Tutorial. MEKA 1.9.1

April 2017

A Multilabel/multitarget Extension to WEKA.
http://meka.sourceforge.net

Contents

1 Introduction

This is a tutorial for the open source machine learning framework MEKA. MEKA is closely based upon the WEKA framework [2]; providing support for development, running and evaluation of *multi-label* and *multi-target* classifiers (which WEKA does not).

In the *multi-label* problem, a data instance may be associated with multiple labels. This is as opposed to the traditional task of single-label classification (i.e., multi-class, or binary) where each instance is only associated with a single class label. The multi-label context is receiving increased attention and is applicable to a wide variety of domains, including text, music, images and video, and bioinformatics. A good introduction can be found in [7] and [3].

The multi-label problem is in fact a special case of *multi-target* learning. In multi-target, or *multi-dimensional* learning, a data instance is associated with multiple target variables, where each variable takes a number of values. In the multi-label case, all variables are binary, indicating label relevance (1) or irrelevance (0). The multi-target case has been investigated by, for example, [9] and [10].

MEKA can also includes *incremental* classifiers suitable for the *data streams* context. An overview of some of the methods included in MEKA for learning from incremental data streams is given in [4].

MEKA is released under the GNU GPL v3 licence. The latest release, source code, API reference, this tutorial, and further information and links to additional material, can be found at the website: `http://meka.sourceforge.net`.

This tutorial applies to MEKA version `1.9.1`.

2 Getting Started

MEKA can be download from: `http://sourceforge.net/projects/meka/files/`. This tutorial is written for version `1.9.1`; and assumes that you have downloaded and extracted the `meka-release-1.9.1.zip` and that the `meka-1.9.1/`, found within, is your current working directory.

2.1 Requirements

MEKA requires:

- Java version 1.7 or above

MEKA comes bundled with other packages such as WEKA's and MULAN's `mulan.jar`.These files are found in the `lib/` directory.

2.2 Running

MEKA can be used very easily from the command line. For example, to run the Binary Relevance (BR) classifier on the *Music* dataset; type:

```
java -cp "./lib/*" meka.classifiers.multilabel.BR -t data/Music.arff
```

If you are on a Microsoft Windows system, you need to use back slashes instead of forward slashes (`.\lib\*`). If you add the `jar` files to the system's `CLASSPATH`, you do not need to

supply the -cp option at runtime. For the remainder of examples in this tutorial we will assume that this is the case.

Since Version 1.2 MEKA has a graphical user interface (GUI). Run this with either the run.sh script (under Linux or OSX) as follows:

```
./run.sh
```

Run run.bat instead if you are using Microsoft windows.

3 MEKA's Dataset Format

MEKA uses WEKA's ARFF file format. See http://weka.wikispaces.com/ARFF to learn about this format. MEKA uses multiple attributes – one for each target or label – rather than a single class attribute. The *number* of target attributes is specified with either -C or -c; *unlike* in WEKA where the -c flag indicates the position of the *class index*. MEKA uses the reference to the classIndex internally to denote the number of target attributes.

Since the number of target attributes tends to vary with each dataset, for convenience MEKA allows this option (as well as other dataset options like the train/test split percentage) to be stored in the @relation name of an ARFF file, where a colon (:) is used to separate the dataset name and the options. The following is an example ARFF header for multi-target classification with three target variables and four attributes:

```
@relation 'Example_Dataset: -C 3'

@attribute category {A,B,C,NEG}
@attribute label {0,1}
@attribute rank {1,2,3}
@attribute X1 {0,1}
@attribute X2 {0,1}
@attribute X3 numeric
@attribute X4 numeric

@data
```

Note that the format of the label attribute (binary) is the *only* kind of target attribute in multi-*label* datasets. For more examples of MEKA ARFF files; see the data/ directory for several multi-label and multi-target datasets (some of these are in a compressed format).

MEKA can also read ARFF files in the MULAN format where target attributes are the *last* attributes, rather than the first ones. This format can also be read by MEKA by specifying a minus sign '-' before the number of target attributes in the -C option. For example, -C -3 will set the *last* 3 attributes as the target attributes automatically when the file is loaded. Alternatively, the target attributes can be moved using WEKA's Reorder filter, or in the GUI as detailed in the following section.

3.1 Manipulating Datasets in the GUI

A good way to set up an ARFF file for multi-dimensional classification is using the GUI. Open an ARFF file with 'Open' from the File menu. In the Preprocess tab in the right-hand column (see Figure 1; note that the class attributes are listed in **bold** face), simply select the attributes you wish to use as class attributes and click the button 'Use class attributes'. You can then save

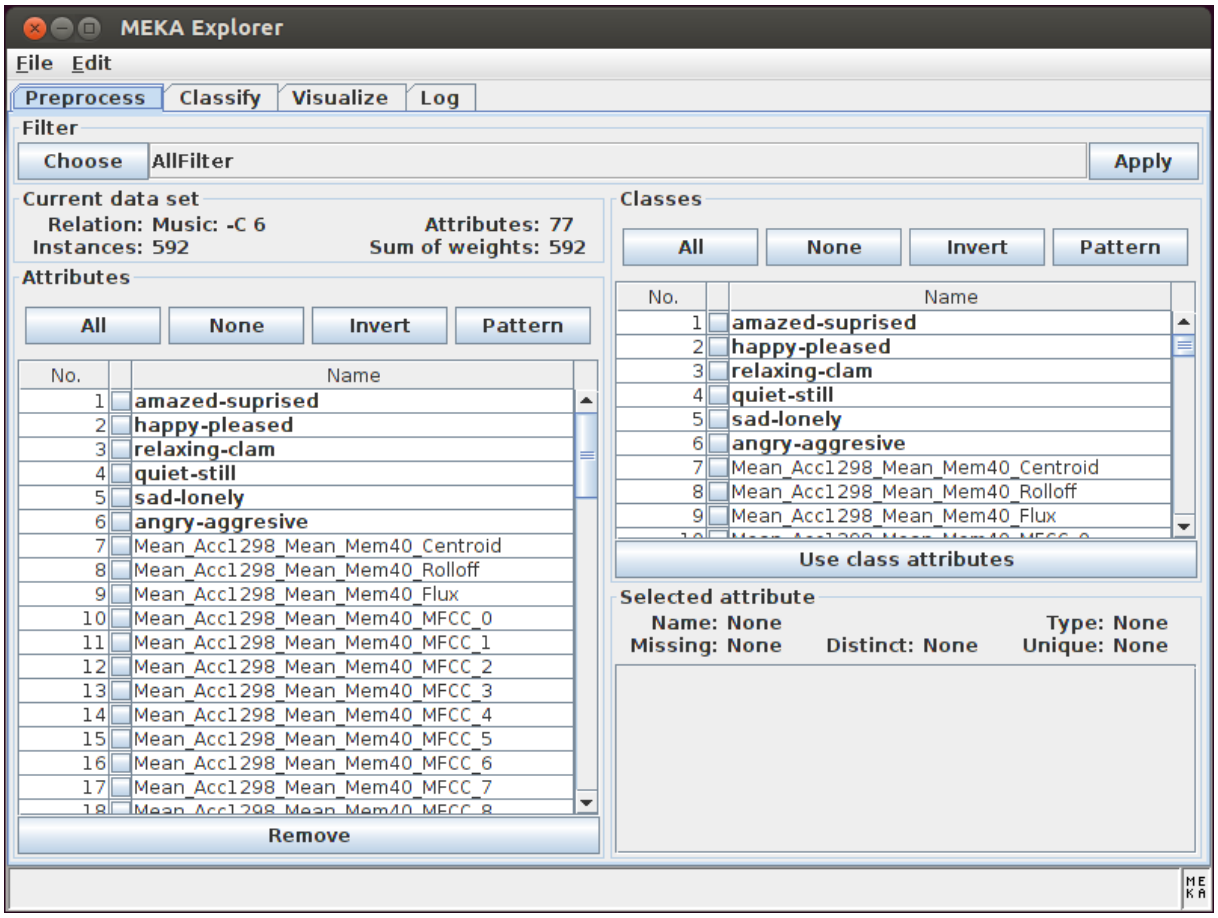

Figure 1: MEKA's GUI interface; having loaded the *Music* dataset.

this file using 'Save' from the File menu, which will also save the `-C` flag into the `@relation` tag as described above (displayed under 'Relation:' in the GUI), so next time the classes will be set automatically.

The datasets that come with MEKA already come with the `-C` flag specified correctly, so you do not need to set this information.

You can also run any of WEKA's filters on the dataset with the Choose button. See the WEKA documentation for more information.

4 Using MEKA

A suitable dataset is the only requirement to begin running experiments with MEKA.

4.1 Command Line Interface

With the exception of the different use of the `-c` flag (see the previous section), many of WEKA's command line options for evaluation work identically in MEKA too. You can obtain a list of them by running any classifier with the `-h` flag, for example running `java meka.classifiers.multilabel.BR -h` on the command line results in the output:

```
java.lang.Exception: [Error] You did not specify a dataset!


Evaluation Options:

-h
  Output help information.
-t <name of training file>
  Sets training file.
-T <name of test file>
  Sets test file (will be used for making predictions).
-predictions <name of output file for predictions>
  Sets the file to store the predictions in (does not work with cross-validation).
-x <number of folds>
  Do cross-validation with this many folds.
-no-eval
  Skips evaluation, e.g., used when test set contains no class labels.
-R
  Randomize the order of instances in the dataset.
-split-percentage <percentage>
  Sets the percentage for the train/test set split, e.g., 66.
-split-number <number>
  Sets the number of training examples, e.g., 800
-i
  Invert the specified train/test split.
-s <random number seed>
  Sets random number seed (use with -R, for different CV or train/test splits).
-threshold <threshold>
  Sets the type of thresholding; where
    'PCut1' automatically calibrates a threshold (the default);
    'PCutL' automatically calibrates one threshold for each label;
    any number, e.g. '0.5', specifies that threshold.
-C <number of labels>
  Sets the number of target variables (labels) to assume (indexed from the beginning).
-d <classifier_file>
  Specify a file to dump classifier into.
-l <classifier_file>
  Specify a file to load classifier from.
-verbosity <verbosity level>
  Specify more/less evaluation output


Classifier Options:

-W
  Full name of base classifier.
  (default: weka.classifiers.trees.J48)
-output-debug-info
  If set, classifier is run in debug mode and
  may output additional info to the console
--do-not-check-capabilities
  If set, classifier capabilities are not checked before classifier is built
  (use with caution).
-num-decimal-places
```

Note that the only required option is just `-t` to specify the dataset (and `-C` to specify the number of target attributes, if not already included within the dataset `@relation` tag, as explained in the previous section). Note also that the `Classifier Options` are specific to each classifier (which in this case is `BR`), and each base classifier (for `BR`, the default base classifier is `J48`). The `J48`-specific options (which are truncated in this output) are visible following the hyphen (`-`).

BR has limited options, for example to change the base classifier (the -W option). For example, to use Naive Bayes, type[1] on the command line:

```
java meka.classifiers.multilabel.BR -t data/Music.arff \
  -W weka.classifiers.bayes.NaiveBayes
```

4.1.1 Evaluation

Running BR with NaiveBayes on the *Music* data will output the following:

```
Picked up JAVA_TOOL_OPTIONS: -Djsse.enableSNIExtension=false -XX:-OmitStackTraceInFastThrow -
    Dsun.java2d.xrender=false
== Evaluation Info

Classifier                      meka.classifiers.multilabel.BR
Options                         [-W, weka.classifiers.bayes.NaiveBayes]
Additional Info
Dataset                         Music
Number of labels (L)            6
Type                            ML
Threshold                       0.9974578524138343
Verbosity                       1


== Predictive Performance

Number of test instances (N)
Accuracy                        0.436
Jaccard index                   0.436
Hamming score                   0.745
Exact match                     0.215


== Additional Measurements

Number of training instances    355
Number of test instances        237
Label cardinality (train set)   1.789
Label cardinality (test set)    1.992
Build Time                      0.264
Test Time                       0.145
Total Time                      0.409
```

Note that by increasing the verbosity level, you can get more or less output. On the command line, this is with -verbosity <n> where <n> =

verbosity	Output
1 (default)	basic output
2	plus more metrics
3	plus individual metrics (for each label)
4	plus more individual metrics (for each label)
5	plus individual classifications
6	plus individual confidence outputs (rounded to 1 d.p.)
7	... (rounded to 2 d.p.)
8	... (rounded to 3 d.p.)

[1] If typed on one line, the backslash '\' should be omitted

Note that some measures may be computationally intensive to calculate. Low verbosity levels will lead to faster evaluation times. Note also that if verbosity is too low some options (for example to draw ROC curves) will appear as disabled in the GUI.

Most of these metrics are described in [6, 7]. The most common metrics reported in the multi-label literature are `Hamming Loss` (evaluation by label, averaged across all labels) and `Exact Match` (evaluation of each example, averaged across all examples). Multi-label `Accuracy` (also called `Jaccard Index` in information retrieval) is a good compromise between both these metrics.

Note that a `Threshold` can be calibrated automatically whenever `-threshold PCut`, by minimizing the difference between the label cardinality of the training set and the predictions on the test set[2] [6], the so-called *P-Cut* method. To calibrate a threshold the same way but for each label individual, use instead `-threshold PCutL` option. In Prequential evaluation, only a fixed threshold is allowed.

MEKA also supports Cross-validation; for example:

```
java meka.classifiers.multilabel.BR -x 10 -R -t data/Music.arff \
  -W weka.classifiers.bayes.NaiveBayes
```

conducts 10 fold cross validation on a randomised version of the *Music* dataset and outputs the average results across all folds with standard deviation. Note that under incremental evaluation, the `-x` option is reused to indicate the number of samples to take.

4.1.2 Examples

In the following we give some examples, with an emphasis on general usage and parameters. An extensive list of examples is given at `http://meka.sourceforge.net/methods.html`.

Binary Relevance (BR) On the *Music* data, loading from two separate sets, using Naive Bayes as a base classifier, calibrating a separate threshold automatically for each label:

```
java meka.classifiers.multilabel.BR \
  -t data/Music_train.arff \
  -T data/Music_test.arff \
  -threshold PCutL \
  -W weka.classifiers.bayes.NaiveBayes
```

Ensembles of Pruned Sets (EPS; see [5]) With 10 ensemble members (the default) on the *Enron* dataset with Support Vector Machines as the base classifier; each `PS` model is set with `N=1` and `P` to a random selection of {`1,2,3,4,5`}:

```
java meka.classifiers.multilabel.meta.EnsembleML \
  -t data/Yeast.arff \
  -W meka.classifiers.multilabel.PS -- \
    -P 1-5 -N 1 -W weka.classifiers.functions.SMO
```

[2]This does *not* require access to the true predictions in the test set

Ensembles of Classifier Chains (ECC; see [6]) With 50 ensemble members (`-I 50`), and some textual output (`-output-debug-info`) on the *Enron* dataset with Support Vector Machines as a base classifier:

```
java meka.classifiers.multilabel.meta.BaggingML -I 50 -P 100 \
  -t data/Enron.arff  \
  -output-debug-info -W meka.classifiers.multilabel.CC -- \
    -W weka.classifiers.functions.SMO
```

Mulan Classifier (`RAkEL` see [8]) With parameters `k=3, m=2C` where `C` is the number of labels (these options are hardwired; you need to edit `MULAN.java` to specify new parameter configurations) on the *Scene* dataset with Decision Trees as the base classifier (remember `mulan.jar` must be in the classpath):

```
java meka.classifiers.multilabel.MULAN -t data/Scene.arff -verbosity 5 \
  -S RAkEL2 -W weka.classifiers.trees.J48
```

the `-verbosity 5` options increases the amount of evaluation output.

Binary Relevance (BR) On the *Music* data, loading from two separate sets, using Naive Bayes as a base classifier, performing no evaluation and outputting the collected predictions to a file:

```
java meka.classifiers.multilabel.BR \
  -t data/Music_train.arff \
  -T data/Music_test.arff \
  -predictions output/Music_predictions.arff \
  -no-eval \
  -W weka.classifiers.bayes.NaiveBayes
```

Incremental Classification: Ensembles of Binary Relevance (see [6, 4]) With 10 ensemble members (default) on the *Enron* dataset with `NaiveBayesUpdateable` as a base classifier; prequential evaluation taking 20 samples of performance over time:

```
java meka.classifiers.multilabel.incremental.meta.BaggingMLUpdateable\
  -x 20 -t data/Enron.arff \
  -W meka.classifiers.multilabel.incremental.BRUpdateable -- \
    -W weka.classifiers.bayes.NaiveBayesUpdateable
```

Under *prequential evaluation*, the cumulative *final* evaluation is reported, and `x` samples are stored along the way (these can be visualised as a plot over time from within the GUI). An initial window is used for initial training but is not included in evaluation, the equivalent to $1/(x+1)$-th of the data; the remaining data is divided into `x` windows. If using *batch-incremental evaluation*, evaluation is carried out in batches, but this is non-cumulative (the result is reset before each new batch). This mode is selected with `-b` on the command line. The `-x` option is, in this case, considered as the number of batches (minus an original batch used only for training – similarly to under the prequential option).

Multi-target: Ensembles of Class Relevance (see [9]) The multi-target version of the Binary Relevance classifier) on the *solar flare* dataset with Logistic Regression as a base classifier under 10-fold cross-validation:

```
java meka.classifiers.multitarget.meta.BaggingMT -x 10 -R \
  -t data/solar_flare.arff \
  -W meka.classifiers.multitarget.CR -- \
    -W weka.classifiers.functions.Logistic
```

Semi-supervised: EM with CC with Logistic Regression as a base classifier, using a separate set of unlabelled data (in two separate commands):

```
java meka.classifiers.multilabel.meta.EM -t data-train.arff \
  -T data-unlabelled.arff -d em.saved
java meka.classifiers.multilabel.meta.EM -t data-train.arff \
  -T data-test.arff -l em.saved
```

Note that the training set is required again in the second command because when `PCutL` is used (currently the default) it is necessary to know the label cardinality of the training data in order to calibrate a threshold for the test data.

Updateable Binary Relevance (BRUpdateable) Prequentially trained/evaluated on the *Music* data, and the final model batch-evaluated against the test set:

```
java meka.classifiers.multilabel.BR \
  -t data/Music_train.arff \
  -T data/Music_test.arff \
  -W weka.classifiers.bayes.NaiveBayes
```

Updateable Binary Relevance (BRUpdateable) On the *Music* data, incrementally built on the training set and then outputting the predictions generated for the test set to the specified predictions file, outputting no evaluation:

```
java meka.classifiers.multilabel.BR \
  -t data/Music_train.arff \
  -T data/Music_test.arff \
  -predictions output/Music_predictions.arff \
  -no-eval \
  -W weka.classifiers.bayes.NaiveBayes
```

4.2 Graphical User Interface

In recent versions of Meka, the GUI is as fully-functioned as the command line. In fact, it has additional visualization functionality. Refer to Section 2.2 on how to open the GUI. Once opened, you will get to chose between the Explorer and the Experimenter.

4.2.1 The Explorer

You will see a number of tabs: Preprocess, Classify, Visualize, and Log. The following process will guide you through a simple scenario.

1. Load a dataset file using Open from the file menu.
2. Click on the Classify tab.

3. Choose a multi-label or multi-target classifier (or leave the default choice).

4. Click on the label to the right of this button and set specific options to the classifier. In most cases, these options also involve setting a WEKA single-label base classifier (and also its options). For MEKA's `meta` classifiers, you will need to first choose a multi-label base classifier, and then a single-label (WEKA) base classifier for this classifier. See, for example, Figure 2, using a `BaggingML` ensemble of `CC` (classifier chains) with `SMO` as the single-label base classifier.

5. In the Evaluation panel you configure what type of evaluation you want to do, and some of the options given in the previous section are available here. For example, a 55/45 train/test split, with `PCutL` threshold calibration, as being specified in Figure 3.

6. When you click Start the experiment will be run. When finished, the result will appear in the History panel, as in Figure 4. This is the same output as would be seen on the command line, and explained in the following section. The verbosity of the results is controlled by the Verbosity option (see Section 4.1.1)

7. (Optional) View the Precision-Recall curves by right-clicking on the results and selecting Show Precision-Recall. See Figure 5

8. (Optional) If you chose a `MultiLabelDrawable` classifier (such as `BR`, `CC`, `LP`, or any derivatives) and a `Drawable` base classifier such as `J48`, you can right click on the results and select Show Graphs. See Figure 6

9. (Optional) Click on the Visualize tab inspect the dataset.

10. (Optional) To remember later what you have done, Save the log from under the Log tab.

Note that the threshold auto-calibration option of `PCutL` being specified in Figure 3, (note that one threshold is calibrated for each label in Figure 4.

4.2.2 The Experimenter

As in the Explorer, you will see a number of tabs. The following process will guide you through a simple scenario.

1. Select File → New → Default Experiment

2. Click Add... (on the left pane) to add a classifier configuration

3. Repeat the process of adding a classifier

4. Click Add... (on the right pane) to add a classifier configuration

5. Configure the number of Runs and Folds for Cross-validation

6. Click on the KeyValuePairs label, and then on the . labels to select a **file** to save the experiment

7. Click OK and then Apply from the main tab

8. Select Execution → Start from the menu

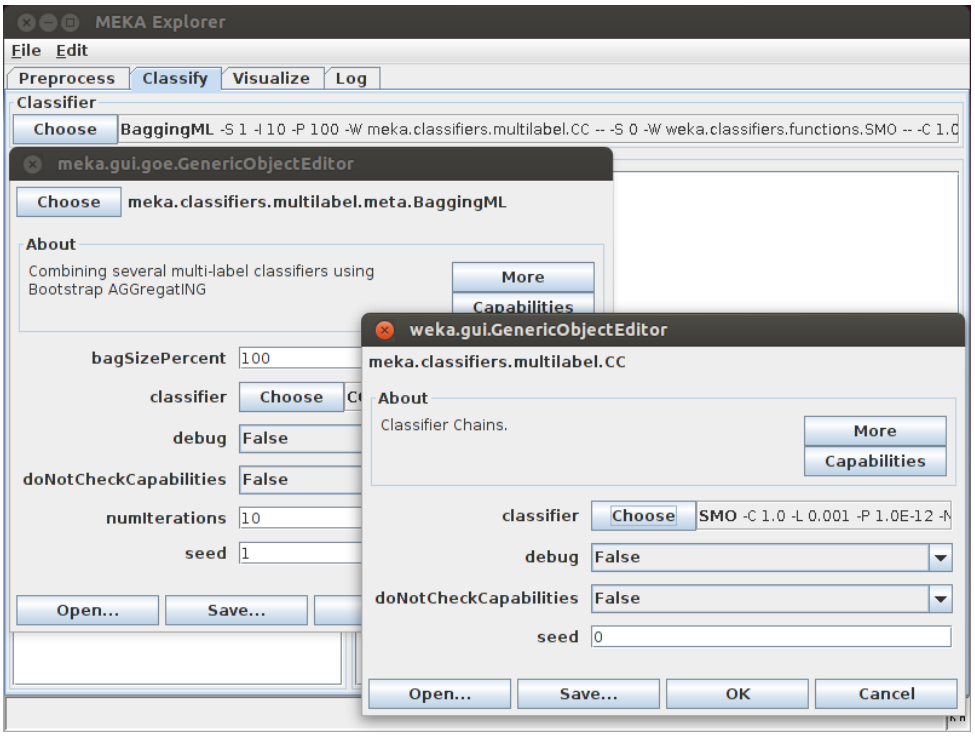

Figure 2: MEKA's GUI interface; setting Bagging of Classifier Chains with SMO.

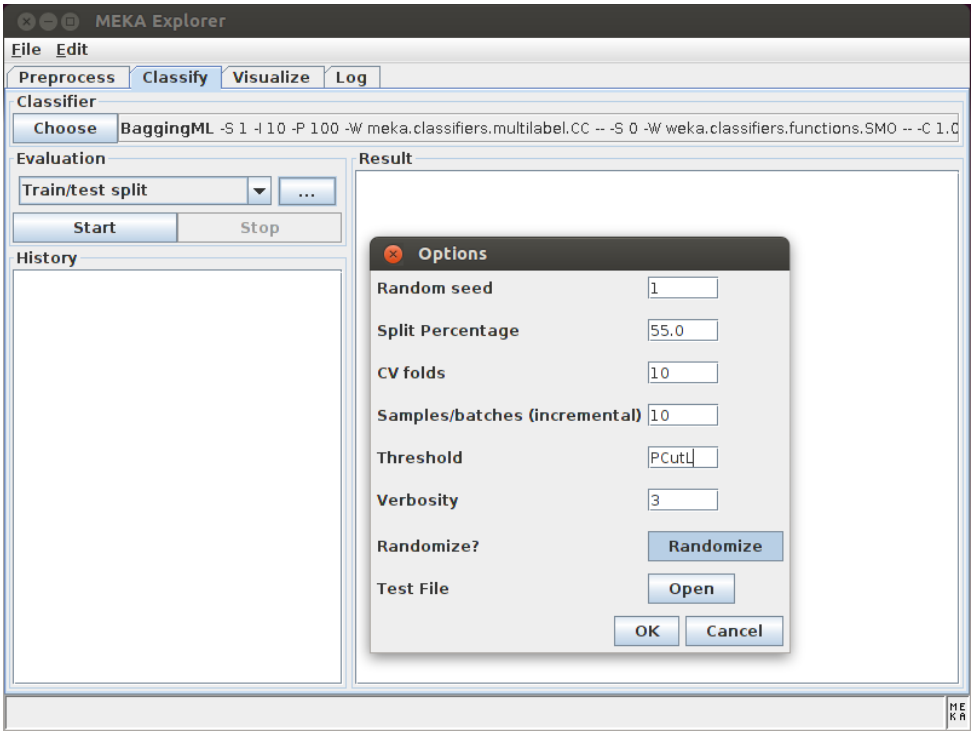

Figure 3: MEKA's GUI interface; setting a train/test split.

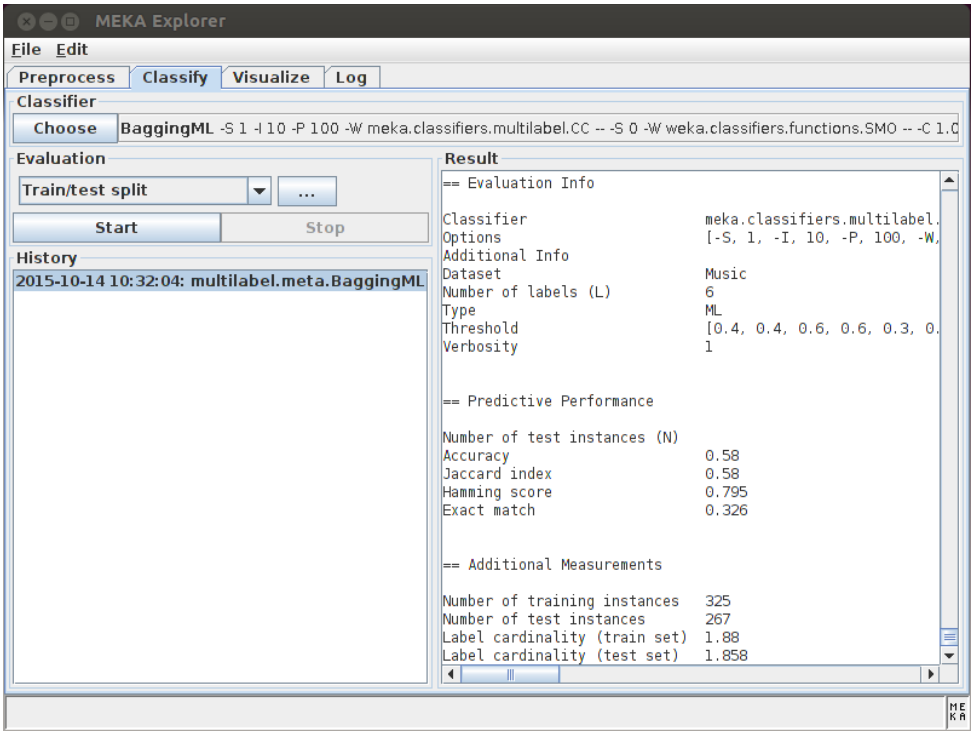

Figure 4: MEKA's GUI interface; results.

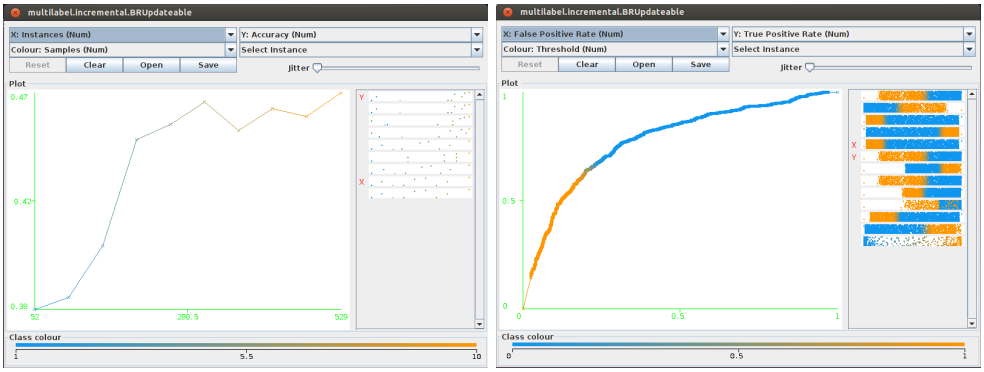

Figure 5: MEKA's GUI interface; viewing incremental performance over time (left) and a ROC curve (right).

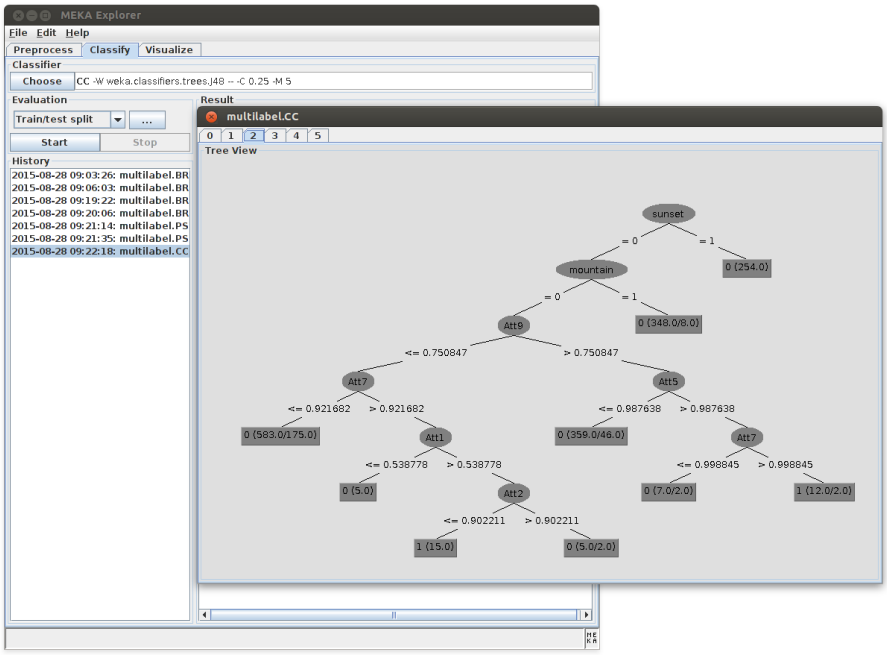

Figure 6: MEKA's GUI interface; viewing a Decision Tree model of `CC`.

9. The MEKA icon will animate and you will see Running... on the taskbar
10. When it is finished, click on the Statistics tab
11. View the Aggregated subtab
12. Click Statistics → Save as (aggregated) to save this table as either a `tsv` or `tex` file
13. View the Measurement subtab
14. Select a Measurement that you are particularly interested in, e.g., Exact match
15. Click Statistics → Save as (measurement) to save this table as either a `tsv` or `tex` file
16. A `tex` file can be imported directly into a latex document with the `\input` command

If you wish to carry out this process from within Java code, you can inspect the file `./src/main/java/meka/experiment/ExperimentExample.java` to see how it can be done.

5 Development

The following sections explain a bit more in detail of how to obtain MEKA's source code, how to compile it and how to develop new algorithms.

5.1 Source code

For obtaining the source code of MEKA, you have two options:

- Using subversion[3]
- Release archive

In the case of *subversion*, you can obtain the source code using the following command in the console (or *command prompt* for Windows users):

```
svn checkout svn://svn.code.sf.net/p/meka/code/trunk meka
```

This will create a new directory called `meka` in the current directory, containing the source code and build scripts.

Instead of using subversion, you can simply use the source code that is part of each MEKA release, contained in the `meka-src-1.9.1.jar` Java archive[4]. A Java archive can be opened with any archive manager that can handle ZIP files.

5.2 Compiling

Using `Maven`
MEKA uses *Apache Maven*[5] as build tool. You can compile MEKA as follows:

```
mvn clean install
```

If you want to generate an archive containing all source code, pdfs and binary jars, then you can use the following commands (make sure that the version in the `release.xml` file matches the one in the `pom.xml`):

```
mvn clean install
mvn -f release.xml package
```

Please note, if you develop new algorithms, you should also create a unit test for it, to ensure that it is working properly. See section 5.3 for more details.

Using IntelliJ IDEA
After obtaining the source code, you can simply import Meka as a Maven project. Select File → New → Project from Existing Sources from the main menu and point to the directory containing the `pom.xml` file.

Using Eclipse
If you are using the *Eclipse IDE for Java Developers*, then you can import as a Maven project. Select File → Import... and then select Maven → Existing Maven Projects. After selecting the root directory with MEKA's `pom.xml`, click on Finish.

5.3 Unit tests

MEKA uses the JUnit 3.8.x unit testing framework.
A classifier test case is derived from the following abstract super class:

[3] `http://subversion.apache.org/`
[4] `http://en.wikipedia.org/wiki/JAR_(file_format)`
[5] `http://maven.apache.org/`

`meka.classifiers.AbstractMekaClassifierTest`

A filter test case is derived from the following abstract super class:

`meka.filters.AbstractMekaFilterTest`

You can execute all the unit tests by calling the following class from the command-line:

```
mvn test
```

5.4 Extending MEKA

MEKA classifiers are regular WEKA classifiers that extend either the `MultilabelClassifier` or `MultiTargetClassifier` class (most of them are `ProblemTransformationMethods`), and expect the `classIndex()` of `Instances` and `Instancess` to indicate the number of target attributes (indexed at the beginning) rather than the class index (see Section 3).

The following is an example of a functioning (but extremely minimalistic) classifier, `TestClassifier`, that predicts 0-relevance for all labels:

```
package meka.classifiers.multilabel;
import weka.core.*;

public class TestClassifier extends ProblemTransformationMethod {

    public void buildClassifier(Instances D) throws Exception {
        testCapabilities(D);
        int C = D.classIndex();
    }

    public double[] distributionForInstance(Instance x) throws Exception
        {
        int C = x.classIndex();
        return new double[C];
    }

    public static void main(String args[]) {
        MultilabelClassifier.runClassifier(new TestClassifier(),args);
    }
}
```

This shows how easy it is to create a new classifier. If it is not a *problem transformation method*, then use the appropriate WEKA superclass and implement the `MultiLabelClassifier` interface (`ProblemTransformationMethod` implements this interface). For more useful examples see the source code of existing MEKA classifiers. The `testCapabilities(D)` line is optional but highly recommended. Note that the `distributionForInstance` method returns a `double[]` array exactly like in WEKA. However, whereas in WEKA, there is one value in the array for each possible value of the single target attribute, in MEKA this function returns an array of C values, where C is the *number* of target attributes, and the jth value of the array is the *value* corresponding to the jth target attribute.

5.4.1 Building Classifiers

In the `buildClassifier(Instances)` method, you build your classifier. Here is where you can take advantage of all of WEKA's libraries. You can use any WEKA classifier to your needs.

The m_Classifier variable is already available for this, which *already contains* the WEKA classifier you specify on the command line. You can simply do:

```
    public void buildClassifier(Instances D) throws Exception {
        testCapabilities(D);
        int C = D.classIndex();
        D.setClassIndex(0);
        m_Classifier.buildClassifier(D);
    }
```

to train a classifier to learn the first label of your data (using the other labels, and all other input-space feature attributes). So if you then run on the command line

```
java meka.classifiers.multilabel.TestClassifier -t data/Music.arff \
  -W weka.classifiers.functions.SMO
```

an instantiation of SMO will already be be available in m_Classifier. You can also do this explicitly with

```
...
    m_Classifier = new SMO();
    m_Classifier.buildClassifier(D);
...
```

5.4.2 Classifying New Instances

In the multi-label case, for a test Instance x, the double[] array returned by the method distributionForInstance(x) might contain the 0/1 label relevances, for example (assuming -C 5):

```
[0.0, 0.0, 1.0, 1.0, 0.0]
```

or it might contain posterior probabilities / prediction confidences / votes for each label, for example:

```
[0.1, 0.0, 0.9, 0.9, 0.2]
```

where clearly the third and fourth labels are most relevant. Under a threshold of 0.5 the final classification for x would be [0,0,1,1,0]. MEKA will by default automatically calibrate a threshold to convert all values into 0/1 relevances like these (see Section 4.1.1).

In the multi-target case, the double[] values returned by the method distributionForInstance must indicate the *relevant value*; for example (assuming -C 3):

```
[3.0, 1.0, 0.0]
```

If this were the dataset exemplified in Section 3, this classification would be C,1,1 for the class attributes category,label,rank, respectively.

Therefore, at the current release, thresholds are not automatically calibrated for multi-target data.

Associated probabilistic/confidence may be stored in an extended part of the array, e.g., [C+1,...,2C], for example (if -C 6):

```
[3.0, 1.0, 0.0, 0.5, 0.9, 0.9]
```

where the 3-rd *value* is predicted for the first *target variable*, with confidence 0.5, and so on. Note, however, that this mechanism will be replaced in future versions of MEKA.

5.4.3 Incremental Classifiers

MEKA comes with incremental versions of many classifiers as well as incremental evaluation methods, located in the `meka/classifiers/multilabel/incremental/` folder. Incremental classifiers implement WEKA's `UpdatebleClassifier` interface and therefore must implement the `updateClassifier(Instance)` method. The following extends `TestClassifier` for incremental learning.

```
package meka.classifiers.multilabel.incremental;
import meka.classifiers.multilabel.TestClassifier;
import weka.classifiers.UpdateableClassifier;
import weka.core.*;

public class TestClassifierUpdateable extends TestClassifier
    implements UpdateableClassifier{

    public void updateClassifier(Instance x) throws Exception {
        int L = x.classIndex();
    }

    public static void main(String args[]) {
        IncrementalEvaluation.runExperiment(
            new TestClassifierUpdateable(),args);
    }
}
```

Note that the `IncrementalEvaluation` class is called for evaluation in this case; see Section 4.1.1, and example in Section 4.1.2. Note also the MOA framework [1] contains more possibilities than WEKA, and its classifiers can be used via WEKA's `MOA` meta classifier (wrapper), e.g.,

```
java meka.classifiers.multilabel.BRUpdateable -t data/Music.arff \
  -W weka.classifiers.meta.MOA -- -B moa.classifiers.trees.HoeffdingTree
```

Note that MOA now also supports MEKA classifiers via a wrapper class.

5.4.4 Semi-supervised Classifiers

Semi-supervised classifiers implement `SemisupervisedClassifier` interface and must implement the method `setUnlabelledData(Instances)` which is called automatically by the `Evaluation` class prior to `buildClassifier(Instances)`. It is up to the classifier to decide what to do with the unlabelled instances. Currently, MEKA assumes that unlabelled instances are the test instances (e.g., supplied with the `-T` flag). It sets all labels to missing beforehand just in case. In the case of a separate set of unlabelled instances, a model can be built, given unlabelled instances with `-T`, and then saved/dumped into a file with the `-d <filename>` option, and loaded again (with the `-l` option) along with a *different* set with `-T`. See the example in Section 4.1.2.

5.5 Miscellaneous

Note that MEKA's home directory is OS-specific,

- In Microsoft Windows: `%USERPROFILE%\mekafiles`
- In Unix-based systems: `$HOME/.meka`

Note that the `.props` files have preference the following order:

1. the `jar`
2. user's home directory
3. MEKA's home directory
4. current directory

6 Getting Help / Reporting Bugs / Contributing

A list of methods available using MEKA is maintained at `http://meka.sourceforge.net/methods.html` with descriptions and command-line examples.

If you need help with MEKA, you can post your problem on Meka's Mailing List: `http://sourceforge.net/p/meka/mailman/`.

If you have found a bug with MEKA, you can report in via the Tracker of MEKA's `SourceForge.net` site: `http://sourceforge.net/p/meka/bugs/`.

If you would like to contribute to MEKA, such as adding new classifiers, please get in touch with the developers.

More information (such as contact information) can be found at the MEKA website: `http://meka.sourceforge.net`.

References

[1] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa massive online analysis, 2010. `http://mloss.org/software/view/258/`.

[2] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Reutemann Peter, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[3] Jesse Read. *Scalable Multi-label Classification*. PhD thesis, University of Waikato, 2010.

[4] Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoffrey Holmes. Scalable and efficient multi-label classification for evolving data streams. *Machine Learning*, 2012. Accepted for publication.

[5] Jesse Read, Bernhard Pfahringer, and Geoff Holmes. Multi-label classification using ensembles of pruned sets. In *ICDM'08: Eighth IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2008.

[6] Jesse Read, Bernhard Pfahringer, Geoffrey Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

[7] G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*. 2nd edition, Springer, 2010.

[8] Grigorios Tsoumakas and Ioannis P. Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *ECML '07: 18th European Conference on Machine Learning*, pages 406–417. Springer, 2007.

[9] Julio H. Zaragoza, Luis Enrique Sucar, Eduardo F. Morales, Concha Bielza, and Pedro Larrañaga. Bayesian chain classifiers for multidimensional classification. In *24th International Conference on Artificial Intelligence (IJCAI '11)*, pages 2192–2197, 2011.

[10] Bernard Zenko and Saso Dzeroski. Learning classification rules for multiple target attributes. In *PAKDD '08: Twelfth Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 454–465, 2008.