Tutorial. MEKA 1.0

Jesse Read

May 2012

MEKA

A Multilabel/multitarget Extension to WEKA.

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 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 `1.0`.

2 Getting Started

MEKA can be download from: `http://meka.sourceforge.net`. This tutorial is written for version `1.0`. The rest of this tutorial assumes that the extracted package folder is your working directory.

2.1 Requirements

MEKA requires:

- Java version 1.6 or above

MEKA comes bundled with WEKA's `weka.jar`. If WEKA is already installed on your system, it must be at least version release (`3.7.X`) to be compatible with `Meka`.

Similarly, for running MULAN classifiers from MEKA, this package is necessary. We have included the latest *compatible* `mulan.jar` in the main MEKA package.

2.2 Running

MEKA does not yet have a graphical user interface. However, it can be used very easily from the command line. Any classifier can be run directly, by suppling the relevant `jar` files to the Java Runtime Environment. For example, to run the Binary Relevance (BR) classifier on the *Enron* dataset; type:

```
java -cp meka.jar:weka.jar weka.classifiers.multilabel.BR -t data/Enron.arff
```

If you are on a Microsoft Windows system, the `jar`s in the classpath are separated by a semicolon '`;`' instead of a colon. 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.

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 -split-percentage 50'

@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 in the Meka package which includes several multi-label and multi-target datasets.

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 class attributes can be moved using Weka's `Reorder` filter.

4 Using Meka: Running Experiments

Deciding on (or creating) a suitable dataset is the only requirement to begin running experiments with Meka.

4.1 Command Line Options

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: `java weka.classifiers.multilabel.BR -h` displays the following:

```
Evaluation Options:

-h
    Output help information.
-t <name of training file>
    Sets training file.
-x <number of folds>
    Do cross-validation with this many folds.
-p
    Specify a range in the dataset (@see weka.core.Range)
-R
    Randomise the dataset
    (done after a range is removed, but before the train/test split)
-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.
-T <threshold>
    Sets the type of thresholding; where
    'c' automatically calibrates a threshold (the default);
    'C' automatically calibrates one threshold for each label; and
    any double number, e.g. 0.5, specifies that threshold.
-C <number of target attributes>
    Sets the number of target attributes to expect
    (indexed from the beginning).
```

The only required options are `-t` to specify the dataset, and `-C` to specify the number of target attributes; the latter is typically included within the dataset, as explained in the previous section.

Below this output, we also see the output for `Classifier Options`:

```
Classifier Options:

-D
        If set, classifier is run in debug mode and
        may output additional info to the console
-W
        Full name of base classifier.
        (default: weka.classifiers.rules.ZeroR)
```

which in this case (for `java weka.classifiers.multilabel.BR`) are not very extensive. However, to get decent results with this classifier, we will have to specify a more competitive *base classifier* with the `-W` option, for example Naive Bayes. To run this on the *Music* data, we would type[1]:

[1] If typed on one line, the bar '\' should not be typed into the command line

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

4.2 Evaluation

The example used above for running a BR classifier with Naive Bayes on the *Enron* data, will output the following:

```
Classifier_name : weka.classifiers.multilabel.BR
 Classifier_ops : [-W, weka.classifiers.bayes.NaiveBayes, --, , , ]
Classifier_info :
   Dataset_name : Music
      Threshold : 0.9974578524138343
           Type : ML

              N : 237
              L : 6
       Accuracy : 0.436
         H_loss : 0.255
          H_acc : 0.745
    Exact_match : 0.215
   ZeroOne_loss : 0.785
      One_error : 0.414
       LogLossD : 7.302
       LogLossL : 2.972
      Precision : 0.629
         Recall : 0.564
       F1_micro : 0.594
     F1_macro_D : 0.508
     F1_macro_L : 0.551
   EmptyVectors : 0.177
     LCard_pred : 1.785
     LCard_real : 1.992

        N_train : 355.0
         N_test : 237.0
    LCard_train : 1.7887323943661972
     LCard_test : 1.9915611814345993
     Build_time : 0.148
      Test_time : 0.118
     Total_time : 0.266
```

Most of these measures are described in [3, 6, 7]. The most common measures in the multi-label literature are *Hamming Loss* (`H_loss`), which is the accuracy for each label, averaged across all labels; *Exact Match* (`Exact_match`), which is the is the accuracy of each *example* – where all label relevances must match exactly for an example to be correct; and *Accuracy* (`Accuracy`), which is neither as strict as Exact Match nor as 'easy' as Hamming Loss.

Note that a `Threshold` has been calibrated automatically; to minimise the difference between the label cardinality of the training set and the predictions on the test set; a practice described in [6]. To calibrate a threshold for *each* label, add the `-T C` option. This gives a vector of thresholds which, in this case, increases `Accuracy` slightly (to `0.456`). Different thresholds are calculated for different methods and datasets.

Sometimes it can be useful to save predictions and information about an experiment. MEKA can produce plain-text files with the option `-f <file name>`; for example:

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

which produces the plain-text file `BR-NB.dat`. This provides a way to analyse individual classifications and evaluate other software with MEKA's evaluation framework. The results can be displayed again with:

```
java weka.core.Result -f out.dat
```

MEKA also supports *cross validation*; for example:

```
java weka.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.arff* data and outputs the average results across all folds with standard deviation.

More examples are given in the following subsection.

4.3 Examples

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

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

```
java weka.classifiers.multilabel.meta.BaggingML -I 50 -D \
  -t data/Enron.arff -W weka.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) on the *Scene* dataset with Decision Trees as the base classifier (`mulan.jar` from the MULAN package must be added to the classpath):

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

Incremental Classification: Ensembles of Binary Relevance (see [6, 4]) With 10 ensemble members (default) on the *Enron* dataset with `NaiveBayesUpdateable` as a base classifier (`mice.jar` from the MEKA package must be added to the classpath):

```
java weka.classifiers.multilabel.meta.EnsembleMLUpdateable \
  -t data/Enron.arff \
  -W weka.classifiers.multilabel.BRUpdateable -- \
    -W weka.classifiers.bayes.NaiveBayesUpdateable
```

Evaluating incremental classifiers will carry out evaluation and display statistics for the data over 19 evaluation windows (an initial window is used for the initial training; making 20 windows in total). Note that the MOA framework [1] for evaluating incremental classifiers is much more sophisticated; and a new release is will include a wrapper to MEKA.

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 5-fold cross-validation:

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

5 Extending MEKA: Writing New Classifiers

Writing MEKA classifiers involves writing regular WEKA classifiers that extend either the `MultilabelClassifier` or `MultiTargetClassifier` class, and expect the `classIndex()` of `Instances` and `Instances`s to indicate the number of target attributes (indexed at the beginning) rather than the class index (as explained in Section 3).

The easiest way to extend MEKA is to create new Java classes within the existing MEKA folder hierarchy and recompile the simply by typing:

```
ant jar
```

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

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

public class TestClassifier extends MultilabelClassifier {

    public void buildClassifier(Instances D) throws Exception {
        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 can be used for a template for creating classifiers. For more useful examples see the source code of the MEKA classifiers. 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.

5.1 Multi-label Classifiers

In the multi-label case, for a test `Instance x`, the `double[]` array returned by the method `distributionForInstance(x)` might look like (assuming `-C 5`):

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

where clearly the third and fourth labels are most relevant. Thus, these values may be label relevances, votes, or posterior probabilities. Under a threshold of 0.5 the final classification for `x` would be `[0,0,1,1,0]`. MEKA will automatically calibrate a threshold to convert all values into 0/1 relevances like these (see Section 4.2). Naturally, a multi-label classifier may return 0/1 binary relevances directly (represented as doubles).

5.2 Multi-target Classifiers

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

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

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

Note that no threshold is calibrated. However, any associated voting or probabilistic values may be stored in the following `C+1`, . . . , `2C` values; for example (again assuming `-C 6`):

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

where `C` is predicted as the value of the first target attribute with confidence 0.9, and so on. However these values are currently only for use at *classification time* (for example the voting scheme of an ensemble method, see `weka.classifiers.multitarget.BaggingMT`); and not taken into account for evaluation.

5.3 Incremental Classifiers

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

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

public class TestClassifierUpdateable extends MultilabelClassifier
    implements UpdateableClassifier{

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

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

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

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

Note that the `WindowIncrementalEvaluator` class is called for evaluation in this case; see Section 4.2. The Moa framework [1] for learning in data streams is currently being developed to support multi-label classification with a wrapper for Meka classifiers, and will offer more types of evaluation for incremental classification in a data stream.

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.