

ADAMS

Advanced **D**ata mining **A**nd **M**achine learning **S**ystem

Module: adams-moa



Albert Bifert
Peter Reutemann

November 18, 2015

©2011-2015



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by-sa/3.0/>

Contents

1	Introduction	7
2	Flow	9
2.1	Data sources	9
2.2	Classification	11
2.3	Regression	15
2.4	Clustering	16
2.5	Filtering	17
2.6	Provenance	19
3	Tools	21
	Bibliography	23

List of Figures

1.1	MOA, the (extinct) New Zealand bird.	7
2.1	Flow for generating and displaying artificial data.	10
2.2	The generated data.	10
2.3	Flow for evaluating a classifier on a stream.	11
2.4	The evaluation result.	12
2.5	Flow for serializing a trained classifier.	13
2.6	Flow for classifying data using a pre-built model.	14
2.7	The classification result.	14
2.8	Cluster visualization.	16
2.9	Filtering data streams.	17
2.10	Comparison of streams, filtered and unfiltered.	18
2.11	Flow for displaying provenance information.	19
2.12	The generated provenance trace.	20
3.1	The main MOA interface.	21

Chapter 1

Introduction

MOA (“Massive Online Analysis”, [2]) is a framework for data stream mining. It includes a collection of machine learning algorithms (classification, regression, and clustering) and tools for evaluation. Related to the WEKA project, MOA is also written in Java, while scaling to more demanding problems.

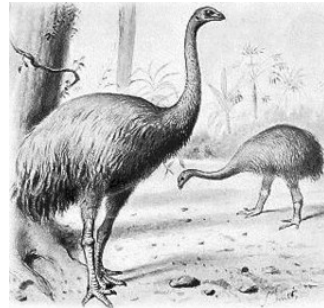


Figure 1.1: MOA, the (extinct) New Zealand bird.

Chapter 2

Flow

If you are familiar with the WEKA actors in ADAMS, then you won't have any problems getting up to speed with using MOA in the flow. The following sections explain the various actors in more detail.

2.1 Data sources

Since MOA uses the WEKA data structures as backend, you can basically use any actor that outputs `weka.core.Instance` tokens as source for the other MOA actors. MOA also comes with a range of stream generators for artificial data (or ARFF-file based ones), which you can make use of the *MOAStream* source. Figure 2.1 shows a flow¹ that generates some artificial data with a stream generator and displays it (see Figure 2.2).

¹adams-moa-datastream.flow

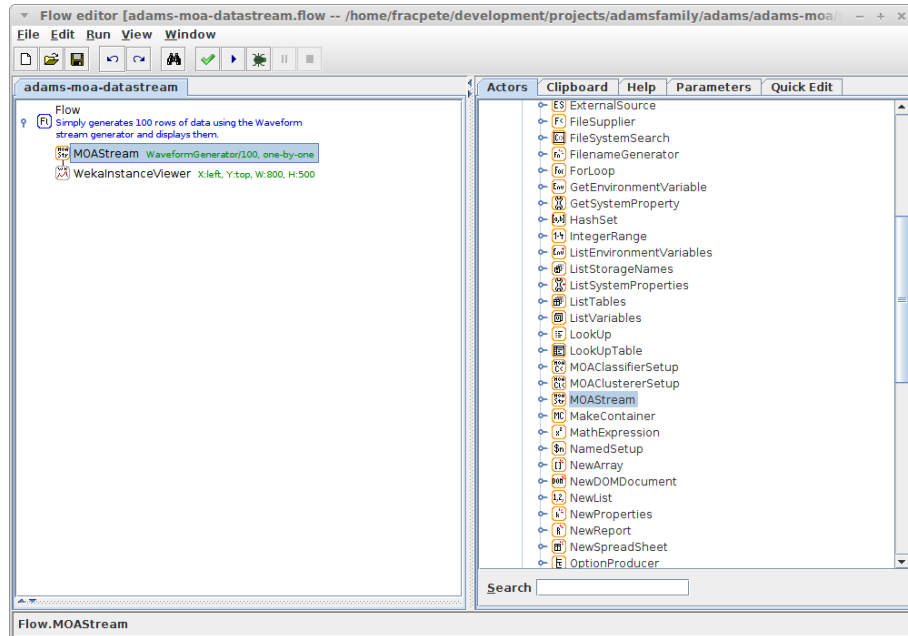


Figure 2.1: Flow for generating and displaying artificial data.

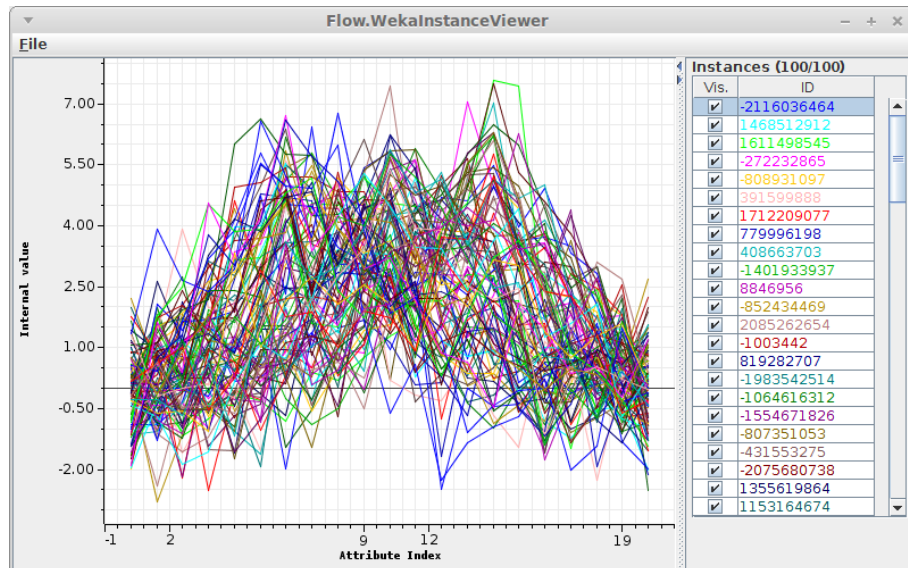


Figure 2.2: The generated data.

Classification in the flow work very similar to ones for WEKA. But instead of performing cross-validation or train/test splits, you use a special stream evaluator which performs an evaluation every X instances that come through. The transformer performing the evaluation is *MOAClassifierEvaluation*. It references a callable classifier of type *MOAClassifierSetup* to evaluate on the data stream and also what type of MOA evaluation you want to perform. Figures 2.3 and 2.4 show a flow² and its associated output (kappa and percentage correct). The classifier is being evaluated every 100 instances of the 10,000 that the stream generator outputs.

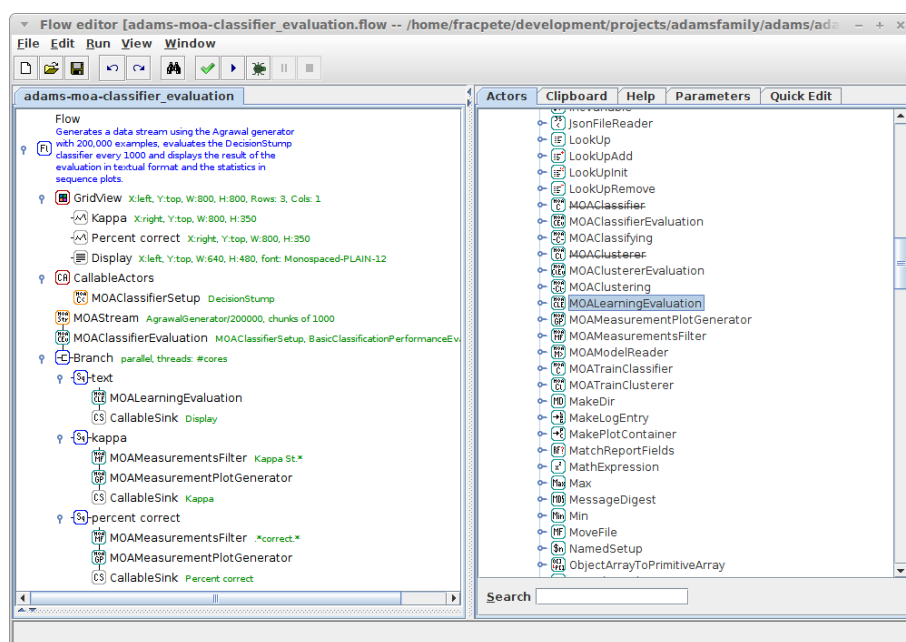


Figure 2.3: Flow for evaluating a classifier on a stream.

²adams-moa-classifier-evaluation.flow

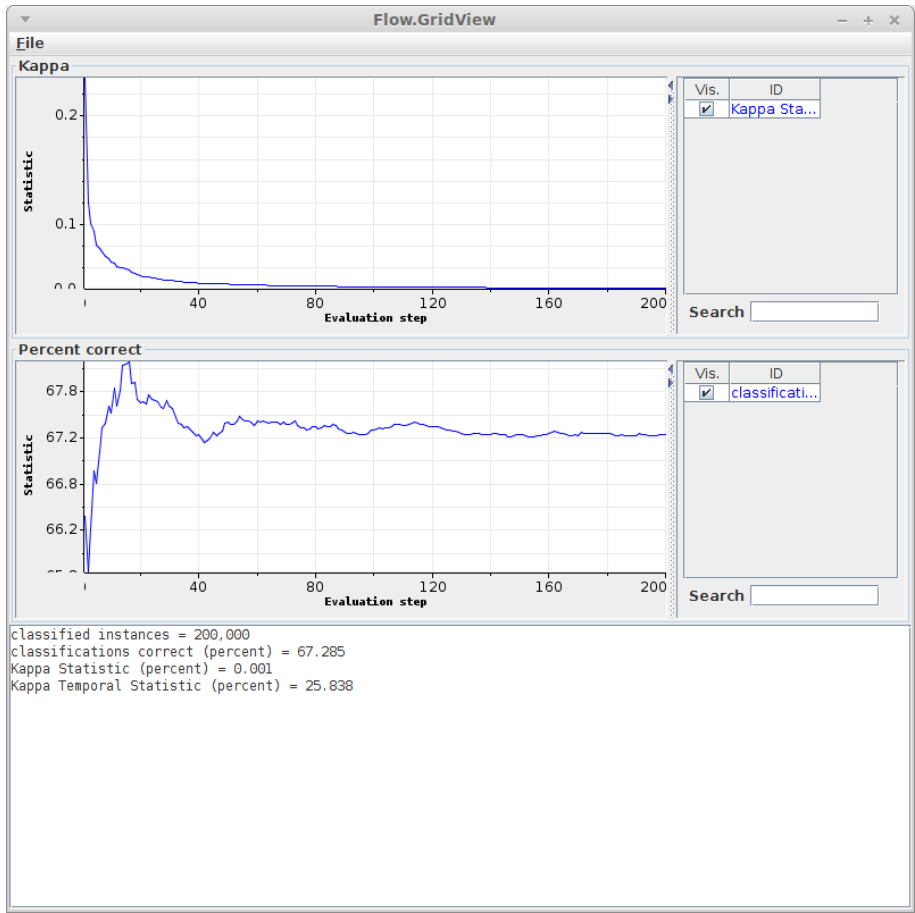


Figure 2.4: The evaluation result.

Just like with WEKA, you can also use a serialized classifier to classify incoming data. First, you need to train and serialize a classifier. How this is done, is shown in the flow³ in Figure 2.5. This flow uses the *MOAModelWriter* sink to save the trained classifier to a file. Then, you can use this serialized model (e.g., loading it with the *MOAModelReader*) in conjunction with the *MOAClassifying* transformer to make predictions on the incoming data. Figures 2.6 and 2.7 show the flow⁴ and the predicted class distributions for the incoming data.

There are some transformers that help you turning the evaluation object that the *MOAClassifierEvaluation* outputs into useful output:

- *MOALearningEvaluation* – generates a string representation of the evaluation object
- *MOAMeasurementsFilter* – picks the measurements from the evaluation that match the regular expression (matching sense can be inverted).
- *MOAMeasurementPlotGenerator* – turns a measurement into a plot container that can be displayed in the *SequencePlotter* sink.

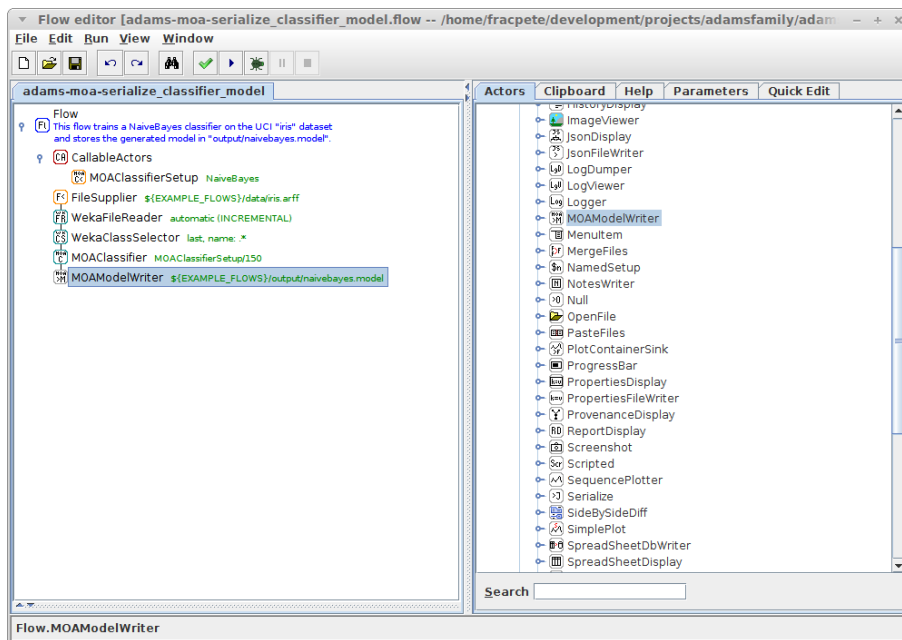


Figure 2.5: Flow for serializing a trained classifier.

³adams-moa-serialize_classifier_model.flow

⁴adams-moa-classifying_with_model.flow

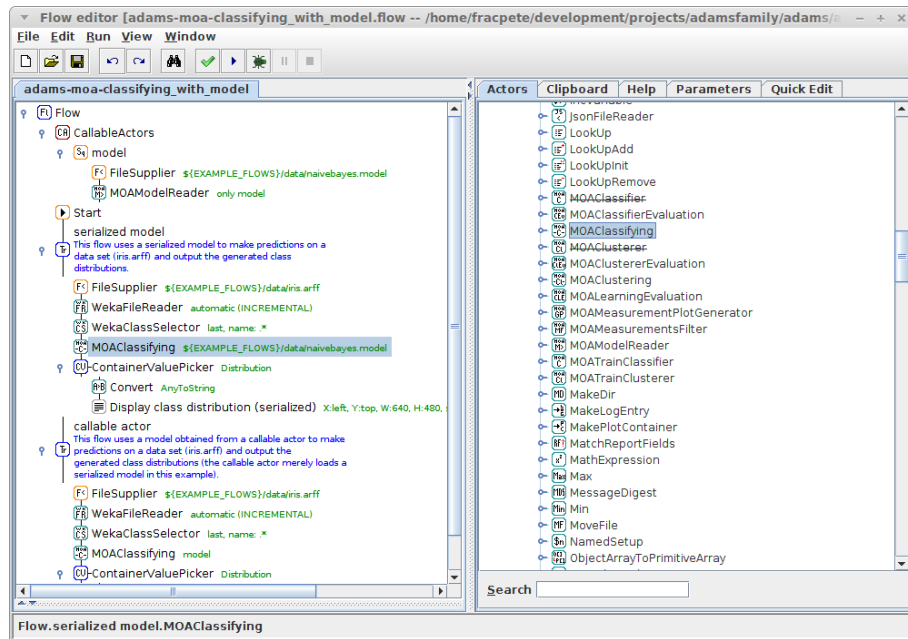


Figure 2.6: Flow for classifying data using a pre-built model.

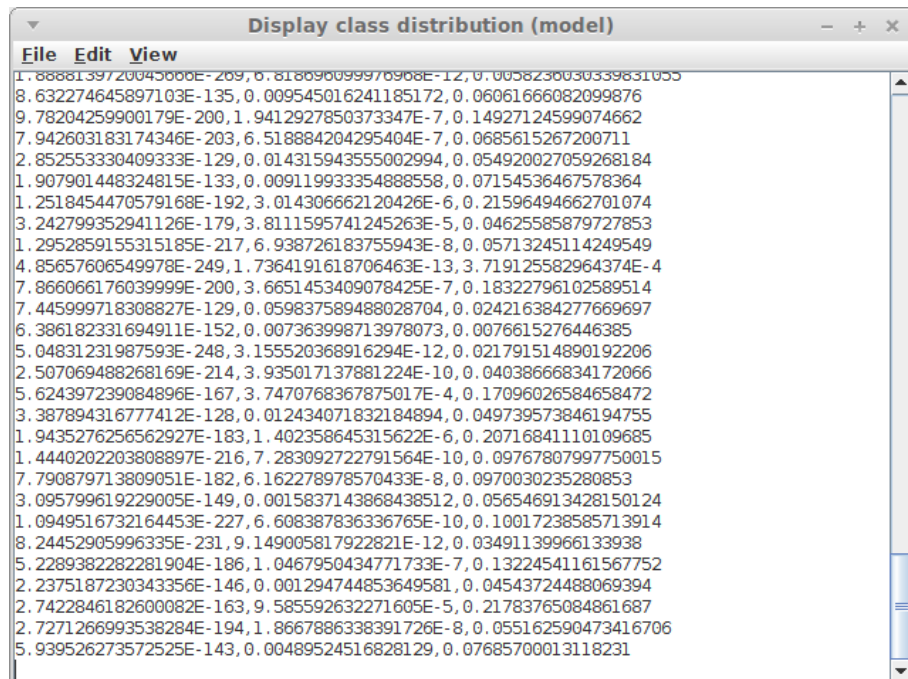


Figure 2.7: The classification result.

2.3 Regression

Regression is very similar to classification with the corresponding actors below:

- *source.MOARegressorSetup* – outputs a regressor object.
- *transformer.MOARegressing* – makes predictions on the incoming data⁵.
- *transformer.MOARegressorEvaluation* – evaluates a regressor on a data stream⁶.
- *transformer.MOATrainRegressor* – builds a regressor model on a data stream.

⁵adams-moa-regressing_with_model.flow

⁶adams-moa-regressor_evaluation.flow

2.4 Clustering

At the moment, only cluster visualization is available in the flow⁷:

- *MOAClustererSetup* – source for defining cluster algorithm setups
- *MOAClusterVisualization* – sink that visualizes clusterings

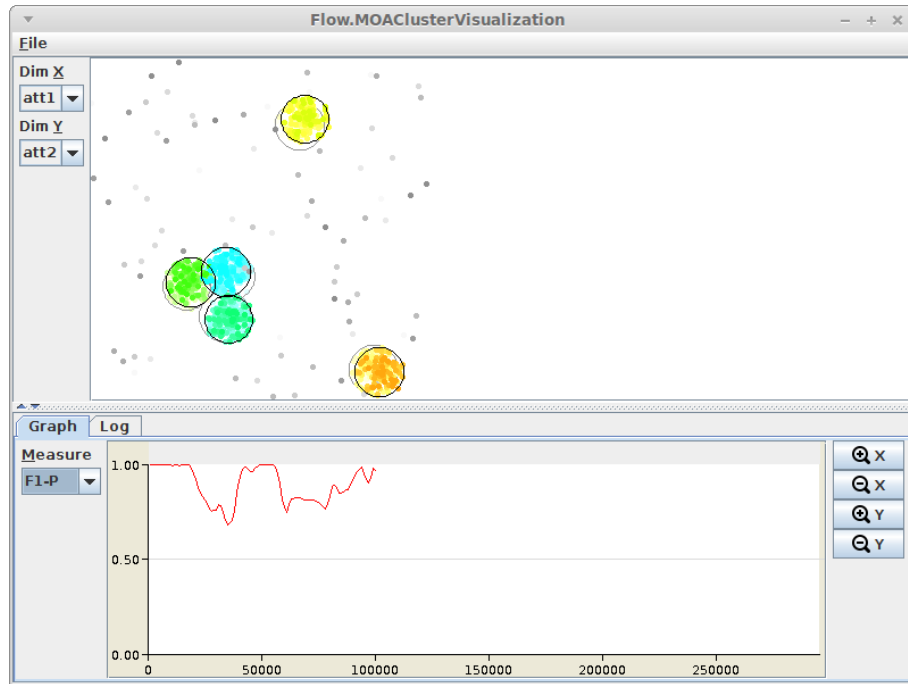


Figure 2.8: Cluster visualization.

⁷adams-moa-cluster_visualization.flow

2.5 Filtering

Even though there is no filtering support in MOA at the time of writing, it is possible using WEKA's stream filters to filter data streams in MOA. You can use the *WekaStreamFilter* transformer to apply one of WEKA's stream filters to the stream of *weka.core.Instance* objects passing through.

In Figure 2.9 you can see a flow⁸ that generates a data stream using the *RandomRBFGenerator* class. It outputs a stream with 40 attributes and 4 class labels. This flow applies the *DownSample* filter (only uses every nth attribute) to the stream and plots the classifier performance, percentage correct and kappa, in a graph (see Figure 2.10). Three plots are generated: evaluation on the full attribute range, down-sampled with using only every 2nd and 4th attribute.

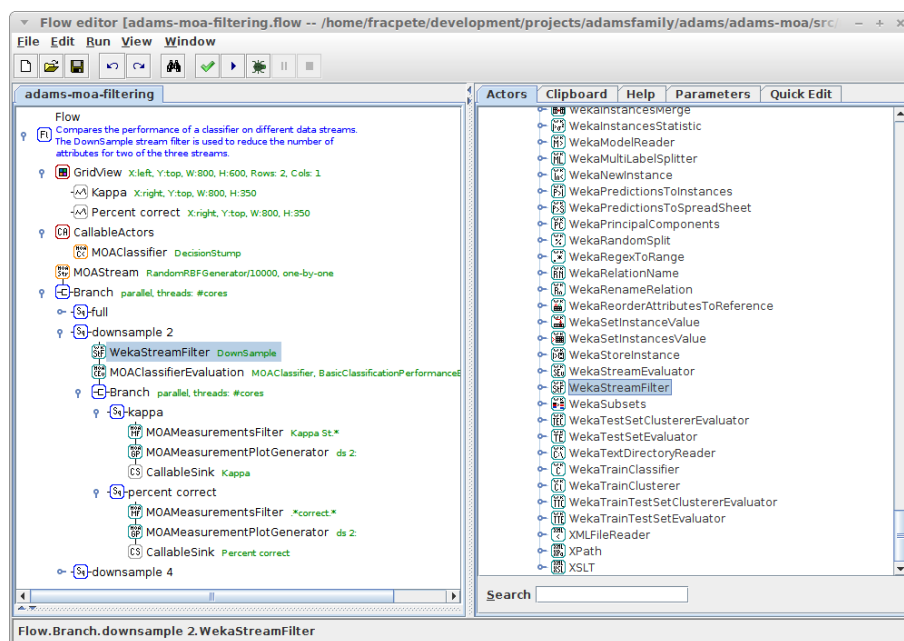


Figure 2.9: Filtering data streams.

⁸adams-moa-filtering.flow

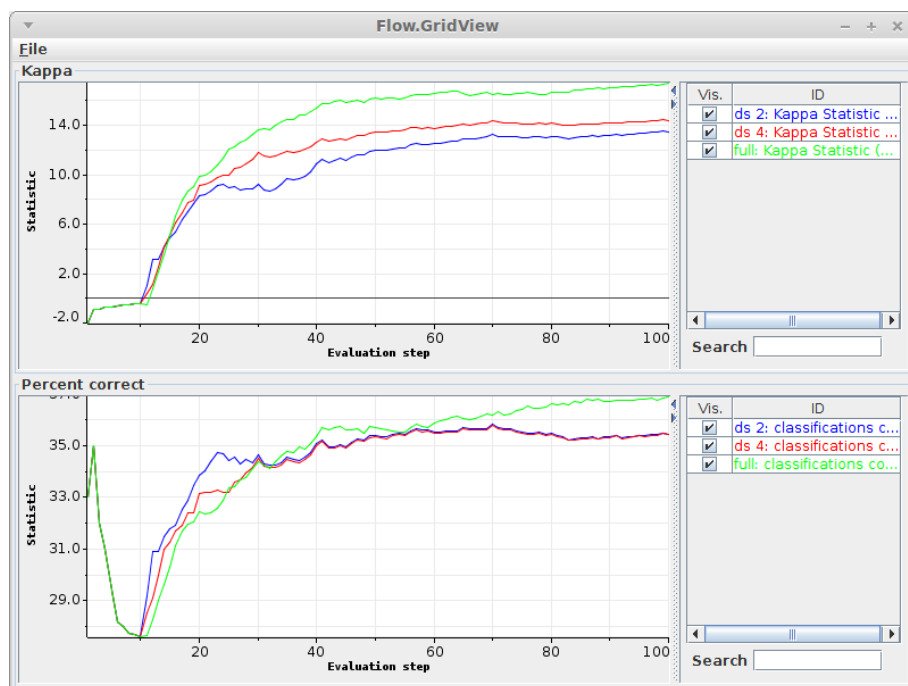


Figure 2.10: Comparison of streams, filtered and unfiltered.

2.6 Provenance

Just like with WEKA, provenance is supported by MOA's actors as well. In Figure 2.11 you can see a flow⁹ that will display the provenance information that the tokens accumulated, from generation through to evaluation. Figure 2.12 then shows the visualization of the provenance trace.

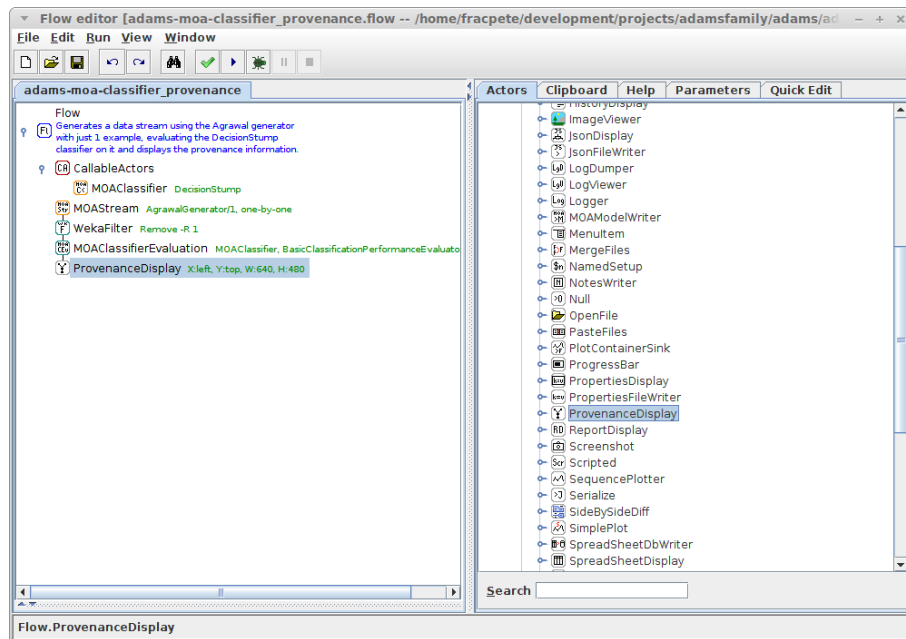


Figure 2.11: Flow for displaying provenance information.

⁹adams-moa-classifier_provenance.flow

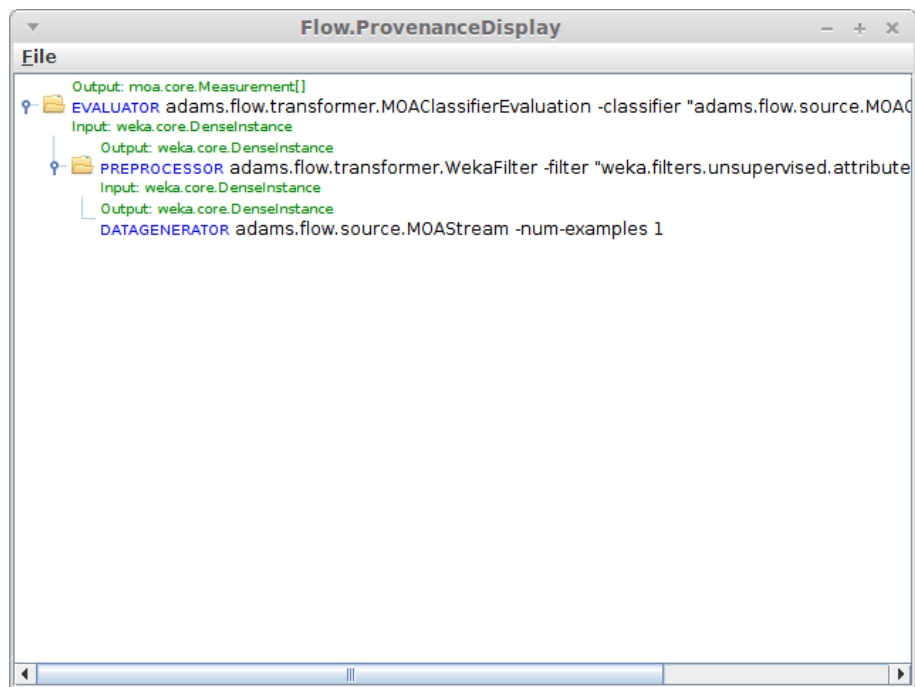


Figure 2.12: The generated provenance trace.

Chapter 3

Tools

The main interface for MOA is available from within ADAMS as well. You can find it under the *MOA* menu. Figure 3.1 shows a screenshot of the user interface in action.

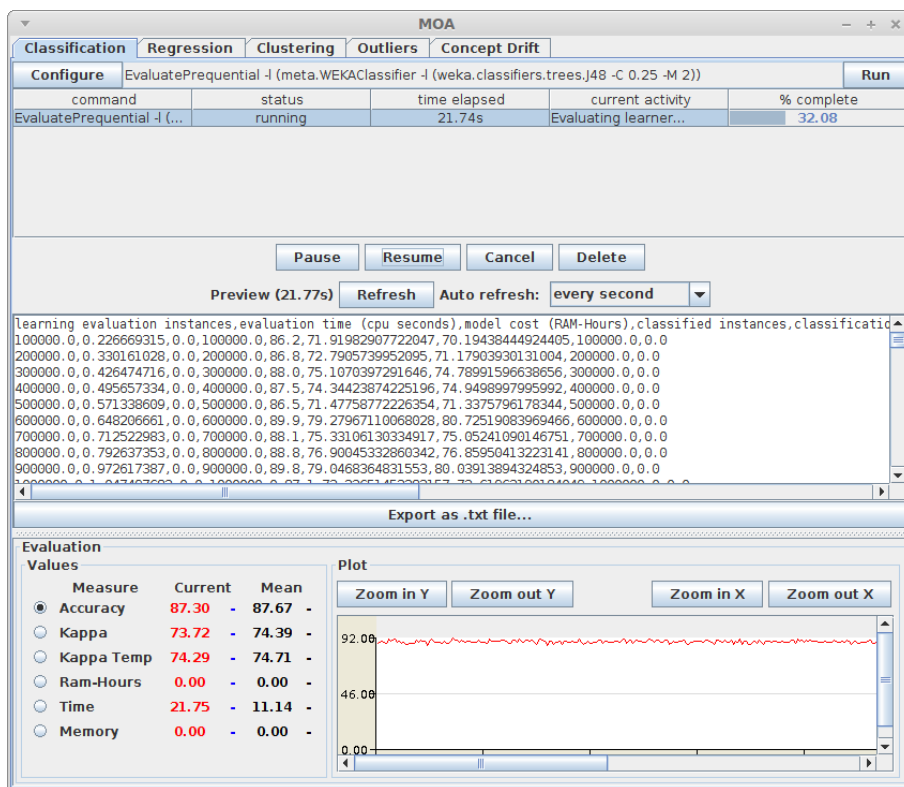


Figure 3.1: The main MOA interface.

Bibliography

- [1] *ADAMS* – Advanced Data mining and Machine learning System
<https://adams.cms.waikato.ac.nz/>
- [2] Albert Bifet, Geoff Holmes, Richard Kirkby, Bernhard Pfahringer (2010).
MOA: Massive Online Analysis; Journal of Machine Learning Research
(JMLR), Volume 11, pp 1601–1604.
<http://moa.cms.waikato.ac.nz/>