MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering.

Albert Bifet[1], Geoff Holmes[1], Bernhard Pfahringer[1],
Philipp Kranen[2], Hardy Kremer[2], Timm Jansen[2], and Thomas Seidl[2]

[1] Department of Computer Science, University of Waikato, Hamilton, New Zealand
{abifet, geoff, bernhard}@cs.waikato.ac.nz
[2] Data Management and Exploration Group, RWTH Aachen University, Germany
{kranen, kremer, jansen, seidl}@cs.rwth-aachen.de

Abstract. In today's applications, massive, evolving data streams are ubiquitous. **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 is designed to deal with the challenging problems of scaling up the implementation of state of the art algorithms to real world dataset sizes and of making algorithms comparable in benchmark streaming settings. It contains a collection of offline and online algorithms for both classification and clustering as well as tools for evaluation. Researchers benefit from MOA by getting insights into workings and problems of different approaches, practitioners can easily compare several algorithms and apply them to real world data sets and settings. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, and is released under the GNU GPL license. Besides providing algorithms and measures for evaluation and comparison, MOA is easily extensible with new contributions and allows the creation of benchmark scenarios through storing and sharing setting files.

1 Introduction

Nowadays data is generated at an increasing rate from sensor applications, measurements in network monitoring and traffic management, log records or clickstreams in web exploring, manufacturing processes, call detail records, email, blogging, twitter posts and others. In fact, all data generated can be considered as streaming data or as a snapshot of streaming data, since it is obtained from an interval of time.

In data stream scenarios data arrives at high speed strictly constraining processing algorithms in space and time. To adhere to these constraints, specific requirements have to be fulfilled by the stream processing algorithms that are different from traditional batch processing settings. The most significant requirements are the following:

Requirement 1 Process an example at a time, and inspect it at most once
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

Stream learning algorithms are an important type of stream processing algorithms: In a repeated cycle, the learned model is constantly updated to reflect the incoming examples from the stream. They do so without exceeding their memory and time bounds. After processing an incoming example, the algorithms are always able to output a model. Typical learning tasks in stream scenarios are classification, outlier analysis, and clustering.

Since a multitude of algorithms exist for stream learning scenarios, a thorough comparison by experimental evaluation is crucial. In most publications, newly proposed algorithms are only compared to a small subset or even none of the competing solutions, making the assessment of their actual effectiveness tough. Moreover, the majority of experimental evaluations use only small amounts of data. In the context of data streams this is disappointing, because to be truly useful 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 data stream applications [27].

In traditional batch learning scenarios, evaluation frameworks were introduced to cope with the comparison issue. One of these frameworks is the well-known WEKA Data Mining Software that supports adding new algorithms and evaluation measures in a plug-and-play fashion [21, 31, 32]. As data stream learning is a relatively new field, the evaluation practices are not nearly as well researched and established as they are in the traditional batch setting.

For this purpose we introduce **M**assive **O**nline **A**nalysis (MOA) [8], a framework for stream learning evaluation that builds on the work in WEKA. MOA contains state-of-the-art algorithms and measures for both stream classification and stream clustering and permits evaluation of data stream mining algorithms on large streams, in the order of tens of millions of examples where possible, and under explicit memory limits. The main contributions and benefits of the MOA framework are:

- Analysis and comparison both for different approaches (new and state-of-the-art algorithms) and for different (large) streaming setting
- Creation and usage of benchmark settings for comparable and repeatable evaluation of stream mining algorithms
- Open source framework that is easily extensible for data feeds, algorithms and evaluation measures

In the following we first introduce the general architecture of MOA before describing how to use it for classification and clustering on evolving data streams. Section 5 points to additional material including source codes and tutorials and Section 6 concludes the paper.

2 System architecture

A simplified system architecture is illustrated in Figure 1. It shows at the same time the work flow of MOA and its extension points, since all aspects follow the same principle. First a data feed is chosen, then a learning algorithm is configured, i.e. a stream classification or stream clustering algorithm and finally an evaluation method is chosen to analyze the desired scenario. The choice of streams, algorithms and especially evaluation methods differs between the classification and clustering parts and is therefore described separately in the following sections. For both tasks, users can extend the framework in all three aspects to add novel data generators, algorithms or evaluation measures. To run experiments using MOA users can chose between the command line or a graphical user interface.

Generally, MOA permits to define three environments that are simulated using memory limits, since memory limits cannot be ignored and can significantly limit capacity to learn from data streams. Potential practical deployment of data stream learning algorithms has been divided into scenarios of increasing memory utilization, from the restrictive sensor environment, to a typical consumer grade handheld PDA environment, to the least restrictive environment of a dedicated server.

Sensor Network This environment represents the most restrictive case, learning in 100 kilobytes of memory. Because this limit is so restrictive, it is an interesting test case for algorithm efficiency.

Handheld Computer In this case the algorithm is allowed 32 megabytes of memory. This simulates the capacity of lightweight consumer devices designed to be carried around by users and fit into a shirt pocket.

Server This environment simulates either a modern laptop/desktop computer or server dedicated to processing a data stream. The memory limit assigned in this environment is 400 megabytes. Considering that several algorithms have difficulty in fully utilizing this much working space, it seems sufficiently realistic to impose this limit.

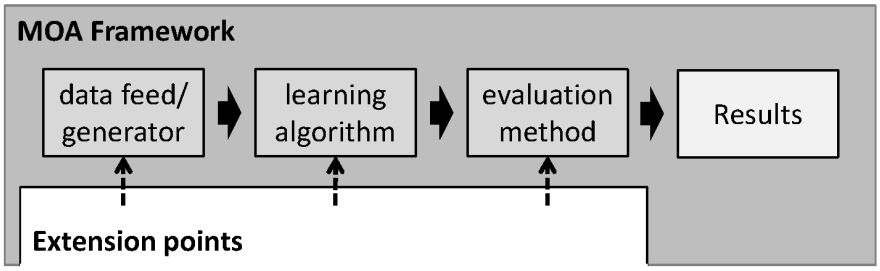

Fig. 1. Architecture, extension points and work flow of the MOA framework.

3 Classification

In this section we detail the features and the usage of stream classfication in MOA.

3.1 Data streams generators for stream classification

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 [9, 10].

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:

SEA Concepts Generator This artificial dataset contains abrupt concept drift, first introduced in [42]. It is generated using three attributes, where only the two first attributes are relevant. All three attributes have values between 0 and 10. The points of the dataset are divided into 4 blocks with different concepts. In each block, the classification is done using $f_1 + f_2 \leq \theta$, where f_1 and f_2 represent the first two attributes and θ is a threshold value. The most frequent values are 9, 8, 7 and 9.5 for the data blocks.

STAGGER Concepts Generator They were introduced by Schlimmer and Granger in [39]. The STAGGER Concepts are boolean functions of three attributes encoding objects: size (small, medium, and large), shape (circle, triangle, and rectangle), and colour (red,blue, and green). A concept description covering either green rectangles or red triangles is represented by (shape= rectangle and colour=green) or (shape=triangle and colour=red).

Rotating Hyperplane It was used as testbed for CVFDT versus VFDT in [25]. A hyperplane in d-dimensional space is the set of points x that satisfy

$$\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i$$

where x_i, is the ith coordinate of x. Examples for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labeled positive, and examples for which $\sum_{i=1}^{d} w_i x_i < w_0$ are labeled negative. Hyperplanes are useful for simulating time-changing concepts, because we can change the orientation and position of the hyperplane in a smooth manner by changing the relative size of the weights. We introduce change to this dataset adding drift to each weight attribute $w_i = w_i + d\sigma$, where σ is the probability that the direction of change is reversed and d is the change applied to every example.

Random RBF Generator This generator was devised to offer an alternate complex concept type that is not straightforward to approximate with a decision tree model. The RBF (Radial Basis Function) generator works as follows: A fixed number of random centroids are generated. Each center has a random position, a single standard deviation, class label and weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weight are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. The length of the displacement is randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. This effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities. Only numeric attributes are generated. Drift is introduced by moving the centroids with constant speed. This speed is initialized by a drift parameter.

LED Generator This data source originates from the CART book [11]. An implementation in C was donated to the UCI [4] machine learning repository by David Aha. The goal is to predict the digit displayed on a seven-segment LED display, where each attribute has a 10% chance of being inverted. It has an optimal Bayes classification rate of 74%. The particular configuration of the generator used for experiments (led) produces 24 binary attributes, 17 of which are irrelevant.

Waveform Generator The goal of the task is to differentiate between three different classes of waveform, each of which is generated from a combination of two or three base waves. The optimal Bayes classification rate is known to be 86%.

Function Generator It was introduced by Agrawal et al. in [3], and was a common source of data for early work on scaling up decision tree learners.The generator produces a stream containing nine attributes, six numeric and three categorical. Although not explicitly stated by the authors, a sensible conclusion is that these attributes describe hypothetical loan applications. There are ten functions defined for generating binary class labels from the attributes. Presumably these determine whether the loan should be approved.

3.2 Classifiers methods

MOA contains several classifier methods such as: Naive Bayes, Decision Stump, Hoeffding Tree, Hoeffding Option Tree, Bagging, Boosting, Bagging using `ADWIN`, and Bagging using Adaptive-Size Hoeffding Trees.

A *Hoeffding tree* [15] is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations (in our case, examples) needed to estimate some statistics within a prescribed precision (in our case, the

goodness of an attribute). More precisely, the Hoeffding bound states that with probability $1-\delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}.$$

A theoretically appealing feature of Hoeffding Trees not shared by other incremental decision tree learners is that it has sound guarantees of performance. Using the Hoeffding bound one can show that its output is asymptotically nearly identical to that of a non-incremental learner using infinitely many examples.

Hoeffding Option Trees [34] are regular Hoeffding trees containing additional option nodes that allow several tests to be applied, leading to multiple Hoeffding trees as separate paths. They consist of a single structure that efficiently represents multiple trees. A particular example can travel down multiple paths of the tree, contributing, in different ways, to different options.

`ADWIN` [7] is a change detector and estimator that solves in a well-specified way the problem of tracking the average of a stream of bits or real-valued numbers. `ADWIN` keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window".

Bagging using `ADWIN` [10] is based on the online bagging method of Oza and Rusell [33] with the addition of the `ADWIN` algorithm as a change detector. When a change is detected, the worst classifier of the ensemble of classifiers is removed and a new classifier is added to the ensemble.

Adaptive-Size Hoeffding Trees (ASHT) [10] are derived from the Hoeffding Tree algorithm with the following differences: it has a value for the maximum number of split nodes, or *size*, and after one node splits, if the number of split nodes of the ASHT tree is higher than the maximum value, then it deletes some nodes to reduce its size. The intuition behind this method is as follows: smaller trees adapt more quickly to changes, and larger trees perform better during periods with little or no change, simply because they were built on more data.

3.3 Evaluation methods for stream classification

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:

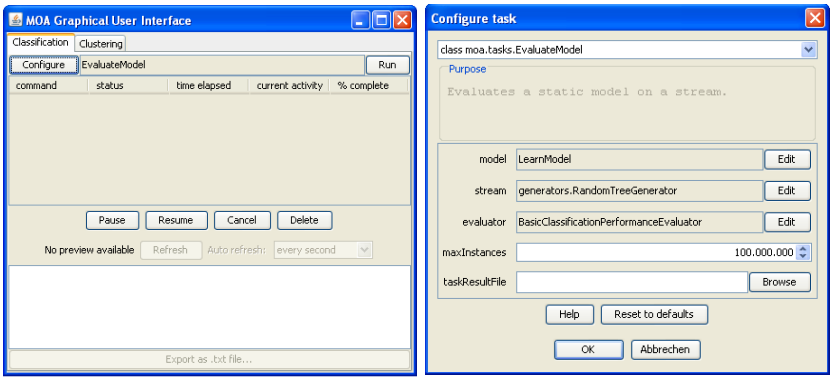

Fig. 2. MOA Graphical User Interface

- **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 [20].

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

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.

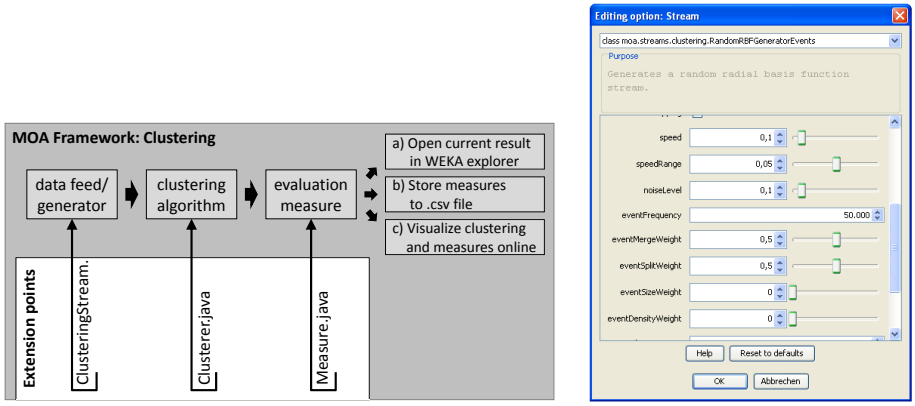

Fig. 3. Left: Extension points and work flow of the MOA stream clustering framework. Right: Option dialog for the RBF data generator (by storing and loading settings benchmark streaming data sets can be shared for repeatability and comparison).

4 Clustering

The stream clustering component of MOA has the following main features:

- data generators for stream clustering on evolving streams (including events like novelty, merge, etc. [41]),
- a set of state-of-the-art stream clustering algorithms,
- evaluation measures for stream clustering,
- visualization tools for analyzing results and comparing different settings.

The left part of figure 3 shows the extension points of MOA stream clustering and thereby illustrates the architecture as well as the usage of the clustering component. First a data feed is chosen and configured, then a stream clustering algorithm and its settings are fixed, then a set of evaluation measures is selected and finally the experiment is run to obtain and analyze the result. We detail these four aspects in the following subsections.

4.1 Data feeds and data generators

For stream clustering we added new data generators that support the simulation of cluster evolution events such as merging or disappearing of clusters [41].

The right part of figure 3 shows a screenshot of the configuration dialog for our RBF data generator with events. Generally the dimensionality, number and size of clusters can be set as well as the drift speed, decay horizon (aging) and noise rate etc. Events constitute changes in the underlying data model such as growing of clusters, merging of clusters or creation of new clusters [41]. Using the event frequency and the individual event weights, one can study the behaviour and performance of different approaches on various settings. Finally, the settings

for the data generators can be stored and loaded, which offers the opportunity of sharing settings and thereby providing benchmark streaming data sets for repeatability and comparison. New data feeds and generators can be added to the MOA framework by implementing the ClusteringStream interface (further description and source code can be found on the MOA website, cf. Section 5).

4.2 Stream clustering algorithms

Currently MOA contains several stream clustering methods including:

- StreamKM++ [1]: It computes a small weighted sample of the data stream and it uses the k-means++ algorithm as a randomized seeding technique to choose the first values for the clusters. To compute the small sample, it employs coreset constructions using a coreset tree for speed up.
- CluStream [2]: It maintains statistical information about the data using micro-clusters. These micro-clusters are temporal extensions of cluster feature vectors. The micro-clusters are stored at snapshots in time following a pyramidal pattern. This pattern allows to recall summary statistics from different time horizons.
- ClusTree [28]: It is a parameter free algorithm automatically adapting to the speed of the stream and it is capable of detecting concept drift, novelty, and outliers in the stream. It uses a compact and self-adaptive index structure for maintaining stream summaries.
- Den-Stream [13]: It uses dense micro-clusters (named core-micro-cluster) to summarize clusters. To maintain and distinguish the potential clusters and outliers, this method presents core-micro-cluster and outlier micro-cluster structures.
- D-Stream [43]: This method maps each input data record into a grid and it computes the grid density. The grids are clustered based on the density. This algorithm adopts a density decaying technique to capture the dynamic changes of a data stream.
- CobWeb [18]. One of the first incremental methods for clustering data. It uses a classification tree. Each node in a classification tree represents a class (concept) and is labeled by a probabilistic concept that summarizes the attribute-value distributions of objects classified under the node.

The set of algorithms is extensible through classes that implement the interface `Clusterer.java`. These are added to the framework via reflections on start up. The three main methods of this interface are

- `void resetLearningImpl()`: a method for initializing a clusterer learner
- `void trainOnInstanceImpl(Instance)`: a method to train a new instance
- `Clustering getClusteringResult()`: a method to obtain the current clustering result for evaluation or visualization

Internal measures	External measures
Gamma [5]	Rand statistic [35]
C Index [24]	Jaccard coefficient [19]
Point-Biserial [30]	Folkes and Mallow Index [19]
Log Likelihood [22]	Hubert Γ statistics [23]
Dunn's Index [17]	Minkowski score [12]
Tau [37]	Purity [44]
Tau A [24]	van Dongen criterion [16]
Tau C [24]	V-measure [38]
Somer's Gamma [24]	Completeness [38]
Ratio of Repetition [24]	Homogeneity [38]
Modified Ratio of Repetition [24]	Variation of information [29]
Adjusted Ratio of Clustering [24]	Mutual information [14]
Fagan's Index [24]	Class-based entropy [40]
Deviation Index [24]	Cluster-based entropy [44]
Z-Score Index [24]	Precision [36]
D Index [24]	Recall [36]
Silhouette coefficient [26]	F-measure [36]

Table 1. Internal and external clustering evaluation measures.

4.3 Stream clustering evaluation measures

For cluster evaluation various measures have been developed and proposed over the last decades. A common classification of these measures is the separation in so called internal measures and external measures. Internal measures only consider the cluster properties, e.g. distances between points within one cluster or between two different clusters. External evaluation measures compare a given clusterings to a ground truth. Table 1 shows a selection of popular measures from the literature. MOA contains an extensible set of both internal and external measures that can be applied to both micro and macro clusterings. Moreover, specialized measures that take the peculiarities of an evolving data stream into account to fairly evaluate the performance of a stream clustering algorithm will be included in our set of measures.

To extend the available collection with additional or novel evaluation measures one has to implement the Measure interface. The main methods are:

- `void evaluateClustering(Clustering clustering, Clustering trueClustering)`: uses the implemented measure to evaluate the given clustering w.r.t. to the provided ground truth.
- `double getLastValue()`: a method that outputs the last result of the evaluation measure.
- `double getMaxValue(), getMinValue(), getMean()`: methods that provide more statistics about the measure's distribution.

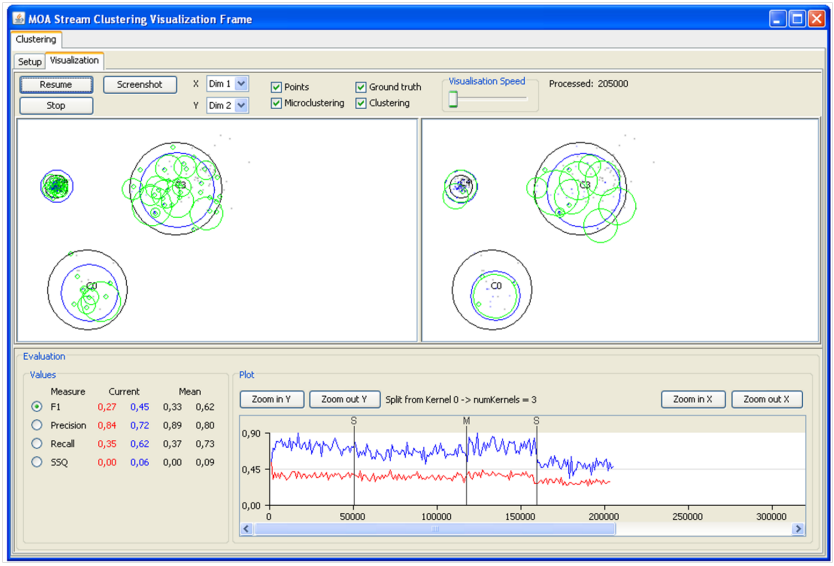

Fig. 4. Visualization tab of the clustering MOA graphical user interface.

4.4 Visualization and analysis

After the evaluation process is started, several options for analyzing the outputs are given: a) the stream can be stopped and the current (micro) clustering result can be passed as a data set to the WEKA explorer for further analysis or mining; b) the evaluation measures, which are taken at configurable time intervals, can be stored as a .csv file to obtain graphs and charts offline using a program of choice; c) last but not least both the clustering results and the corresponding measures can be visualized online within our framework.

Our framework allows the simultaneous configuration and evaluation of two different setups for direct comparison, e.g. of two different algorithms on the same stream or the same algorithm on streams with different noise levels etc.

The visualization component allows to visualize the stream as well as the clustering results, choose dimensions for multi dimensional settings, and compare experiments with different settings in parallel. Figure 4 shows a screen shot of our visualization tab. For this screen shot two different settings of the CluStream algorithm [2] were compared on the same stream setting (including merge/split events every 50000 examples) and four measures were chosen for online evaluation (F1, Precision, Recall, and SSQ). The upper part of the GUI offers options to pause and resume the stream, adjust the visualization speed, choose the dimensions for x and y as well as the components to be displayed (points, micro- and macro clustering and ground truth). The lower part of the GUI displays the measured values for both settings as numbers (left side, including mean values) and the currently selected measure as a plot over the arrived

examples (right, F1 measure in this example). For the given setting one can see a clear drop in the performance after the split event at roughly 160000 examples (event details are shown when choosing the corresponding vertical line in the plot). While this holds for both settings, the left configuration (red, CluStream with 100 micro clusters) is constantly outperformed by the right configuration (blue, CluStream with 20 micro clusters). A video containing an online demo of our system can be found at our website along with more screenshot and explanations. (cf. Section 5).

5 Website, Tutorials, and Documentation

MOA is open source and released under the GNU GPL License. It can be downloaded 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. Additional material regarding the extension of MOA to stream clustering can be found at

`http://dme.rwth-aachen.de/moa-datastream/`

The material includes a live video of the software as well as screenshots and explanations for the most important interfaces that are needed for extending our framework through novel data feeds, algorithms or measures.

6 Conclusions

Our goal is to build an experimental framework for classification and clustering on data streams similar to the WEKA framework. Our stream learning framework provides a set of data generators, algorithms and evaluation measures. Practitioners can benefit from this by comparing several algorithms in real world scenarios and choosing the best fitting solution. For researchers our framework yields insights into advantages and disadvantages of different approaches and allows the the creation of benchmark streaming data sets through stored, shared and repeatable settings for the data feeds. The sources are publicly available and are released under the GNU GPL license. Although the current focus in MOA is on classification and clustering, we plan to extend the framework to include regression, and frequent pattern learning [6].

7 Acknowledgments

This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

1. M. R. Ackermann, C. Lammersen, M. Märtens, C. Raupach, C. Sohler, and K. Swierkot. StreamKM++: A clustering algorithm for data streams. In *SIAM ALENEX*, 2010.
2. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.
3. R. Agrawal, S. P. Ghosh, T. Imielinski, B. R. Iyer, and A. N. Swami. An interval classifier for database mining applications. In *VLDB '92*, pages 560–573, 1992.
4. A. Asuncion and D. Newman. UCI machine learning repository, 2007.
5. F. B. Baker and L. J. Hubert. Measuring the power of hierarchical cluster analysis. *Journal of the American Statistical Association*, 70(349):31–38, 1975.
6. A. Bifet. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. IOS Press, 2010.
7. A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.
8. A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis http://sourceforge.net/projects/moa-datastream/. *Journal of Machine Learning Research (JMLR)*, 2010.
9. A. Bifet, G. Holmes, B. Pfahringer, and R. Gavaldà. Improving adaptive bagging methods for evolving data streams.
10. A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *15th ACM SIGKDD*, 2009.
11. L. Breiman et al. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.
12. M. Brun, C. Sima, J. Hua, J. Lowey, B. Carroll, E. Suh, and E. R. Dougherty. Model-based evaluation of clustering validation measures. *Pattern Recognition*, 40(3):807–824, 2007.
13. F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.
14. T. Cover and J. Thomas. *Elements of Information Theory (2nd Edition)*. Wiley-Interscience, 2006.
15. P. Domingos and G. Hulten. Mining high-speed data streams. In *Knowledge Discovery and Data Mining*, pages 71–80, 2000.
16. S. Dongen. Performance criteria for graph clustering and markov cluster experiments. Technical report, Amsterdam, The Netherlands, The Netherlands, 2000.
17. J. Dunn. Well separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4:95–104, 1974.
18. D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.
19. E. Folkes and C. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78:553–569, 1983.
20. J. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *15th ACM SIGKDD*, 2009.
21. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.
22. J. A. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.
23. L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

24. L. J. Hubert and J. R. Levin. A general statistical framework for assessing categorical clustering in free recall. *Psychological Bulletin*, 83(6):1072–1080, 1976.
25. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD'01*, pages 97–106, San Francisco, CA, 2001. ACM Press.
26. L. Kaufmann and P. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.
27. R. Kirkby. *Improving Hoeffding Trees*. PhD thesis, University of Waikato, November 2007.
28. P. Kranen, I. Assent, C. Baldauf, and T. Seidl. Self-adaptive anytime stream clustering. In *IEEE ICDM*, pages 249–258, 2009.
29. M. Meila. Comparing clusterings: an axiomatic view. In *ICML*, pages 577–584, 2005.
30. G. W. Milligan. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45(3):325–342, 1980.
31. E. Mller, I. Assent, R. Krieger, T. Jansen, and T. Seidl. Morpheus: Interactive exploration of subspace clustering. In *ACM KDD*, pages 1089–1092, 2008.
32. E. Müller, I. Assent, S. Günnemann, T. Jansen, and T. Seidl. OpenSubspace: An open source framework for evaluation and exploration of subspace clustering algorithms in weka. In *OSDM in conjunction with PAKDD*, pages 2–13, 2009.
33. N. Oza and S. Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.
34. B. Pfahringer, G. Holmes, and R. Kirkby. New options for hoeffding trees. In *Australian Conference on Artificial Intelligence*, pages 90–99, 2007.
35. W. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.
36. C. Rijsbergen. *Information Retrieval (2nd Edition)*. Butterworths, London, 1979.
37. F. J. Rohlf. Methods for comparing classifications. *Annual Review of Ecology and Systematics*, 5:101–113, 1974.
38. A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP*, pages 410–420, 2007.
39. J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.
40. M. J. Song and L. Zhang. Comparison of cluster representations from partial second- to full fourth-order cross moments for data stream clustering. In *ICDM*, pages 560–569, 2008.
41. M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. MONIC: modeling and monitoring cluster transitions. In *ACM KDD*, pages 706–711, 2006.
42. W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *KDD '01*, pages 377–382, New York, NY, USA, 2001. ACM Press.
43. L. Tu and Y. Chen. Stream data clustering based on grid density and attraction. *ACM Trans. Knowl. Discov. Data*, 3(3):1–27, 2009.
44. Y. Zhao and G. Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.