This is a portion of the eBook at doi:10.7551/mitpress/10654.001.0001

14 Using the API

MOA can be used from external Java code by calling its API. It is very easy to use the generators, methods, and tasks of MOA inside your Java applications. In this chapter we show how to use the MOA API and include stream ML capabilities inside your programs. We assume basic knowledge of Java and object-oriented programming.

14.1 MOA Objects

The basic objects available from the MOA API are:

- `Task`: All tasks in MOA follow this interface. All tasks that write their result to a file must extend `MainTask` and implement the `doMainTask` method.
- `InstanceStream`: Streams used in MOA use this interface. To use a stream, call `prepareForUse` to prepare it and then `nextInstance` to get each new instance from the stream.
- `Classifier`: Classifiers should extend `AbstractClassifier`. To use a classifier, first call `prepareForUse` to prepare it, and then use `trainOnInstance` to train the model with a new instance, or `getVotesForInstance` to get the classification predictions for an instance.

14.2 Options

MOA classes that have parameters or options have to extend the class `AbstractOptionHandler`. These options can be of the following types:

- Integer: `IntOption (name, char, purpose, default Value, min Value, max Value)`
- Float: `FloatOption (name, char, purpose, default Value, min Value, max Value)`
- Flag: `FlagOption (name, char, purpose)`
- File: `FileOption (name, char, purpose, default File name, default File extension, Output)`
- String: `StringOption (name, char, purpose, default Value)`

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024

This is a portion of the eBook at doi:10.7551/mitpress/10654.001.0001

- Multichoice: `MultichoiceOption (name, char, purpose, option labels, option descriptions, default option index)`
- Class: `ClassOption (name, char, purpose, required type, default CLI string)`
- List: `ListOption (name, char, purpose, expected option type, default list, separator char)`

These are some examples of options defined as Java classes:

Listing 14.1: Definition of option classes with default values.

```
public IntOption gracePeriodOption = new
    IntOption(
         "gracePeriod",
         'g',
         "The number of instances a leaf should
            observe between split attempts.",
         200, 0, Integer.MAX_VALUE);

public ClassOption splitCriterionOption = new
    ClassOption("splitCriterion",
         's', "Split criterion to use.",
            SplitCriterion.class,
         "InfoGainSplitCriterion");

public FloatOption splitConfidenceOption = new
    FloatOption(
         "splitConfidence",
         'c',
         "The allowable error in split decision,
            values closer to 0 will take longer
            to decide.",
         0.0000001, 0.0, 1.0);

public FlagOption binarySplitsOption = new
    FlagOption("binarySplits", 'b',
         "Only allow binary splits.");
```

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024

This is a portion of the eBook at doi:10.7551/mitpress/10654.001.0001


```
public FileOption dumpFileOption = new
    FileOption("dumpFile", 'd',
         "File to append intermediate csv
            results to.", null, "csv", true);

public StringOption xTitleOption = new
    StringOption("xTitle", 'm',
         "Title of the plots' x-axis.",
            "Processed instances");
```

To change the values of these options from your Java source code via API, there are two possibilities:

- `setValueViaCLIString(String s)`: Sets the value for this option via text that can contain several nested parameters, similar to the CLI text.
- Each option has a particular method to change its value:
  - Integer: `setValue(int v)`
  - Float: `setValue(double v)`
  - Flag: `setValue(boolean v)`
  - File: `setValue(String v)`
  - String: `setValue(String v)`
  - Multichoice: `setChosenIndex(int index)`
  - Class: `setCurrentObject(Object obj)`
  - List: `setList(Option[] optList)`

  There are also two ways to get the values of these options:

- `getValueAsCLIString()`: Gets the text that describes the nested parameters of this option.
- Each option has a particular method to get its value:
  - Integer: `getValue()`
  - Float: `getValue()`
  - Flag: `isSet()`
  - File: `getFile()`

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024

This is a portion of the eBook at doi:10.7551/mitpress/10654.001.0001

- String: `getValue()`
- Multichoice: `getChosenIndex()`
- Class: `getPreparedClassOption(ClassOption classOption)`
- List: `getList()`

14.3 Prequential Evaluation Example

As an example, we will write Java code to perform prequential evaluation.

We start by initializing the data stream from which we will read instances. In this example we will use the RandomRBFGenerator, explained in section 12.2.2.

Listing 14.2: Java code for stream initialization.

```
RandomRBFGenerator stream = new
    RandomRBFGenerator();
stream.prepareForUse();
```

Now we have to build an empty learner. We create a Hoeffding Tree, and tell it the information about instance attributes using `setModelContext`.

Listing 14.3: Java code for learner initialization.

```
Classifier learner = new HoeffdingTree();
learner.setModelContext(stream.getHeader());
learner.prepareForUse();
```

To perform prequential evaluation, we have to first test and then train on each one of the instances in the stream. At the same time, we keep some accuracy statistics to be able to compute the final accuracy value.

Listing 14.4: Java code for prequential evaluation.

```
int numInstances=10000;
int numberSamplesCorrect=0;
int numberSamples=0;
boolean isTesting = true;
```

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024

This is a portion of the eBook at doi:10.7551/mitpress/10654.001.0001


```
while(stream.hasMoreInstances() &&
    numberSamples < numInstances){
     Instance inst =
         stream.nextInstance().getData();
     if(isTesting){
          if(learner.correctlyClassifies(inst)){
              numberSamplesCorrect++;
          }
     }
     numberSamples++;
     learner.trainOnInstance(inst);
}
```

Finally, we output the results of the evaluation. In our case, we are interested in accuracy, so we print the final prequential accuracy of the HoeffdingTree in this setting.

Listing 14.5: Java code to output result.

```
double accuracy = 100.0*(double)
    numberSamplesCorrect /
    (double)numberSamples;
System.out.println(numberSamples+" instances
    processed with "+accuracy+"% accuracy");
```

The complete Java code of our evaluation method is:

Listing 14.6: Complete Java code.

```
RandomRBFGenerator stream = new
    RandomRBFGenerator();
stream.prepareForUse();

Classifier learner = new HoeffdingTree();
learner.setModelContext(stream.getHeader());
learner.prepareForUse();

int numInstances=10000;
```

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024

This is a portion of the eBook at doi:10.7551/mitpress/10654.001.0001


```
int numberSamplesCorrect=0;
int numberSamples=0;
boolean isTesting = true;
while(stream.hasMoreInstances() &&
    numberSamples < numInstances){
     Instance inst =
         stream.nextInstance().getData();
     if(isTesting){
          if(learner.correctlyClassifies(inst)){
              numberSamplesCorrect++;
          }
     }
     numberSamples++;
     learner.trainOnInstance(inst);
}
double accuracy = 100.0*(double)
    numberSamplesCorrect /
    (double)numberSamples;
System.out.println(numberSamples+" instances
    processed with "+accuracy+"% accuracy");
```

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024

This is a section of doi:10.7551/mitpress/10654.001.0001

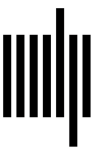
Machine Learning for Data Streams

with Practical Examples in MOA

By: Albert Bifet, Ricard Gavaldà, Geoffrey Holmes, Bernhard Pfahringer

Citation:

Machine Learning for Data Streams: with Practical Examples in MOA
By: Albert Bifet, Ricard Gavaldà, Geoffrey Holmes, Bernhard Pfahringer
DOI: 10.7551/mitpress/10654.001.0001
ISBN (electronic): 9780262346047
Publisher: The MIT Press
Published: 2023



The MIT Press

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024

MIT Press Direct

© 2017 Massachusetts Institute of Technology

All rights reserved. No part of this book may be reproduced in any form or by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

This book was set in Times Roman and Mathtime Pro 2 by the authors.

Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data is available

ISBN: 978-0-262-03779-2

10 9 8 7 6 5 4 3 2 1

Downloaded from http://direct.mit.edu/books/oa-monograph/chapter-pdf/2263151/9780262346047_cdx.pdf by guest on 19 June 2024