Stream mining. Lab 1: introduction to stream mining environments

Maciej Grzenda, PhD, DSc
Maciej.Grzenda@pw.edu.pl

Faculty of Mathematics and Information Science
Warsaw University of Technology, Poland

Warszawa, 2025

1. Introduction
2. Running MOA software
3. River in action
4. Modifying MOA code

Introduction

Lab objectives

The objective of this lab is to set up key stream mining frameworks and libraries and use them to run some simple scenarios.

We will start from the overview of software packages providing stream mining methods and more

Next, we will run some calculations with Massive Online Analysis (MOA)

Finally, we will move on to `river`, i.e. Python-based library

Software packages

Probably the largest selection of stream mining methods can still be found in MOA software available at `https://moa.cms.waikato.ac.nz/`.

MOA is partly based on WEKA software including the use of ARFF file format for data files.

MOA is a Java-based open source implementation of many stream mining methods, including Hoeffding tree, and Adaptive Random Forest.

It includes also baseline methods such as streaming kNN and Naive Bayes classifier.

A recent book [1] documents both stream mining methods and the MOA package.

Another option is to use River library `https://riverml.xyz` or CapyMOA `https://capymoa.org`.

Comparison of software packages

Table: Stream mining libraries

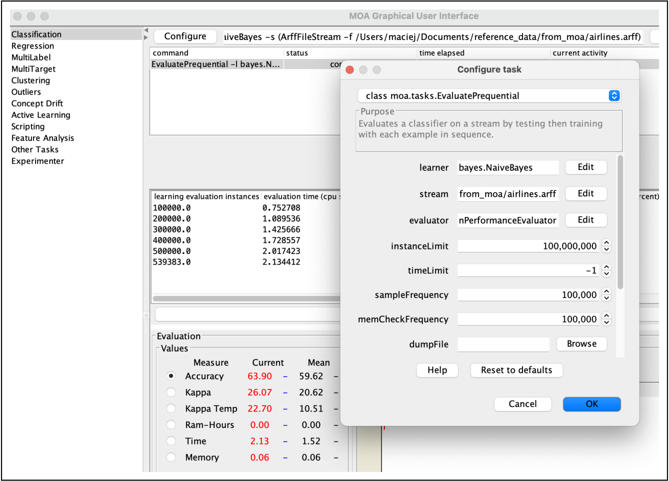
Library	Language	GUI	Comments
MOA `https://moa.cms.waikato.ac.nz/`	Java	YES	An applications rather than explicit library. Still, frequently used for introducing new methods for the first time. Can be called in batch mode.
river `https://riverml.xyz`	Python	NO	A library to be used for developing custom solutions.
CapyMOA `https://capymoa.org`	Python	NO	Unified interface for MOA, PyTorch (for neural networks) and scikit-learn. Performance gains over river.

Running MOA software

Task 1: running the first task in MOA

1. Download MOA from `https://moa.cms.waikato.ac.nz/downloads/` to have software
2. Download `Airlines` and `Forest Covertype` from `https://moa.cms.waikato.ac.nz/datasets/` to have data to work with
3. Run e.g. `moa-release-2024.07.0/bin/moa.sh` or `moa-release-2024.07.0/bin/moa.bat` depending on your operating system and
   - select `Classification/Configure` to configure `moa.tasks.EvaluatePrequential` task
   - in the task configuration window change stream to moa.streams.ArffFileStream and select airlines.arff as your data
   - confirm all the settings and click on `Run`
   - Investigate accuracy and other outcomes of stream processing

Task 1: running the first task in MOA



```
iiveBayes -s (ArffFileStream -f /Users/maciej/Documents/reference_data/from_moa/airlines.arff)
```

command	status	time elapsed	current activity
EvaluatePrequential -l bayes.N...	cor		

```
learning evaluation instances  evaluation time (cpu
100000.0                       0.752708
200000.0                       1.089536
300000.0                       1.425666
400000.0                       1.728557
500000.0                       2.017423
539383.0                       2.134412
```

class moa.tasks.EvaluatePrequential

Purpose
Evaluates a classifier on a stream by testing then training with each example in sequence.

learner	bayes.NaiveBayes
stream	from_moa/airlines.arff
evaluator	nPerformanceEvaluator
instanceLimit	100,000,000
timeLimit	-1
sampleFrequency	100,000
memCheckFrequency	100,000
dumpFile	

Evaluation
Values

	Measure	Current		Mean	
●	Accuracy	63.90	-	59.62	-
○	Kappa	26.07	-	20.62	-
○	Kappa Temp	22.70	-	10.51	-
○	Ram-Hours	0.00	-	0.00	-
○	Time	2.13	-	1.52	-
○	Memory	0.06	-	0.06	-

Figure: EvaluatePrequential task

ARFF file format

ARFF is the file format originally associated with WEKA (`https://www.cs.waikato.ac.nz/ml/weka/`).

ARFF files include:

- headers defining features
- data block in which each line defines one instance

The files can include both independent features and label feature (for classification and regression tasks). A label feature is optional.

Now supported inter alia by `MOA`, `scikit-multiflow` and `river`.

Some of frequently used data streams are available at `https://moa.cms.waikato.ac.nz/datasets/`

Sample ARFF file

Sample content (adapted from Electricity - normalised data set linked at `https://moa.cms.waikato.ac.nz/datasets/`):

```
@relation myElectricityData

@attribute date numeric
@attribute day {1,2,3,4,5,6,7}
@attribute period numeric
@attribute nswprice numeric
@attribute nswdemand numeric
@attribute vicprice numeric
@attribute vicdemand numeric
@attribute transfer numeric
@attribute class {UP,DOWN}

@data
0,2,0,0.056443,0.439155,0.003467,0.422915,0.414912,UP
0,2,0.021277,0.051699,0.415055,0.003467,0.422915,0.414912,UP
0,2,0.042553,0.051489,0.385004,0.003467,0.422915,0.414912,UP
```

Task 2: extending the first task in MOA

1. extend the `moa.tasks.EvaluatePrequential` task from Task 1 by
   - configuring it to save statistics (`dumpFile`) and predictions (`outputPredictionFile`)
   - make MOA save statistics (such as accuracy) every 1,000 instances i.e. change the setting of the task responsible for it
2. investigate the content of `dumpFile`
3. investigate the content of `outputPredictionFile`
4. is the interpretation of the two columns in `outputPredictionFile` clear?

MOA provides a reference implementation of entire tasks, e.g. test-then-train, evaluation over a window of instances and more. Investigating the source code of MOA is a way to learn about even the most detailed stream mining solutions.

Task 3: Obtaining and building MOA software I

- In order to get the most recent version of the MOA software clone `https://github.com/Waikato/moa` repository
- analyse the code of `moa.tasks.EvaluatePrequential` task:
  - What is the interpretation of the two columns in `outputPredictionFile`?
  - are predictions made for instance x before updating a classifier with (x, y) instance or after it?
- you can use Maven to build the software using Java 11 (tested setting). This can be done with the settings below. Tests do not have to be skipped.
  - on Windows:

```
set JAVA_HOME=D:\Program Files\Java\jdk-11.0.6
set JAVA_TOOL_OPTIONS=-Dfile.encoding=UTF8
mvn -Dhttps.protocols=TLSv1.2 clean install
    -DskipTests=true  -Dmaven.javadoc.skip=true
```

Task 3: Obtaining and building MOA software II

- on MacOS:

```
mvn clean install -DskipTests=true
```

Javadoc problem

If `mvn` fails, which may happen with openjdk, the reason may be the `javadoc` plugin. It is enough to comment out `maven-javadoc-plugin` in `pom.xml` then.

MOA: sample command line execution

```
java -cp D:\tools\moa-release-2019.04.0\lib\moa.jar
    -javaagent:c:\tools\moa-release-2016.04\lib\sizeofag-1
    .0.0.jar moa.DoTask EvaluatePrequential -l (moa.
    classifiers.trees.HoeffdingTree) -s (ArffFileStream -f c:\
    work\streamData.arff) -f 1 -d c:\work\statistics.txt -i
    1000000 -o c:\work\predictions.txt
```

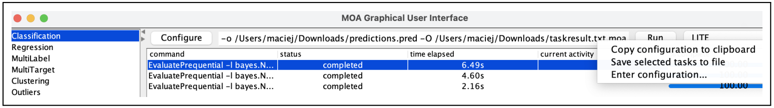
MOA can be (relatively) easily called from command line to run inter alia:

- test-then-train process (EvaluatePrequential)
- of a given stream mining method (here: Hoeffding trees),
- based on input data (frequently an `ARFF` file)
- and save both statistics (such as accuracy) and predictions to files.

Task 4: batch execution

Now execute your task in batch mode:

- copy task configuration from MOA GUI



```
-o /Users/maciej/Downloads/predictions.pred -O /Users/maciej/Downloads/taskresult.txt moa
```

command	status	time elapsed	current activity
EvaluatePrequential -l bayes.N...	completed	6.49s	
EvaluatePrequential -l bayes.N...	completed	4.60s	
EvaluatePrequential -l bayes.N...	completed	2.16s	

Figure: Getting task definition for batch execution (`Right click/Copy configuration to clipboard`)

- investigate moa.sh or moa.bat for Java-related settings and adapt example from previous slide
- develop a batch script for running calculations with two streams and two different methods and saving statistics and predictions to file

River in action

River: sample execution

```python
from river import stream
filePath='/mydata/journeys.arff'

myStream = stream.
    iter_arff(filePath, target='travelAggregation')

from river import tree
from river import metrics

metric = metrics.ClassificationReport()
model = tree.HoeffdingTreeClassifier()
myStream = stream.
    iter_arff(filePath, target='travelAggregation')
for x, y in myStream:
    y_pred = model.predict_one(x)
    model.learn_one(x, y)
    if y_pred is not None:
        metric.update(y, y_pred)
print(metric)
```

Task 5: the first stream mining in River

Now execute a similar task in Python using river library:

- start `Anaconda-Navigator` and next `JupyterLab`
- in `JupyterLab`:
  - start `New/Notebook`
  - customise the script from the previous slide to make it process `airlines.arff` data
  - look for other methods such as Naive Bayes in `river` documentation and try to use them and compare results obtained with different methods for the same data
- Finally, please consider the pros and cons of test-then-train evaluation. Is it easy to implement with any data stream? Is it realistic?

Task 6: mining your own data

Now configure a similar test-then-train task using river or MOA:

- get one of the labelled data sets for a sample classification task you are familiar with
- convert these data to ARFF format. You may find libraries doing it e.g. in `R`
- execute different stream mining methods with it
- investigate the outcomes. Is the performance the same as in the case of batch methods? Should it be the same? How to compare the two approaches?

Modifying MOA code

Configuring MOA in Eclipse

Once MOA repository is cloned into a local folder, we can manage the code in Java IDE. In the case of Eclipse, this means the following steps:

1. use `File\Import...\Maven\Existing Maven projects`
2. after importing the project, `File\Run as\File\`Maven build... can be used to build the software
3. Try to implement a change e.g. in the MOA task or in the stream mining method and build the project

Eclipse vs. other IDEs

You may use other than Eclipse IDEs such as Visual Studio Code.

Building uber JAR files

In order to use MOA GUI after building MOA from source code, *uber* jar has to be built. In order to do so, the following snippet can be included into plugins section of moa\moa\pom.xml file:

```
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.2.4</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
```

This will make mvn download all libraries needed to use GUI and integrate them into moa.jar file

References I

[1] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018. `https://moa.cms.waikato.ac.nz/book/`.