Introduction to ADAMS and MOA
Tutorial

Peter Reutemann, Albert Bifet and Richard Kirkby

November 2012

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

YAHOO!
RESEARCH

1 Getting Started

This tutorial is a basic introduction to ADAMS and MOA.

WEKA and MOA are powerful tools to perform data mining analysis tasks. Usually, in real applications and professional settings, the data mining processes are complex and consist of several steps. These steps can be seen as a workflow. Instead of implementing a program in JAVA, a professional data miner will build a solution using a workflow, so that it will be much easier to maintain for non-programmer users.

The Advanced Data mining And Machine learning System (ADAMS) is a novel, flexible workflow engine aimed at quickly building and maintaining real-world, complex knowledge workflows.

Flow
Generates a data stream using the Agrawal generator
with 10,000 examples, evaluates the DecisionStump
classifier every 100 and displays the result of the
evaluation in textual format and the statistics in
sequence plots.

GlobalActors
MOAClassifier DecisionStump
GridView X:left, Y:top, W:800, H:800, Rows: 3, Cols: 1
Kappa X:right, Y:top, W:800, H:350
Percent correct X:right, Y:top, W:800, H:350
Display X:left, Y:top, W:640, H:480
MOAStream generators.AgrawalGenerator/10000
MOAClassifierEvaluation MOAClassifier, BasicClassificationPerformanceEvaluator/100
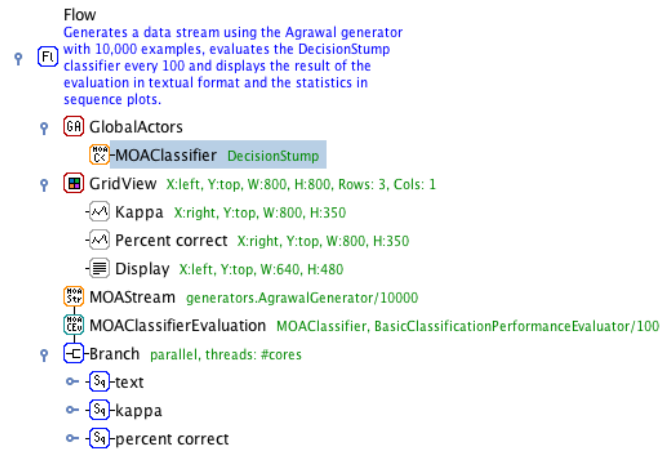Branch parallel, threads: #cores
text
kappa
percent correct

Figure 1: ADAMS flow example

The core of ADAMS is the workflow engine, which follows the philosophy of less is more. Instead of letting the user place operators (or actors in ADAMS terms) on a canvas and then manually connect inputs and outputs, ADAMS uses a tree-like structure. This structure and the control actors define how the data is flowing in the workflow, no explicit connections necessary. The tree-like structure stems from the internal object representation and the nesting of sub-actors within actor-handlers.

Massive **O**nline **A**nalysis (MOA) is a software environment for running experiments and implementing methods for mining evolving data streams.
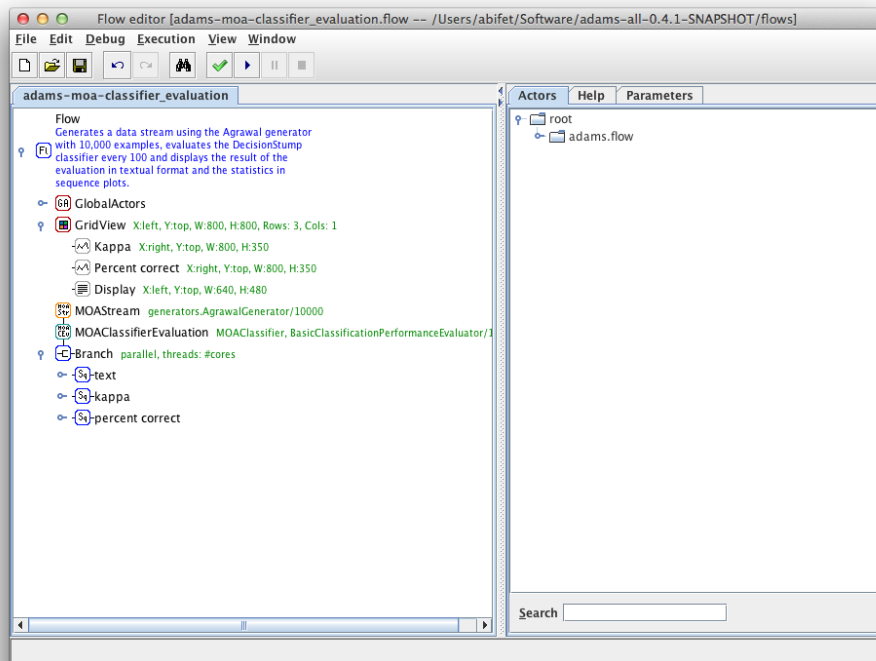

Figure 2: ADAMS Flow Editor

We suppose that ADAMS is installed in your system. Start the Flow Editor and load the **adams-moa-classifier-evaluation** flow. Run the flow. Note that:

- Kappa statistic is a measure of agreement of a classifier with a random one. If Kappa statistic is zero, the classifier is performing as a random one, and if it is equal to one, then it is not random at all.

- A decision stump is a decision tree with only one split node.

Exercise 1 *Explain what this workflow does. Discuss why accuracy is increasing and kappa statistic is decreasing. What happens if you replace the decision stump by a Hoeffding Tree?*
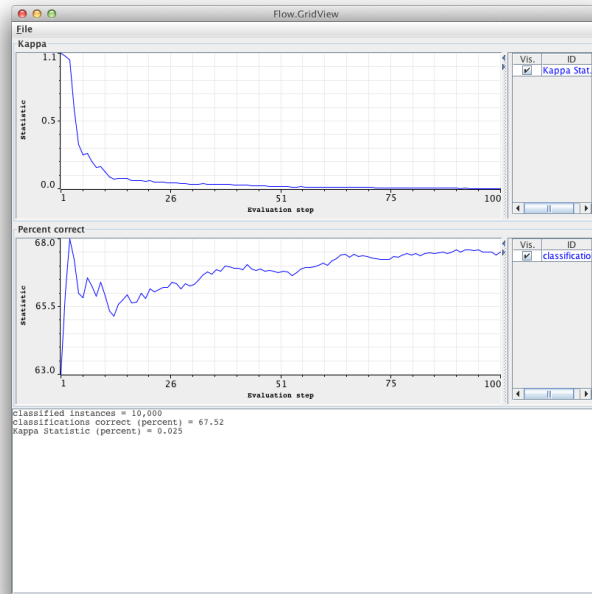

Figure 3: ADAMS flow example

2 Classification using ADAMS

We start comparing the accuracy of two classifiers. First, we explain briefly two different data stream evaluations.

2.1 Data streams Evaluation

The most significant requirements for a data stream setting are the following:

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

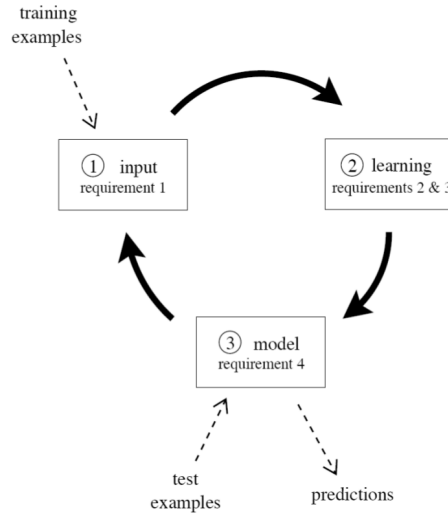**Requirement 2** Use a limited amount of memory

3

Figure 4: The data stream classification cycle

Requirement 3 Work in a limited amount of time

Requirement 4 Be ready to predict at any time

Figure 4 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).

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.

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 have 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.

Holdout evaluation gives a more accurate estimation of the accuracy of the classifier on more recent data. However, it requires recent test data that it is difficult to obtain for real datasets. Gama et al. propose to use a forgetting mechanism for estimating holdout accuracy using prequential accuracy: a sliding window of size w with the most recent observations, or fading factors that weigh observations using a decay factor α. The output of the two mechanisms is very similar (every window of size w_0 may be approximated by some decay factor α_0).

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.

2.2 Exercises

To familiarize yourself with the functions discussed so far, please do the following two exercises, loading and modifying the **adams-moa-compare-classifiers** flow.

Exercise 2 *Compare the accuracy of the Hoeffding Tree with the Naive Bayes classifier, for a RandomTreeGenerator stream of 10,000 instances using Interleaved Test-Then-Train evaluation.*

Exercise 3 *Compare and discuss the accuracy for the same stream of the previous exercise using two different evaluations with a Hoeffding Tree:*

- *Interleaved Test Then Train*

- *Prequential with a sliding window of 100 instances.*

2.3 Drift Stream Generators

MOA streams are build using generators, reading ARFF files, joining several streams, or filtering streams. MOA streams generators allow to simulate potentially infinite sequence of data.

Two streams evolving on time are:

- Rotating Hyperplane

- Random RBF Generator

To model concept drift we only have to set up the drift parameter of the stream.

2.4 Exercises

Exercise 4 *Compare the accuracy of the Hoeffding Tree with the Naive Bayes classifier, for a RandomRBFGenerator stream of 50,000 instances with speed change of 0,001 using Interleaved Test-Then-Train evaluation.*

Exercise 5 *Compare the accuracy for the same stream of the previous exercise using three different classifiers:*

- *Hoeffding Tree with Majority Class at the leaves*

- *Hoeffding Adaptive Tree*

- *OzaBagAdwin with 10 HoeffdingTree*

3 Using MOA from the Graphical User Interface and the command line

Start a MOA graphical user interface for configuring and running tasks.

Click 'Configure' to set up a task, when ready click to launch a task click 'Run'. Several tasks can be run concurrently. Click on different tasks in the list and control them using the buttons below. If textual output of a task is available it will be displayed in the middle of the GUI, and can be saved to disk.
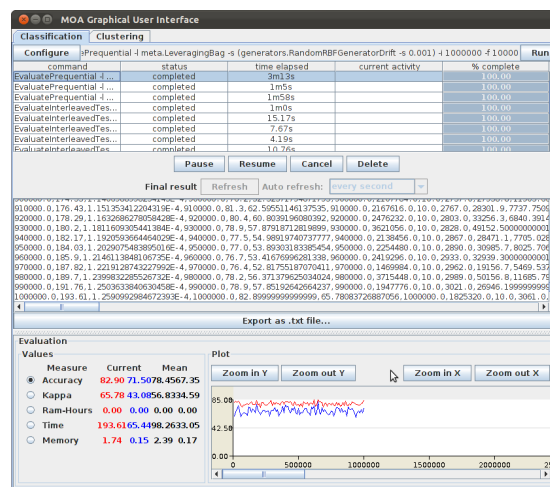

Figure 5: MOA Graphical User Interface

Note that the command line text box displayed at the top of the window represents textual commands that can be used to run tasks on the command line. The text can be selected then copied onto the clipboard. In the bottom of the GUI there is a graphical display of the results. It is possible to compare the results of two different tasks: the current task is displayed in red, and the selected previously is in blue.

An easy way to use the command line, is to copy and paste the text in the Configuration line of the Graphical User Interface.

For example, suppose we want to process the task

```
EvaluatePrequential -l trees.HoeffdingTree -i 1000000 -w 10000
```

using the command line. We simply write

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask  \
"EvaluatePrequential -l trees.HoeffdingTree -i 1000000 -w 10000"
```

Note that some parameters are missing, since they use default values.

3.1 Learning and Evaluating Models

The `moa.DoTask` class is the main class for running tasks on the command line. It will accept the name of a task followed by any appropriate parameters. The first task used is the `LearnModel` task. The `-l` parameter specifies the learner, in this case the `HoeffdingTree` class. The `-s` parameter specifies the stream to learn from, in this case `generators.WaveformGenerator` is specified, which is a data stream generator that produces a three-class learning problem of identifying three types of waveform. The `-m` option specifies the maximum number of examples to train the learner with, in this case one million examples. The `-O` option specifies a file to output the resulting model to:

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask \
  LearnModel -l trees.HoeffdingTree \
  -s generators.WaveformGenerator -m 1000000 -O model1.moa
```

This will create a file named `model1.moa` that contains the decision stump model that was induced during training.

The next example will evaluate the model to see how accurate it is on a set of examples that are generated using a different random seed. The `EvaluateModel` task is given the parameters needed to load the model produced in the previous step, generate a new waveform stream with a random seed of 2, and test on one million examples:

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask \
  "EvaluateModel -m file:model1.moa \
  -s (generators.WaveformGenerator -i 2) -i 1000000"
```

This is the first example of nesting parameters using brackets. Quotes have been added around the description of the task, otherwise the operating system may be confused about the meaning of the brackets.

After evaluation the following statistics are output:

```
classified instances = 1,000,000
classifications correct (percent) = 84.474
Kappa Statistic (percent) = 76.711
```

Note the the above two steps can be achieved by rolling them into one, avoiding the need to create an external file, as follows:

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask \
  "EvaluateModel -m (LearnModel -l trees.HoeffdingTree \
  -s generators.WaveformGenerator -m 1000000) \
  -s (generators.WaveformGenerator -i 2) -i 1000000"
```

The task `EvaluatePeriodicHeldOutTest` will train a model while taking snapshots of performance using a held-out test set at periodic intervals. The following command creates a *comma separated values* file, training the `HoeffdingTree` classifier on the `WaveformGenerator` data, using the first 100 thousand examples for testing, training on a total of 100 million examples, and testing every one million examples:

```
java -cp moa.jar -javaagent:sizeofag.jar moa.DoTask \
  "EvaluatePeriodicHeldOutTest -l trees.HoeffdingTree \
  -s generators.WaveformGenerator \
  -n 100000 -i 10000000 -f 1000000" > dsresult.csv
```

3.2   Exercises

Exercise 6 *Repeat the experiments of exercises 2 and 3 using the MOA graphical interface and the command line.*

Exercise 7 *Compare accuracy and RAM-Hours needed using a prequential evaluation (sliding window of 1,000 instances) of 1,000,000 instances for a Random Radius Based Function stream with speed of change 0,001 using the following methods:*

- *OzaBag with 10 HoeffdingTree*

- *OzaBagAdwin with 10 HoeffdingTree*

- *LeveragingBag with 10 HoeffdingTree*