Journal of Machine Learning Research 11 (2010) 1601-1604 Submitted 11/09; Published 4/10

# MOA: Massive Online Analysis

**Albert Bifet** ABIFET@CS.WAIKATO.AC.NZ
**Geoff Holmes** GEOFF@CS.WAIKATO.AC.NZ
**Richard Kirkby** RKIRKBY@CS.WAIKATO.AC.NZ
**Bernhard Pfahringer** BERNHARD@CS.WAIKATO.AC.NZ
*Department of Computer Science*
*University of Waikato*
*Hamilton, New Zealand*

**Editor:** Mikio Braun

## Abstract

**M**assive **O**nline **A**nalysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naïve Bayes classifiers at the leaves. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, and is released under the GNU GPL license.

**Keywords:** data streams, classification, ensemble methods, java, machine learning software

## 1. Introduction

*Green computing* is the study and practice of using computing resources efficiently. A main approach to green computing is based on algorithmic efficiency. In the data stream model, data arrive at high speed, and an algorithm must process them under very strict constraints of space and time.

MOA is an open-source framework for dealing with massive evolving data streams. MOA is related to WEKA, the Waikato Environment for Knowledge Analysis, which is an award-winning open-source workbench containing implementations of a wide range of batch machine learning methods.

A data stream environment has different requirements from the traditional batch learning setting. The most significant are the following:

**Requirement 1** Process an example at a time, and inspect it only once (at most)

**Requirement 2** Use a limited amount of memory

**Requirement 3** Work in a limited amount of time

**Requirement 4** Be ready to predict at any time

Figure 1 illustrates the typical use of a data stream classification algorithm, and how the requirements fit in a repeating cycle:

1. The algorithm is passed the next available example from the stream (Requirement 1).

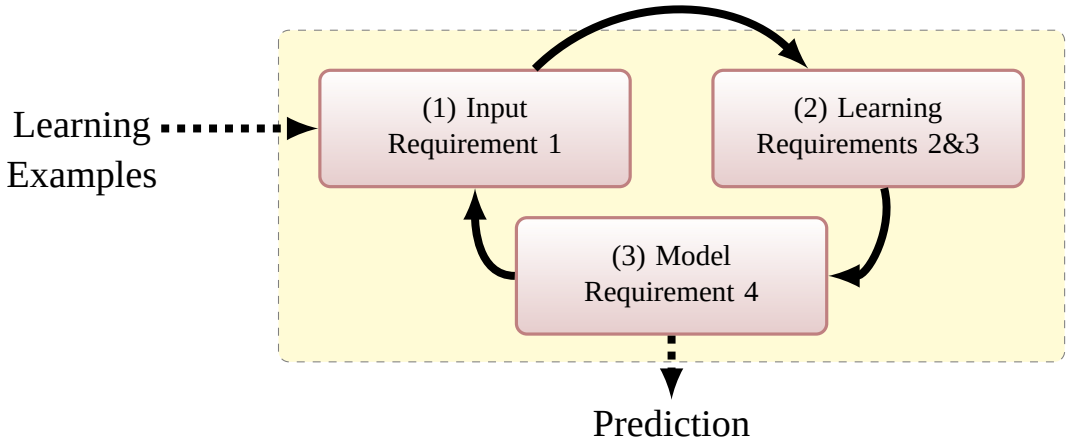


Figure 1: The data stream classification cycle

2. The algorithm processes the example, updating its data structures. It does so without exceeding the memory bounds set on it (requirement 2), and as quickly as possible (Requirement 3).

3. The algorithm is ready to accept the next example. On request it is able to predict the class of unseen examples (Requirement 4).

In traditional batch learning the problem of limited data is overcome by analyzing and averaging multiple models produced with different random arrangements of training and test data. In the stream setting the problem of (effectively) unlimited data poses different challenges. One solution involves taking snapshots at different times during the induction of a model to see how much the model improves.

The evaluation procedure of a learning algorithm determines which examples are used for training the algorithm, and which are used to test the model output by the algorithm.When considering what procedure to use in the data stream setting, one of the unique concerns is how to build a picture of accuracy over time. Two main approaches arise:

- **Holdout**: When traditional batch learning reaches a scale where cross-validation is too time consuming, it is often accepted to instead measure performance on a single holdout set. This is most useful when the division between train and test sets has been pre-defined, so that results from different studies can be directly compared.

- **Interleaved Test-Then-Train or Prequential**: Each individual example can be used to test the model before it is used for training, and from this the accuracy can be incrementally updated. When intentionally performed in this order, the model is always being tested on examples it has not seen. This scheme has the advantage that no holdout set is needed for testing, making maximum use of the available data. It also ensures a smooth plot of accuracy over time, as each individual example will become increasingly less significant to the overall average (Gama et al., 2009).

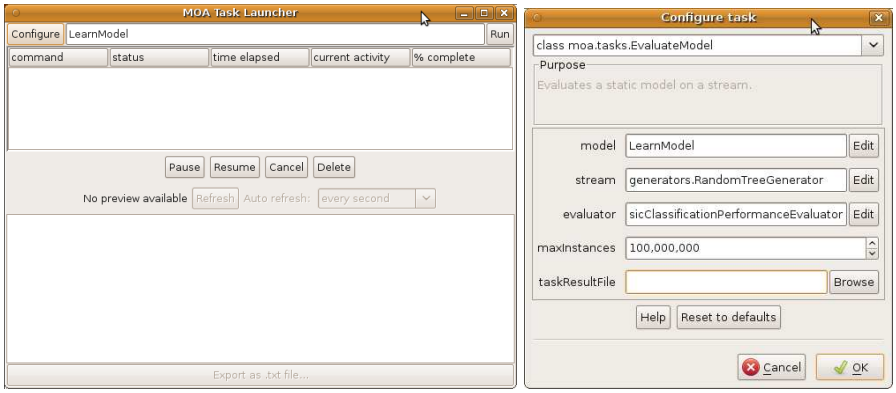


Figure 2: MOA Graphical User Interface

As data stream classification is a relatively new field, such evaluation practices are not nearly as well researched and established as they are in the traditional batch setting. The majority of experimental evaluations use less than one million training examples. In the context of data streams this is disappointing, because to be truly useful at data stream classification the algorithms need to be capable of handling very large (potentially infinite) streams of examples. Demonstrating systems only on small amounts of data does not build a convincing case for capacity to solve more demanding stream applications (Kirkby, 2007).

MOA permits evaluation of data stream classification algorithms on large streams, in the order of tens of millions of examples where possible, and under explicit memory limits. Any less than this does not actually test algorithms in a realistically challenging setting.

## 2. Experimental Framework

MOA is written in Java. The main benefits of Java are portability, where applications can be run on any platform with an appropriate Java virtual machine, and the strong and well-developed support libraries. Use of the language is widespread, and features such as automatic garbage collection help to reduce programmer burden and error.

MOA contains stream generators, classifiers and evaluation methods. Figure 2 shows the MOA graphical user interface. However, a command line interface is also available.

Considering data streams as data generated from pure distributions, MOA models a concept drift event as a weighted combination of two pure distributions that characterizes the target concepts before and after the drift. Within the framework, it is possible to define the probability that instances of the stream belong to the new concept after the drift. It uses the sigmoid function, as an elegant and practical solution (Bifet et al., 2009a,b).

MOA contains the data generators most commonly found in the literature. MOA streams can be built using generators, reading ARFF files, joining several streams, or filtering streams. They allow for the simulation of a potentially infinite sequence of data. The following generators are currently available: Random Tree Generator, SEA Concepts Generator, STAGGER Concepts Generator, Rotating Hyperplane, Random RBF Generator, LED Generator, Waveform Generator, and Function Generator.

MOA contains several classifier methods such as: Naive Bayes, Decision Stump, Hoeffding Tree, Hoeffding Option Tree (Pfahringer et al., 2008), Bagging, Boosting, Bagging using ADWIN, and Bagging using Adaptive-Size Hoeffding Trees (Bifet et al., 2009b).

### 2.1 Website, Tutorials, and Documentation

MOA can be found at: http://moa.cs.waikato.ac.nz/.

The website includes a tutorial, an API reference, a user manual, and a manual about mining data streams. Several examples of how the software can be used are available. For example, a non-trivial example of the EvaluateInterleavedTestThenTrain task creating a comma separated values file, training the HoeffdingTree classifier on the WaveformGenerator data, training and testing on a total of 100 million examples, and testing every one million examples, is encapsulated by the following commandline:

```
java -cp .:moa.jar:weka.jar -javaagent:sizeofag.jar moa.DoTask \
  "EvaluateInterleavedTestThenTrain -l HoeffdingTree \
  -s generators.WaveformGenerator \
  -i 100000000 -f 1000000" > htresult.csv
```

MOA is easy to use and extend. A simple approach to writing a new classifier is to extend moa.classifiers.AbstractClassifier, which will take care of certain details to ease the task. Although the current focus in MOA is on classification, we plan to extend the framework to include data stream clustering, regression, and frequent pattern learning (Bifet, 2010).

## References

Albert Bifet. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. IOS Press, 2010.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavaldà. Improving adaptive bagging methods for evolving data streams. In *First Asian Conference on Machine Learning, ACML 2009*, 2009a.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009b.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in evaluation of stream learning algorithms. In *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009.

Richard Kirkby. *Improving Hoeffding Trees*. PhD thesis, University of Waikato, November 2007.

Bernhard Pfahringer, Geoff Holmes, and Richard Kirkby. Handling numeric attributes in hoeffding trees. In *PAKDD Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 296–307, 2008.