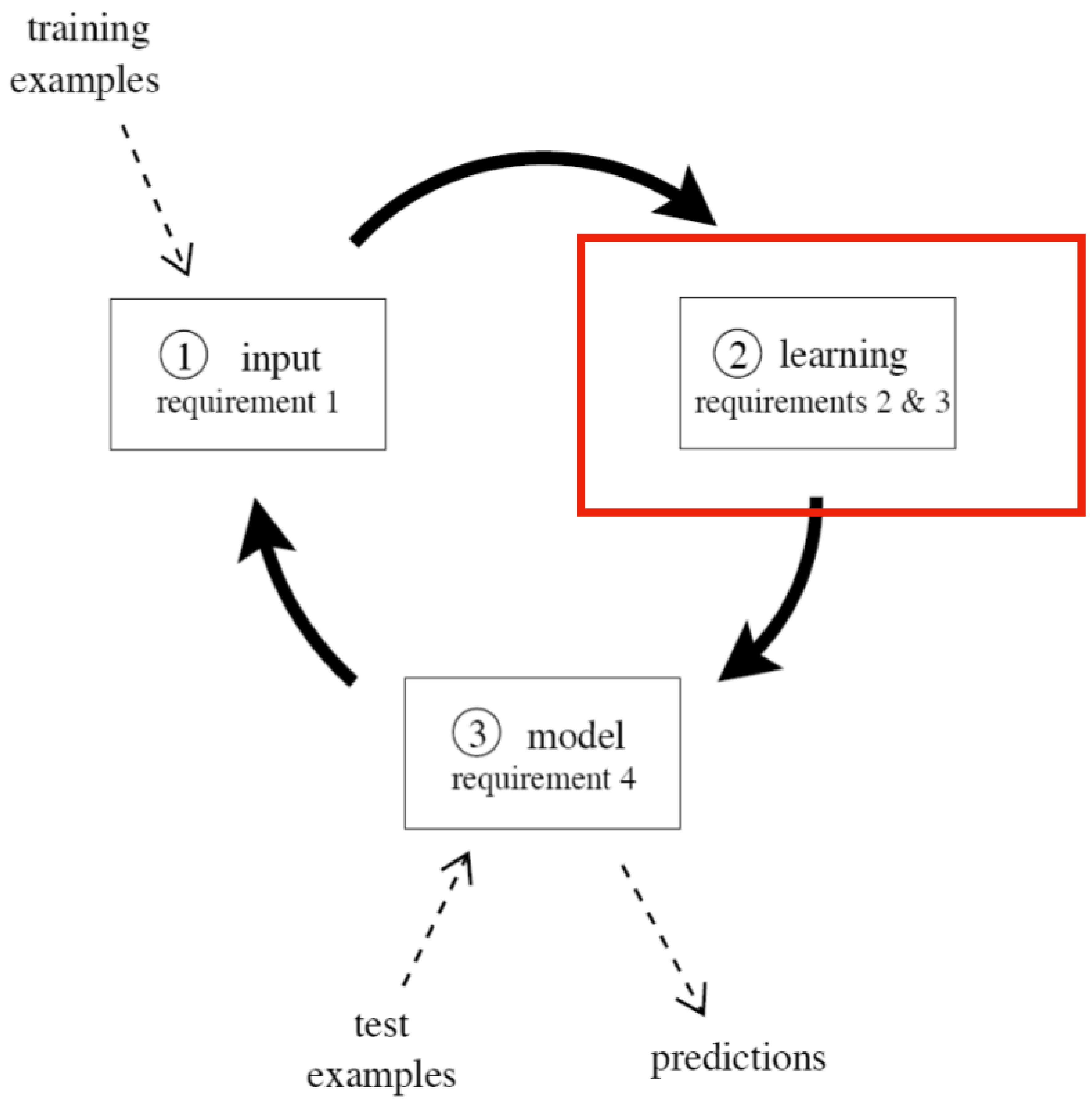


Bifet, A., Gavalda, R., Holmes, G., & Pfahringer, B. (2017). *Machine learning for data streams: with practical examples in MOA*. MIT press.

The Learning Cycle

Requirements

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**

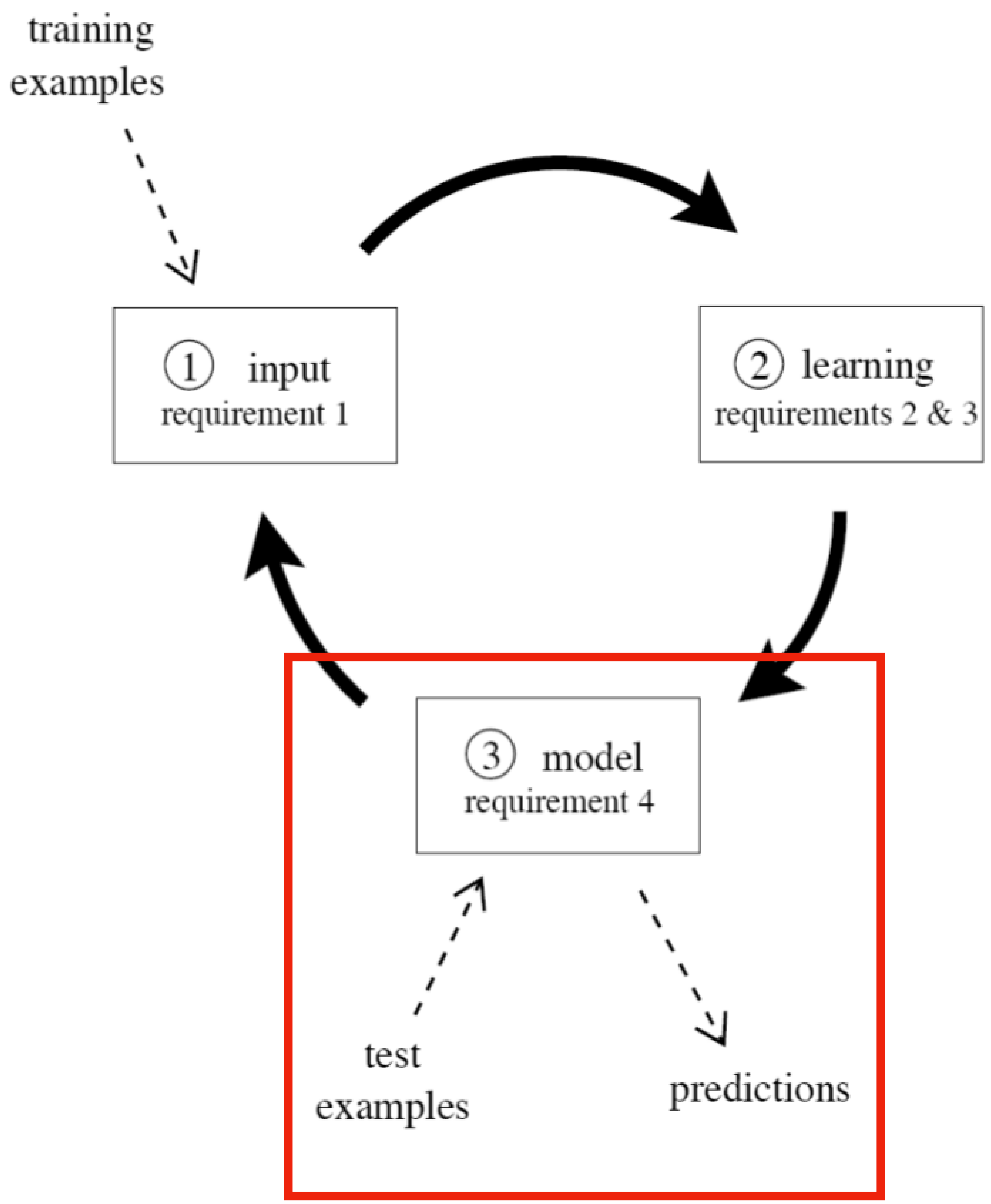


Bifet, A., Gavalda, R., Holmes, G., & Pfahringer, B. (2017). *Machine learning for data streams: with practical examples in MOA*. MIT press.

The Learning Cycle

Requirements

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 **<u>predictive performance</u>** 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

Cumulative (test-then-train): At any point during execution, we observe the average over all instances seen so far

Windowed (prequential): Similar to cumulative, but we observe the metrics over a window of the latest instances

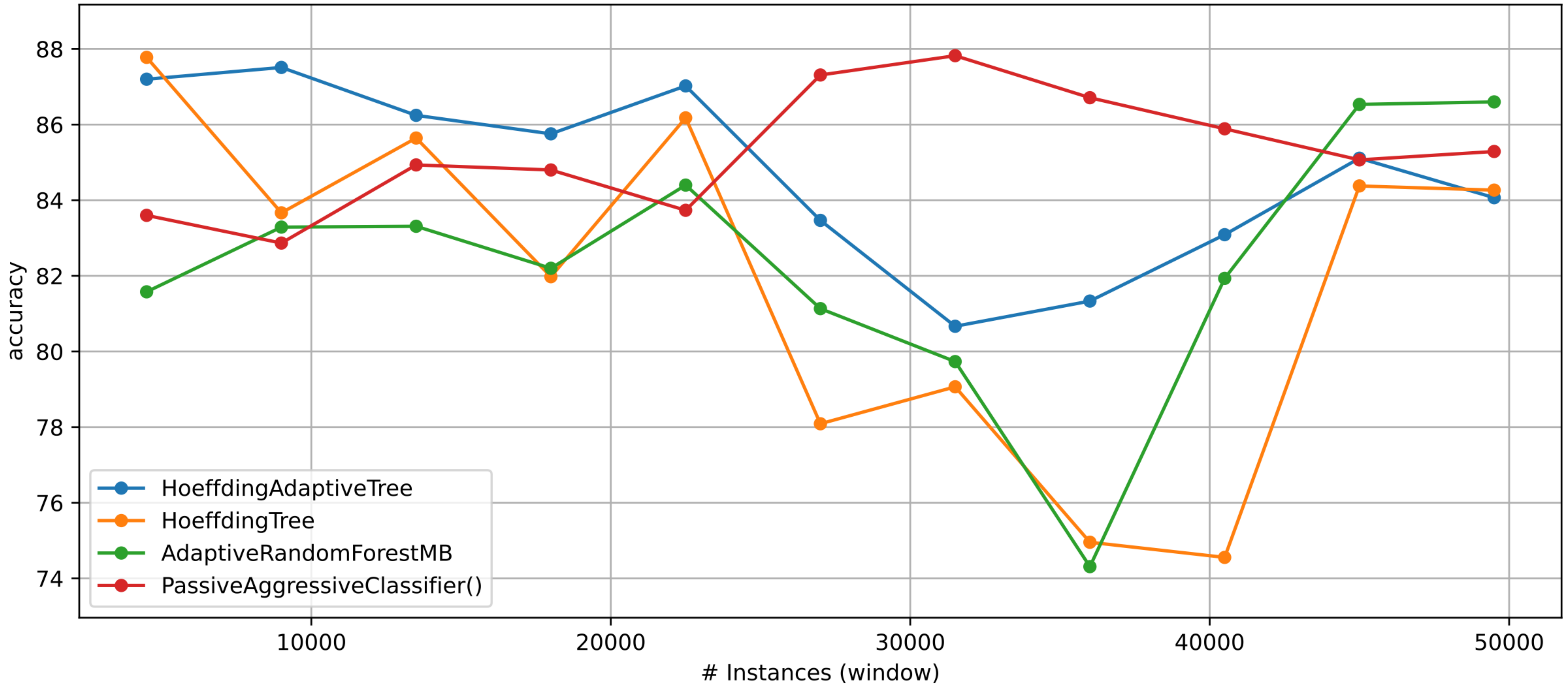
Evaluation Framework (example)

In ***capymoa*** prequential_evaluation(…) will return both results

Cumulative

Algorithm	Accuracy (cumulative)
HoeffdingAdapt.	84.6861
HoeffdingTree	81.6604
AdaptiveRand.	81.9076
PassiveAggr.	85.2445

Windowed



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 three pillars:

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

capymoa is open-source and it was first publicly available on May 03, 2024

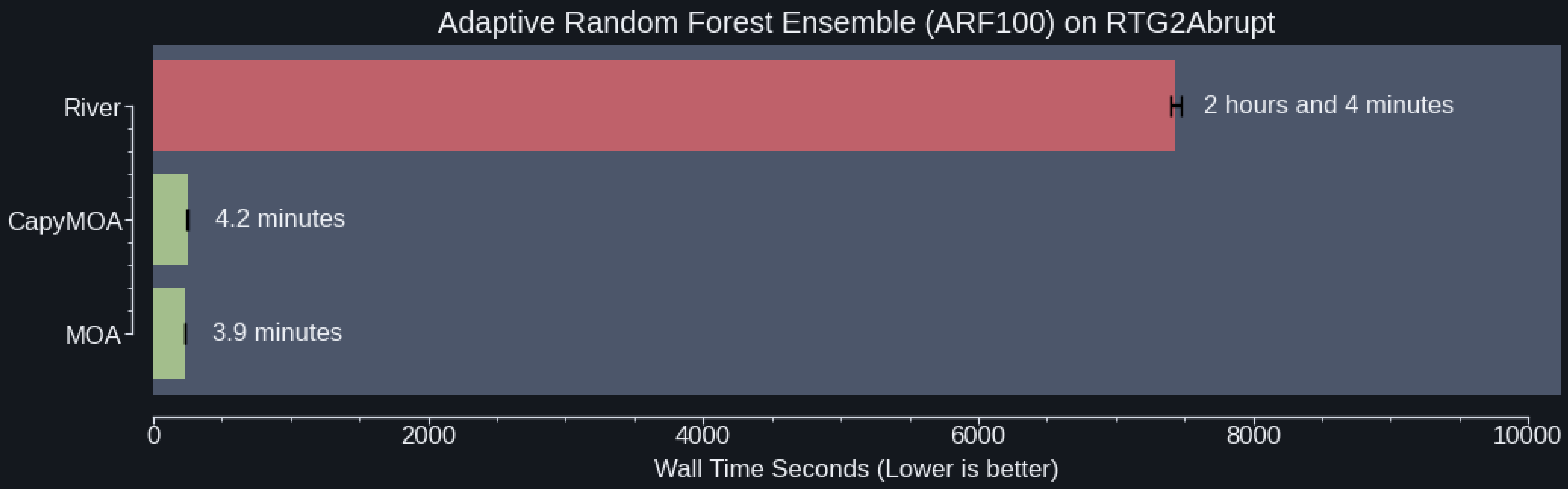
Other frameworks: **MOA** (java)[1], **river** (python)[2] and **scikit-multiflow** (python)[3]

[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? Efficiency



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

Why? Accessibility

Easy to configure and execute complex experiments

Code in Python, but take advantage of MOA (Java) objects

Allows access to existing and future MOA implementations

Integrate stream simulation with evaluation and visualisation

Why? Accessibility

Simulate a data stream with 3 concepts drifts

```
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? Accessibility

Configure an ensemble with 100 learners and 4 jobs (multithreaded)

```
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? Accessibility

Calculate **cumulative** and **windowed** metrics

```
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? Accessibility

```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, metric='accuracy')
```

Plot the **windowed** results

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

- Code in Python or Java, or both
- Integration with PyTorch and scikit-learn
- Streams, learners and evaluation are designed to interoperate with visualisation
- Latest release (0.7.0): August 03, 2024
- 20 classifiers, 8 regressors, 11 drift detectors, 3 anomaly detectors, evaluation, data representation, … as of 0.7.0

www.capymoa.org

Practical examples

01_ECML2024_introduction.ipynb