Chapter 9
Multilabel Software

Abstract Multilabel classification and other learning from multilabeled data tasks are relatively recent, with barely a decade of history behind them. When compared against binary and multiclass learning, the range of available datasets, frameworks, and other software tools is significantly more scarce. The goal of this last chapter is to provide the reader with the proper insight to take advantage of these software tools. A brief overview of them is offered in Sect. 9.1. Section 9.2 discusses the different multilabel file formats, enumerates the data repositories the MLDs can be downloaded from, and describes how to automate some tasks with the mldr.datasets R package. How to perform exploratory data analysis of MLDs is the main topic of Sect. 9.3. Then, the process to conduct experiments with multilabel data using different tools is outlined in Sect. 9.4.

9.1 Overview

Despite the software shortage aforementioned above, currently there are some multilabel data repositories, as well as two frameworks for algorithm developers and at least one exploratory data tool. By using them, tasks such as downloading, citing and partitioning datasets, multilabel data exploration, and conducting experiments with existent MLC algorithms will be at your fingertips.

The present chapter has been structured into three main sections. The first one describes the tools needed to work with multilabel data. This includes details about MLDs file formats, data repositories MLDs can be obtained from, and how most of these tasks can be accomplished by means of a specific software tool, the mldr.datasets R package.

How to perform exploratory analysis of multilabel data is the topic the second section is dedicated to. To do so, two specific programs are depicted, the R mldr package and the Java MEKA framework. Many of the plots in this book have been produced by the former, a tool which also provides methods to filter and transform MLDs.

The concern of the third and final section is how to conduct multilabel experiments, by means of MEKA, MULAN, and a specific Java utility developed by the

F. Herrera et al., *Multilabel Classification*,
DOI 10.1007/978-3-319-41111-8_9

authors. Following these guidelines, and using the data partitions provided in this book repository, the reader should be able to reproduce the experiments described in previous chapters.

9.2 Working with Multilabel Data

The design of any algorithm aimed to deal with multilabeled data, whether its goal is to induce a classification model or to apply some kind of preprocessing, has a key requirement, it will have to be tested against some MLDs. Therefore, whatever is the researching goal, the first step will usually be obtaining enough multilabel data. These MLDs will have to be partitioned, and commonly some exploratory analysis would have to be conducted on them. In addition, they have to be properly documented into the research projects they are used, including the correct citing information.

Fortunately, nowadays there are several data repositories and software tools to fulfill these needs. This first section provides a brief description of such resources, along with useful references to obtain them.

9.2.1 Multilabel Data File Formats

One of the first issues that any multilabel researcher or practitioner (the user henceforward) has to face is the disparate set of MLDs file formats. Unlike traditional datasets, MLDs have more than one output attribute, so that the last feature is the class label cannot be assumed. How to communicate which ones of the features are labels is the main origin of the several file formats, because each developer came with a different approach to solve this obstacle.

Most MLDs are written using one of two base file formats, CSV (*Comma-Separated Values*) or ARFF[1] (*Attribute-Relation File Format*). Both are text file formats, but the latter includes a header with data descriptors followed by the data itself, whereas the former usually only provides the data and the header, if it is present, only brings field names. CSV files cannot contain label information, so the knowledge of how many labels there are, where are they located, or what are their names will depend on an external resource. By contrast, an ARFF file can include this information into the header.

[1]An ARFF file is usually divided into three sections. The first one contains the name of the dataset after de `@relation` tag, the second one provides information about the attributes with `@attribute` tags, and the third one, whose beginning is marked with the `@data` tag, contains the actual data. It is the file format used by the popular WEKA data mining tool.

MLDs can be downloaded from repositories (see the next section) such as MULAN [1], MEKA [2], LibSVM [3], KEEL [4], and RUMDR [5], each one using a different file format. The differences among multilabel file formats can be grouped according to the following criteria:

- **CSV versus ARFF**: ARFF is the most usual base file format for MLDs. The datasets available at MULAN, MEKA, and KEEL are ARFF files. On the contrary, LibSVM chose to use the simpler CSV file format.
- **Label information**: In order to use an MLD, knowing how many labels there are or which are the names of the attributes acting as labels is essential. MULAN datasets provide the label names in a separate XML file. KEEL datasets include in the ARFF header the set of output attributes. MEKA datasets indicate in the header, along with the name of the relation, the number of labels.
- **Label location**: Although multilabel formats providing label names could locate the attributes acting as labels at any position in the MLD, they usually put them at the end, after all the input attributes. This is the case for MULAN and KEEL. On the other hand, MEKA and LibSVM always arrange the labels at the beginning. Knowing the number of labels, the location allows to get the proper attribute names without needing to include them in the ARFF header or providing an XML file.
- **Sparse versus non-sparse**: There are MLDs that have thousands of input attributes plus thousands of labels. Therefore, each data row (instance) consists of a long sequence of values. Many of them could be zero, since labels can only take two values and the same is also applicable to many input attributes. In these cases, the MLD will be a large array, with thousands of columns and maybe rows, with zeroes in most of its values. To avoid storage and memory wasting, these MLDs are usually stored as sparse data. The rows in a sparse MLD are composed of comma-separated pairs of values. In each pair, the first value indicates the index of the attribute, while the second provides the actual value. In non-sparse MLDs, each row will contain the same number of columns, having the values for each attribute.

When some experiment is going to be conducted using a set of MLDs, the user has to choose between converting all of them to the proper file format, suitable for the tool to be used later to conduct the experiment, or being limited to only use those MLDs which are already available in this file format.

9.2.2 *Multilabel Data Repositories*

When it comes to multilabel data gathering, there are several alternatives to choose from. Custom MLDs can be produced for specific fields where multilabel data are still not available. Alternately, existing MLDs produced by someone else can be obtained from several repositories, as long as they suit the faced task needs. The option to generate these MLDs synthetically, by means of some software tools, is another potential choice. This section will look into the second approach.

Multilabel data repositories provide a convenient way to obtain MLDs that other researchers have built and used in their studies. It is an approach that allows to compare different strategies against the same data. Nonetheless, only full datasets are available some times. Few of these repositories also provide citation information. Therefore, the user usually has to download the MLD, partition it, and search for the proper bibliographic entry.

The following are among the best-known multilabel data repositories. For each one, the file format of the MLDs is also indicated:

- **MULAN**: MULAN [1] is a reference multilabel software tool (it will be further described), and its associated repository [6] is probably the most used resource by researchers in this field. The MLDs are provided in the ARFF format. The labels are usually located at the end of the attribute list, and each MLD is linked to an XML file containing the label names and their hierarchical relationship if it exists. Currently, this repository holds 27 MLDs,[2] some of them with prebuilt partitions.
- **MEKA**: MEKA is a multilabel tool based on WEKA. As MULAN, it brings reference implementations for several methods, as will be shown later. The MEKA repository [2] supplies 15 MLDs. Some of them are the same found in MULAN, but using the MEKA file format. This is also ARFF-based, but the labels always appear at the beginning of the attribute list. There are no separate XML file with label names, but the number of labels in the MLD is indicated in the ARFF header, as a parameter of the relation name.
- **LibSVM**: LibSVM [3] is a popular software library for SVMs. There are many classification algorithms, including some multilabel ones, build upon LibSVM. The associated data repository [7] includes 8 MLDs. In this case, the file format is CSV-based instead of ARFF-based, but the attribute values are given according to the sparse representation previously described. The labels are always put at the beginning of each instance. There is no associated XML file nor any header indicating the number of labels or their names.
- **KEEL**: Unlike MULAN and MEKA, KEEL [4] is a general-purpose data mining application, similar to WEKA. This software tool has an extensive data repository with different kinds of datasets, including 16 MLDs [8]. The file format is ARFF-based, indicating in the attribute list which features act as labels.
- **RUMDR**: The *R Universal Multilabel Dataset Repository* [5] is associated with an R package named mldr.datasets [9] (it will be portrayed in the following subsection). Currently, this is the most extensive multilabel data repository, providing more than 60 MLDs. These can be directly downloaded from the repository in R file format; thus, they are designed to be loaded from R. The functions provided by the package allow to export them from this native format to several ones, including MULAN, MEKA, LibSVM, and CSV.

[2]The number of MLDs provided by each repository has been checked as of April 2016.

- **Extreme Classification Repository**: This repository [10] only provides 9 MLDs, all of them sharing a specific characteristic: They have a huge list of input features, output labels, or both. There are MLDs with more than one million attributes, aimed to test solutions for extreme multilabel classification. The file format is a combination of CSV for label indexes, always at the beginning, and sparse representation for input attributes, with a one-line header indicating the number of instances, features, and labels.

These data repositories offer an immediate solution to the user which needs some MLDs, as long as the file format is appropriate and a tool to partition the data are at hand. However, this is not always the case. Depending on the tool being used to conduct the experiments, the MLDs may have to be transformed to other file format and properly partitioned. Some of these needs can be addressed by means of the software package described below.

9.2.3 The mldr.datasets Package

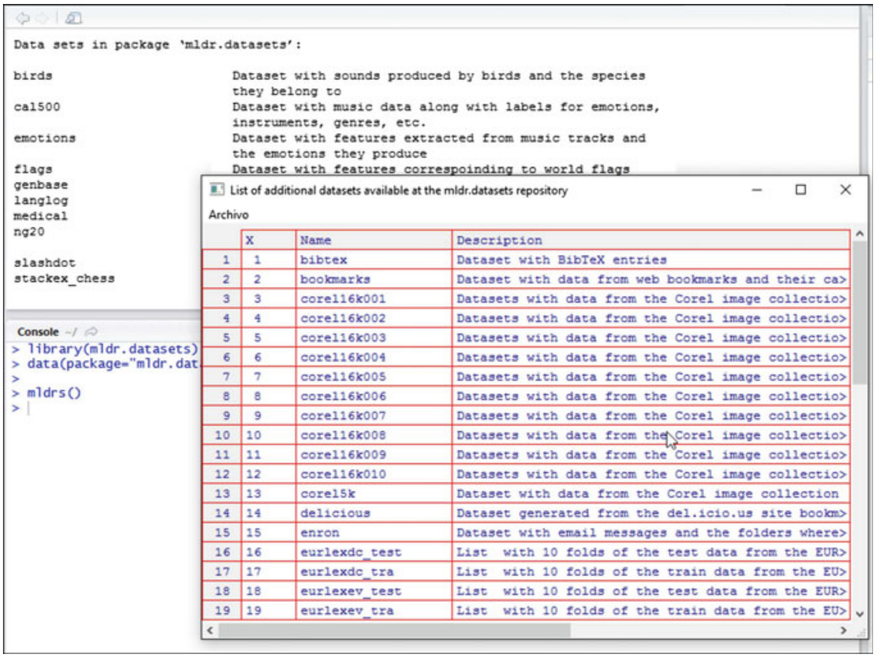
When it comes to data exploration, analysis, and mining, R [11] is a very popular tool/language due to its extensive package list. One of these packages, named mldr.datasets [9], is specifically designed to aid the user in the tasks of obtaining, partitioning, converting, and exporting multilabel datasets. mldr.datasets is tied to the aforementioned RUMDR repository.

The mldr.datasets is available at CRAN (*Comprehensive R Archive Network*), the distributed network providing most R packages. Therefore, it can be downloaded and installed from any up-to-date R version by simply issuing the console the `install.packages("mldr.datasets")` command. Once installed, the package has to be loaded into memory with the usual `library(mldr.datasets)` command. This will bring to the R workspace ten medium-sized MLDs, along with the functions needed to access many more and to manipulate them. The preloaded MLDs are those stored in the `data` folder of the RUMDR repository.

In the following, how to use the mldr.datasets to accomplish some basic tasks over MLDs is explained, assuming the package is already installed and loaded into memory.

9.2.3.1 Loading Available MLDs

After loading the package, the user can know which MLDs are available using the usual `data()` function, passing the name of the package as parameter. These are the MLDs brought to memory by loading the package, but there are many more available on the RUMDR repository. A list of these is returned by the `mldrs()` function, as shown in Fig. 9.1.



```
Data sets in package 'mldr.datasets':

birds                         Dataset with sounds produced by birds and the species
                              they belong to
cal500                        Dataset with music data along with labels for emotions,
                              instruments, genres, etc.
emotions                      Dataset with features extracted from music tracks and
                              the emotions they produce
flags                         Dataset with features correspoinding to world flags
genbase
langlog
medical
ng20

slashdot
stackex_chess
```

Console ~/

```
> library(mldr.datasets)
> data(package="mldr.dat
>
> mldrs()
>
```

List of additional datasets available at the mldr.datasets repository

Archivo

	X	Name	Description
1	1	bibtex	Dataset with BibTeX entries
2	2	bookmarks	Dataset with data from web bookmarks and their ca>
3	3	corel16k001	Datasets with data from the Corel image collectio>
4	4	corel16k002	Datasets with data from the Corel image collectio>
5	5	corel16k003	Datasets with data from the Corel image collectio>
6	6	corel16k004	Datasets with data from the Corel image collectio>
7	7	corel16k005	Datasets with data from the Corel image collectio>
8	8	corel16k006	Datasets with data from the Corel image collectio>
9	9	corel16k007	Datasets with data from the Corel image collectio>
10	10	corel16k008	Datasets with data from the Corel image collectio>
11	11	corel16k009	Datasets with data from the Corel image collectio>
12	12	corel16k010	Datasets with data from the Corel image collectio>
13	13	corel5k	Dataset with data from the Corel image collection
14	14	delicious	Dataset generated from the del.icio.us site bookm>
15	15	enron	Dataset with email messages and the folders where>
16	16	eurlexdc_test	List with 10 folds of the test data from the EUR>
17	17	eurlexdc_tra	List with 10 folds of the train data from the EU>
18	18	eurlexev_test	List with 10 folds of the test data from the EUR>
19	19	eurlexev_tra	List with 10 folds of the train data from the EU>

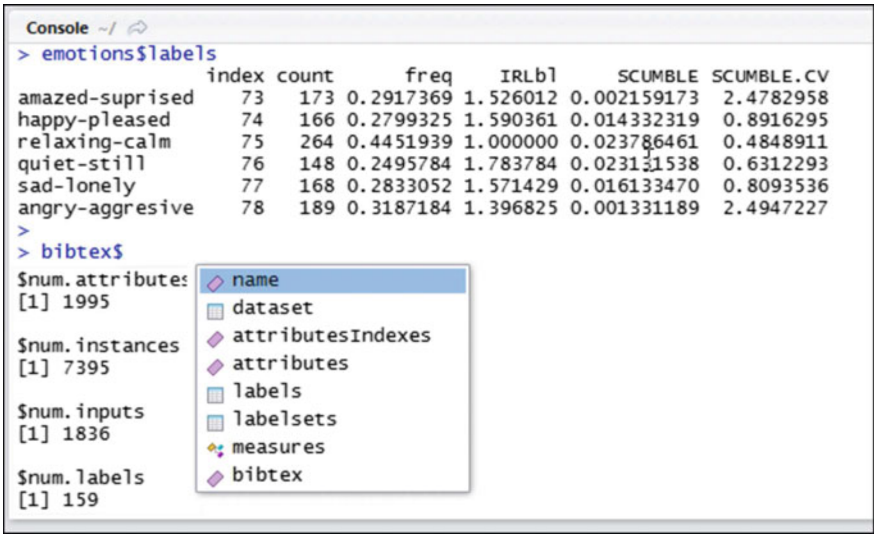
Fig. 9.1 Looking at the available MLDs in the mldr.datasets package

To load any of the available MLDs, all the user has to do is typing in the R console its name followed by empty parentheses. The package will check whether the requested MLD is locally available into the user's computer, loading it into memory if this is the case. On the contrary, the MLD will be automatically downloaded from the RUMDR repository, stored in the local machine, and then loaded into memory, without needing any user intervention.

9.2.3.2 Exploring Loaded MLDs

The MLDs supplied by the mldr.datasets package are `mldr` objects. It is the object format defined by the mldr package, further addressed in this chapter. These objects have several members containing data helpful to explore the MLD structure, such as the names and frequencies of labels and labelsets and domains of input attributes. To access any of these members, the `dataset$member` syntax will be used, as depicted in Fig. 9.2.

The multilabel data are stored into the `dataset` member. This is a standard R `data.frame`; therefore, the usual R syntax to access any of its columns and rows is used. The `measures()` function returns a list of characterization metrics, such as the number of instances, features, and labels, imbalance levels, and theoretical complexity level.



```
> emotions$labels
                index count      freq    IRLbl     SCUMBLE SCUMBLE.CV
amazed-suprised    73   173 0.2917369 1.526012 0.002159173  2.4782958
happy-pleased      74   166 0.2799325 1.590361 0.014332319  0.8916295
relaxing-calm      75   264 0.4451939 1.000000 0.023786461  0.4848911
quiet-still        76   148 0.2495784 1.783784 0.023131538  0.6312293
sad-lonely         77   168 0.2833052 1.571429 0.016133470  0.8093536
angry-aggresive    78   189 0.3187184 1.396825 0.001331189  2.4947227
>
> bibtex$
$num.attributes
[1] 1995

$num.instances
[1] 7395

$num.inputs
[1] 1836

$num.labels
[1] 159
```

Fig. 9.2 The mldr objects have several members with disparate information

9.2.3.3 Obtaining Citation Information

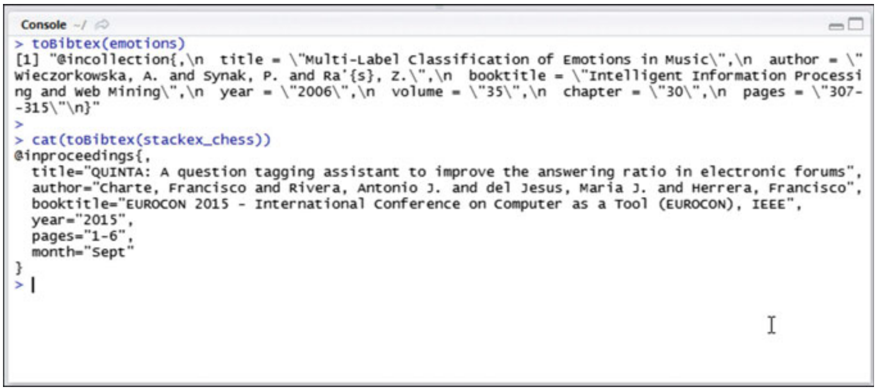
When an MLD produced by any third party is going to be used in a new study, including the proper citation to give the original author, the correct attribution is mandatory. Obtaining the precise citation information is not always easy. The mldr.datasets package includes a custom version of the R's `toBibtex()` function whose goal is to provide the BibTeX entry associated with any `mldr` object.

The value returned by the `toBibtex()` function is properly formatted to copy it to the clipboard and then pasting it in the user's BibTeX editor. As demonstrated in Fig. 9.3, it can also be printed to the console.

9.2.3.4 Partitioning the MLDs

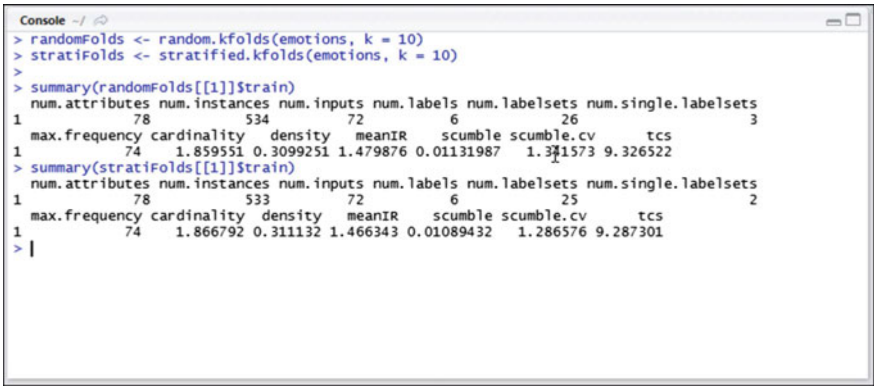
Although partitioned MLDs can be obtained from some repositories, this is not always the case. Furthermore, the number of partitions or their proportions could be not adequate for the user needs. The mldr.datasets package contributes two functions, named `random.kfolds()` and `stratified.kfolds()`, whose goal is to partition any `mldr` object into the number of desired parts. The difference between these two functions relies on the approach followed to choose the instances included in each partition. The former does it randomly, while the latter stratifies the data trying to balance the label distribution among partitions.

Both functions need the `mldr` object to be partitioned as their first argument. Additionally, they can take two more parameters specifying the number of folds, it



```
Console ~/
> toBibtex(emotions)
[1] "@incollection{,\n  title = \"Multi-Label Classification of Emotions in Music\",\n  author = \"
Wieczorkowska, A. and Synak, P. and Ra'{s}, Z.\",\n  booktitle = \"Intelligent Information Processi
ng and Web Mining\",\n  year = \"2006\",\n  volume = \"35\",\n  chapter = \"30\",\n  pages = \"307-
-315\"\n}"
>
> cat(toBibtex(stackex_chess))
@inproceedings{,
  title="QUINTA: A question tagging assistant to improve the answering ratio in electronic forums",
  author="Charte, Francisco and Rivera, Antonio J. and del Jesus, Maria J. and Herrera, Francisco",
  booktitle="EUROCON 2015 - International Conference on Computer as a Tool (EUROCON), IEEE",
  year="2015",
  pages="1-6",
  month="Sept"
}
> |
```

Fig. 9.3 Obtaining the BibTeX entry to cite the MLD



```
Console ~/
> randomFolds <- random.kfolds(emotions, k = 10)
> stratiFolds <- stratified.kfolds(emotions, k = 10)
>
> summary(randomFolds[[1]]$train)
  num.attributes num.instances num.inputs num.labels num.labelsets num.single.labelsets
1             78           534         72          6            26                    3
  max.frequency cardinality   density   meanIR    scumble scumble.cv      tcs
1            74    1.859551 0.3099251 1.479876 0.01131987   1.341573 9.326522
> summary(stratiFolds[[1]]$train)
  num.attributes num.instances num.inputs num.labels num.labelsets num.single.labelsets
1             78           533         72          6            25                    2
  max.frequency cardinality  density   meanIR    scumble scumble.cv      tcs
1            74    1.866792 0.311132 1.466343 0.01089432   1.286576 9.287301
> |
```

Fig. 9.4 An MLD being partitioned using random and stratified approaches

is 5 by default, and the seed for the random generator. The result returned by these functions is a list containing as many elements as folds have been indicated. Each one of these elements is made up of two members, called `train` and `test`, with the corresponding data partitions.

The example code shown in Fig. 9.4 demonstrates how to use these two functions, as well as how to access the training partition of the first fold. The `summary()` function prints a summary of characterization metrics, allowing to compare how the different partitioning approach has influenced the obtained partition.

9.2.3.5 Converting MLDs to Other Formats

Although R is a tool from which the MLDs provided by mldr.datasets can be used with disparate machine learning algorithms, currently software packages such as

MULAN and MEKA are preferred, due to the large prebuilt set of MLC algorithms they incorporate.

The file format of MLDs provided by RUMDR is the R native object format. Nonetheless, once they have been loaded into R, it is possible to convert them to several other file formats. For doing so, the `write.mldr()` function of the mldr.datasets package has to be called.

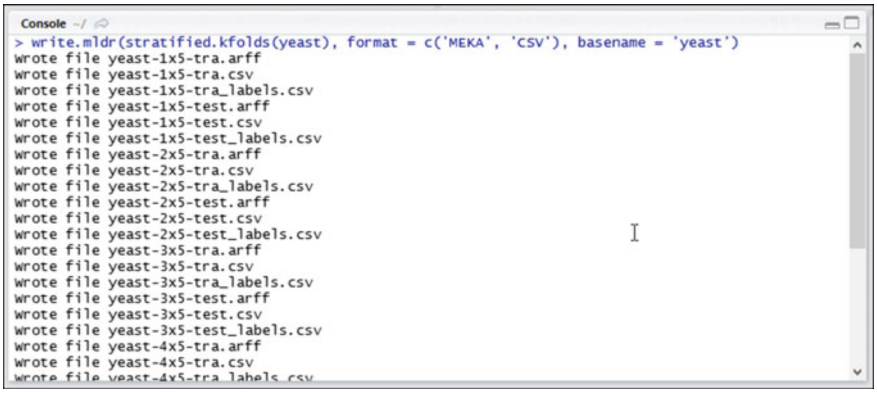
The `write.mldr()` function accepts as first argument an `mldr` object, containing the MLD to be written to a file. It is also able to deal with a list as the one returned by the partitioning functions described above, writing each training and test fold to a different file. This is the only mandatory parameter, and the remaining ones take default values.

As second argument, the `write.mldr()` function takes a vector of strings stating the file formats the data are going to be exported to. Valid formats are `MULAN`, `MEKA`, `KEEL`, `LIBSVM`, and `CSV`. The default value is `c("MULAN," "MEKA")`, being these the two most popular multilabel file formats. If the MULAN format is chosen, the function will also write the corresponding XML file for the MLD. For the CSV format, an additional CSV file containing a list with label names is also created.

The third parameter has to be a boolean value, indicating if sparse representation has to be used to write the data. By default, it takes the `FALSE` value, so the non-sparse format is chosen unless otherwise specified.

Lastly, the fourth argument sets the base filename the `write.mldr()` function will use to name the written files. This filename will be followed by a set of numbers, stating the fold and total number of folds, if the first parameter is a list of `mldr` objects.

The `write.mldr()` function can be combined with the previously described partitioning functions, as shown in Fig. 9.5. In this example, the yeast MLD is being partitioned into fivefolds, and then, the resulting partitions are written in MEKA and CSV file formats.



```
> write.mldr(stratified.kfolds(yeast), format = c('MEKA', 'CSV'), basename = 'yeast')
wrote file yeast-1x5-tra.arff
wrote file yeast-1x5-tra.csv
wrote file yeast-1x5-tra_labels.csv
wrote file yeast-1x5-test.arff
wrote file yeast-1x5-test.csv
wrote file yeast-1x5-test_labels.csv
wrote file yeast-2x5-tra.arff
wrote file yeast-2x5-tra.csv
wrote file yeast-2x5-tra_labels.csv
wrote file yeast-2x5-test.arff
wrote file yeast-2x5-test.csv
wrote file yeast-2x5-test_labels.csv
wrote file yeast-3x5-tra.arff
wrote file yeast-3x5-tra.csv
wrote file yeast-3x5-tra_labels.csv
wrote file yeast-3x5-test.arff
wrote file yeast-3x5-test.csv
wrote file yeast-3x5-test_labels.csv
wrote file yeast-4x5-tra.arff
wrote file yeast-4x5-tra.csv
wrote file yeast-4x5-tra_labels.csv
```

Fig. 9.5 Partitioning and exporting an MLD to MEKA and CSV file formats

Since due to its link to the RUMDR repository, most of the MLDs publicly available nowadays can be downloaded and then properly cited, partitioned, and exported to the common file formats, the mldr.datasets package can be the most convenient way of dealing with existent MLDs.

9.2.4 Generating Synthetic MLDs

The MLDs provided in the previous repositories have been generated from real data coming from different domains, as was explained in Chap. 3. However, sometimes having MLDs with very definite traits would be desirable, for instance while analyzing the behavior of algorithms aimed to solve a very specific problem. In order to produce these kinds of data, an appropriate algorithm has to be designed, usually including a mathematical method to correlate the inputs with the outputs.

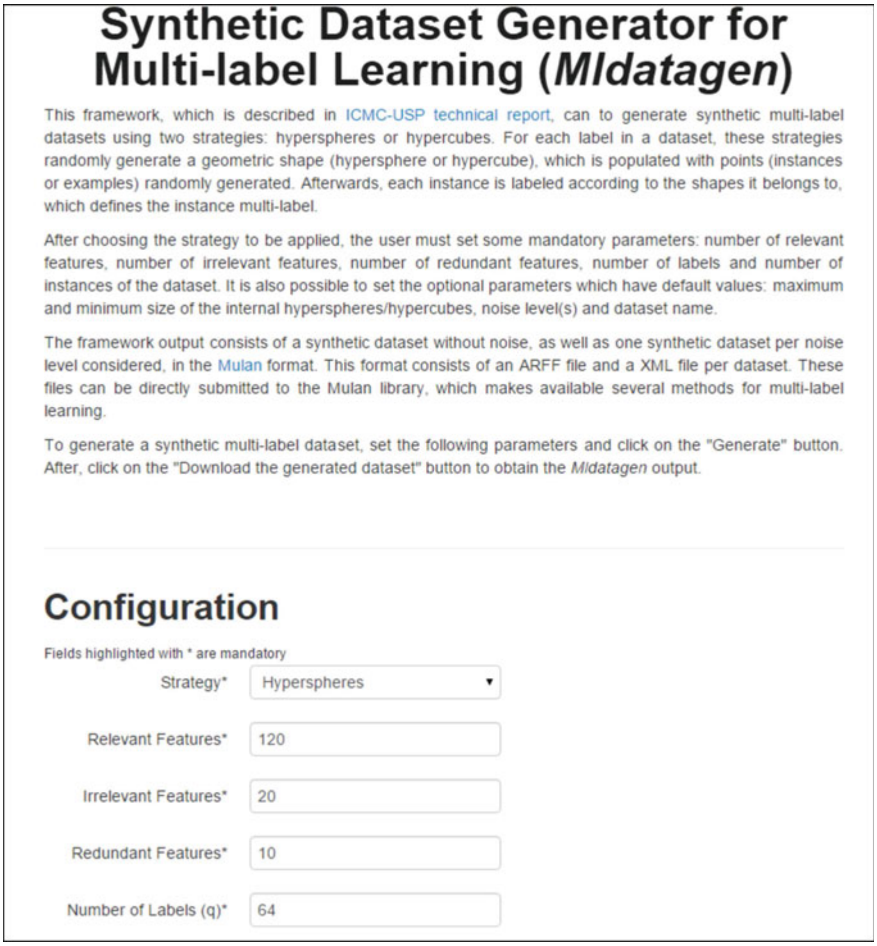
In [12], such an algorithm, which offers two different approaches based on hypercubes and hyperspheres, is presented. The associated synthetic dataset generator for multilabel learning is available online, as a Web application (see Fig. 9.6). It allows the user to choose among the two strategies for generating instances, as well as to indicate how many features, labels, and instances the MLD will have, the noise level to add to the data, and other handful of configuration parameters. Once all of them have been set, the resulting dataset characteristics are shown and the file can be downloaded.

Mldatagen is a quite generic multilabel data generator. In the multilabel literature, some authors have created their own synthetic MLDs following more ad hoc approaches, adapted to suit specific goals. References to several of them can be found in [13].

Multilabel datasets are scattered through a collection of data repositories using disparate file formats. Once we know how to get these datasets and which are the file formats they use, a software tool such as the described mldr.datasets package is all we need to cite, partition, and export all MLDs in the proper format for the experiments we intend to do.

9.3 Exploratory Analysis of MLDs

The Web repositories where the MLDs are downloaded from, such as the previously mentioned MULAN and MEKA, usually also supply some basic information about the datasets. The number of instances, input attributes, and labels, along with label cardinality and sometimes label density, are common metadata. However, before using these MLDs to conduct some experiments, most users will demand additional details. Those will be generally obtained by means of exploratory analysis tasks, including summarizing at different levels and visualizing the data.



Synthetic Dataset Generator for Multi-label Learning (*Mldatagen*)

This framework, which is described in ICMC-USP technical report, can to generate synthetic multi-label datasets using two strategies: hyperspheres or hypercubes. For each label in a dataset, these strategies randomly generate a geometric shape (hypersphere or hypercube), which is populated with points (instances or examples) randomly generated. Afterwards, each instance is labeled according to the shapes it belongs to, which defines the instance multi-label.

After choosing the strategy to be applied, the user must set some mandatory parameters: number of relevant features, number of irrelevant features, number of redundant features, number of labels and number of instances of the dataset. It is also possible to set the optional parameters which have default values: maximum and minimum size of the internal hyperspheres/hypercubes, noise level(s) and dataset name.

The framework output consists of a synthetic dataset without noise, as well as one synthetic dataset per noise level considered, in the Mulan format. This format consists of an ARFF file and a XML file per dataset. These files can be directly submitted to the Mulan library, which makes available several methods for multi-label learning.

To generate a synthetic multi-label dataset, set the following parameters and click on the "Generate" button. After, click on the "Download the generated dataset" button to obtain the *Mldatagen* output.

Configuration

Fields highlighted with * are mandatory

Strategy*	Hyperspheres
Relevant Features*	120
Irrelevant Features*	20
Redundant Features*	10
Number of Labels (q)*	64

Fig. 9.6 The Mldatagen tool allows creating custom synthetic MLDs

As long as the structure of each multilabel file format is adequately decoded, any generic data manipulation tool could be used to explore the MLDs. Nevertheless, this section is focused on interactive software tools specifically built to work with multilabel data. Two particular tools in this category are described below, MEKA and the mldr package. Both provide some EDA (*Exploratory Data Analysis*) functionality.

9.3.1 MEKA

MEKA is a software tool built upon WEKA, and it brings a similar user interface to this popular application but with multilabel capabilities. MEKA is developed in Java; therefore, to use it, the first requirement to meet is having the Java Runtime Envi-

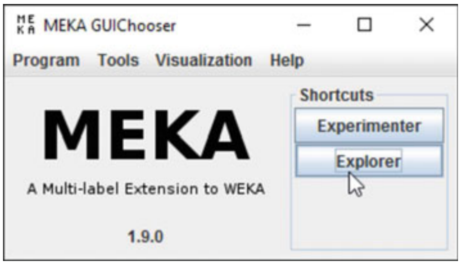


Fig. 9.7 The MEKA main window allows the user to open several tools. The Explorer lets open and explore MEKA datasets, as well as to interactively perform experiments with them

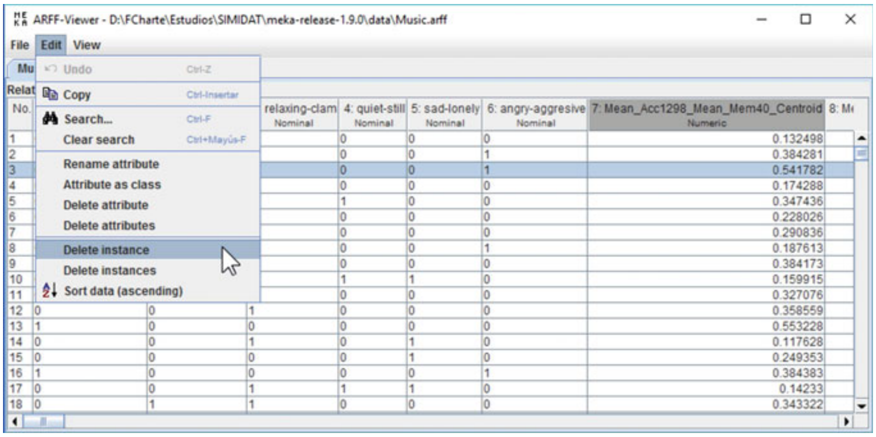
ronment (JRE) installed in the system. Then, the most recent version of MEKA can be downloaded from https://adams.cms.waikato.ac.nz/snapshots/meka. The installation process is just extracting the files of the compressed file to a folder.

In addition to the software itself, the MEKA compressed file also includes a PDF document with a tutorial and some example data files. Two scripts, one for Windows (`run.bat`) and another one for UNIX-based systems `run.sh`, aimed to ease the launch of the software are provided as well.

Launching MEKA through the proper script will open the program's main window (see Fig. 9.7). The options in the **Tools** menu run the essential MEKA tools. Some of them are depicted below.

9.3.1.1 The ARFF Viewer

MEKA includes a viewer able to open any ARFF file, enumerating its attributes, including the labels, and the values assigned to them. Once a dataset has been loaded, its content is shown in a grid like the one in Fig. 9.8. The **Properties** option in the **File**



No.			relaxing-clam Nominal	4: quiet-still Nominal	5: sad-lonely Nominal	6: angry-aggresive Nominal	7: Mean_Acc1298_Mean_Mem40_Centroid Numeric	8: M
1				0	0	0	0.132498	
2				0	0	1	0.384281	
3				0	0	1	0.541782	
4				0	0	0	0.174288	
5				1	0	0	0.347436	
6				0	0	0	0.228026	
7				0	0	0	0.290836	
8				0	0	1	0.187613	
9				0	0	0	0.384173	
10				1	1	0	0.159915	
11				0	0	0	0.327076	
12	0	0	1	0	0	0	0.358559	
13	1	0	0	0	0	0	0.553228	
14	0	0	1	0	1	0	0.117628	
15	0	0	0	0	1	0	0.249353	
16	1	0	0	0	0	1	0.384383	
17	0	0	1	1	1	0	0.14233	
18	0	1	1	0	0	0	0.343322	

Fig. 9.8 The MEKA ARFF viewer allows viewing and editing any ARFF dataset contents

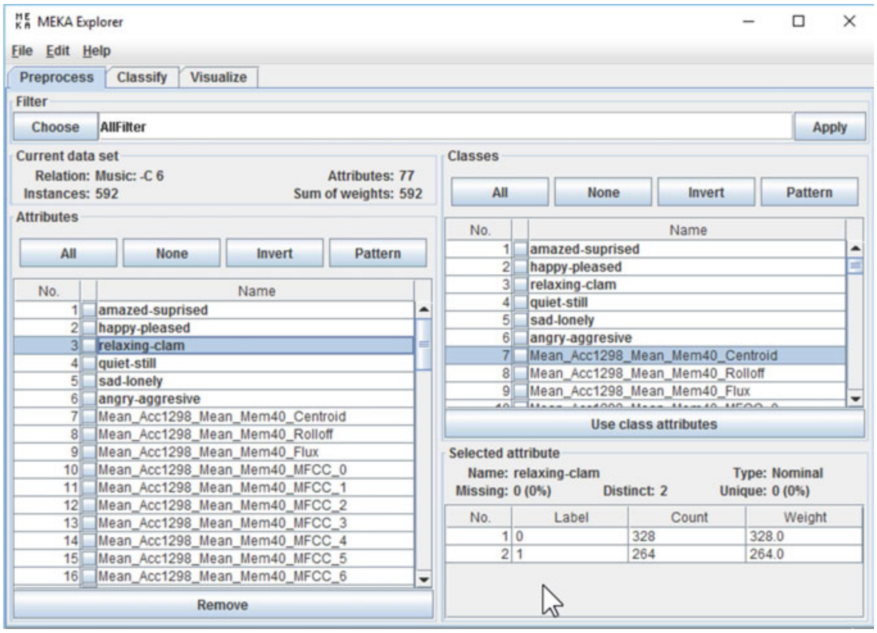
menu will report the number of instances, attributes, and classes. This is a generic ARFF tool, so it is not aware of the multilabel nature of the data.

Actually, this tool is also an editor, since the values can be changed, instances and attributes can be removed, and the order of the data samples can be modified. Any change will be lost unless the data in memory are saved, usually with a different filename.

9.3.1.2 Attribute Exploration

The other exploration tool embedded in MEKA is the MEKA Explorer, accessible through the **Tool→Explorer** option in the main window. This tool is aimed to interactively test preprocessing and classification algorithms, but it also has some exploration capabilities.

After loading an MLD, the **Preprocess** page will show two lists containing all the attributes in the dataset (see Fig. 9.9). At the top of the left list, there is a summary with the number of instances and attributes. If the MLD has been loaded from a file in MEKA format, the program will automatically detect which attributes are labels. These will be highlighted in bold and always will be at the beginning in both lists. Loading MLDs from other ARFF-based file formats is allowed, but the program will be not able to identify the labels. The user has to mark them in the right list and then



AllFilter

Relation: Music: -C 6
Attributes: 77
Instances: 592
Sum of weights: 592

No.		Name
1	☐	amazed-suprised
2	☐	happy-pleased
3	☐	relaxing-clam
4	☐	quiet-still
5	☐	sad-lonely
6	☐	angry-aggresive
7	☐	Mean_Acc1298_Mean_Mem40_Centroid
8	☐	Mean_Acc1298_Mean_Mem40_Rolloff
9	☐	Mean_Acc1298_Mean_Mem40_Flux
10	☐	Mean_Acc1298_Mean_Mem40_MFCC_0
11	☐	Mean_Acc1298_Mean_Mem40_MFCC_1
12	☐	Mean_Acc1298_Mean_Mem40_MFCC_2
13	☐	Mean_Acc1298_Mean_Mem40_MFCC_3
14	☐	Mean_Acc1298_Mean_Mem40_MFCC_4
15	☐	Mean_Acc1298_Mean_Mem40_MFCC_5
16	☐	Mean_Acc1298_Mean_Mem40_MFCC_6

No.		Name
1	☐	amazed-suprised
2	☐	happy-pleased
3	☐	relaxing-clam
4	☐	quiet-still
5	☐	sad-lonely
6	☐	angry-aggresive
7	☐	Mean_Acc1298_Mean_Mem40_Centroid
8	☐	Mean_Acc1298_Mean_Mem40_Rolloff
9	☐	Mean_Acc1298_Mean_Mem40_Flux

Name: relaxing-clam
Type: Nominal
Missing: 0 (0%)
Distinct: 2
Unique: 0 (0%)

No.	Label	Count	Weight
1	0	328	328.0
2	1	264	264.0

Fig. 9.9 The Preprocess page allows to explore the attributes and labels in the MLD

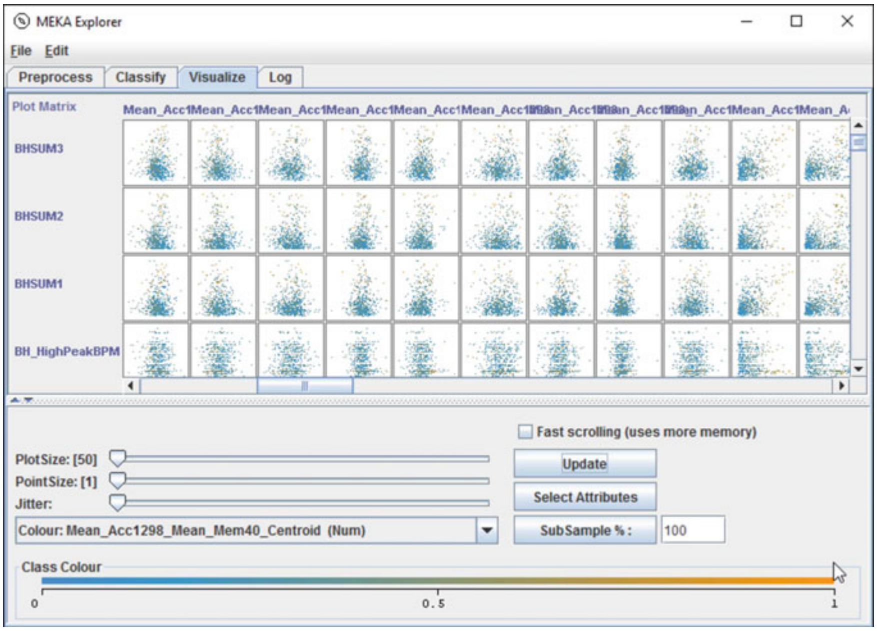


Fig. 9.10 Pairwise scatter plots for the attributes in the MLD

click the **Use class attributes**. By selecting any attribute in the left list, a summary of its domain will be displayed below the right list.

The **Visualize** page in this tool provides a matrix of scatter plots (see Fig. 9.10), each one showing the relationship between a pair of attributes. The controls at the bottom can be used to configure these plots, change their size, sample the instances appearing in them, etc. By clicking any of the plots, a bigger version will be opened in a separate window, with additional controls to customize it. The plot can be saved to a file.

Basically, these are all the exploratory analysis functions offered by MEKA. As can be seen, they are mostly applicable to both traditional and multilabel datasets. There is a lack of information about how the labels are distributed and correlated, which labelsets and other specific multilabel measurements exist.

9.3.2 *The mldr Package*

Unlike MEKA, which can be considered as a general-purpose multilabel software, the R mldr [14] package has been precisely developed as a multilabel exploratory analysis tool. This package is also available at CRAN, like the previously described mldr.datasets package, so it can be installed in the same way. There are not dependencies between the two packages. That means that mldr can be used without installing mldr.datasets and vice versa. Nonetheless, mldr can take advantage of having all the MLDs included in the mldr.datasets package, as well as their functions to partition and export them.

The mldr package is able to load MLDs from MULAN and MEKA file formats, as well as to generate new MLDs on the fly from data synthetically generated or provided by the user in a `data.frame`. The package defines a custom representation for multilabel data. The MLDs are R S3 objects with `mldr` class. This is the same representation used by the MLDs in the mldr.datasets package, and hence, these datasets are compatible with mldr.

Inside the mldr package, a plethora of characterization metrics are computed, along with a set of functions aimed to ease the analysis of multilabel data. In addition, the package provides a Web-based user interface to speed up exploratory tasks, including specific graphic representations of the data.

In this section, the procedures to accomplish different exploratory duties using the mldr package are explained. It is assumed the user has installed the package, by issuing the `install.packages("mldr")` command at the R console, and it is loaded into memory, by typing the `library(mldr)` command.

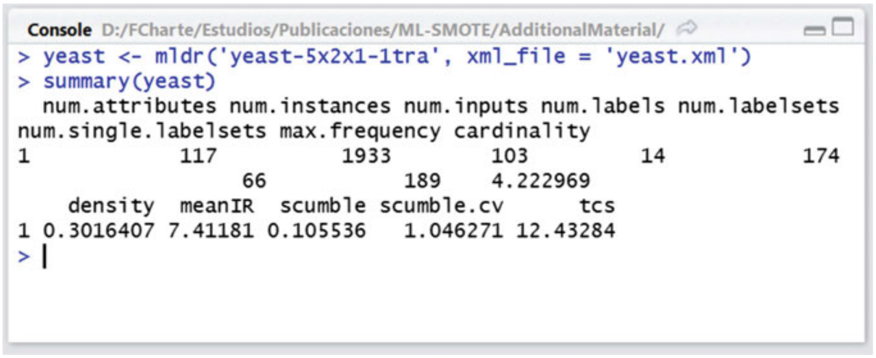
9.3.2.1 Loading and Creating MLDs

After loading the mldr package, three `mldr` objects will be already available. These correspond to the `birds`, `emotions`, and `genbase` MLDs. If the mldr.datasets is also loaded, all the MLDs provided by it will be accessible as well. To load any other MLD, assuming it is stored in an ARFF file using the MEKA or MULAN formats, the `mldr()` function has to be called. The only mandatory argument is the filename without extension. The function will assume the MLD is in MULAN file format and the existence of an XML file with the same name. Additional parameters can be supplied to change this default behavior, stating the XML filename, the number, indexes, or names of the attributes acting as labels if there are not an XML file available nor this information is provided in the ARFF header, etc.

The `mldr()` function always checks whether the mldr.datasets package is installed in the system. If this is the case, the function entrusts the loading of the MLD to the proper mldr.datasets function. To avoid this functionality, forcing the loading from a local file, the `force_read_from_file` parameter has to be included in the call to `mldr()`, assigning it the `TRUE` value.

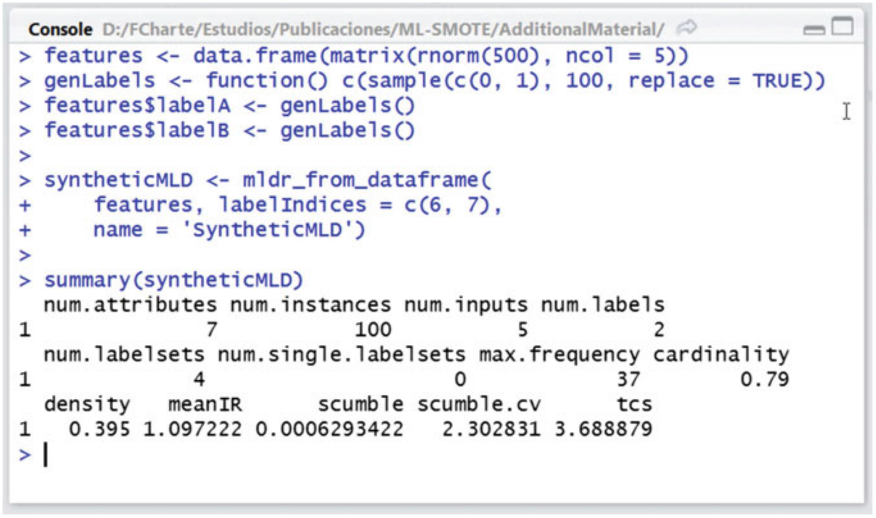
In Fig. 9.11, a use case of the `mldr()` function is shown. In this example, the name of the XML file does not coincide with the ARFF filename, so it is explicitly set by means of the `xml_file` parameter. Once the MLD has been loaded, the object can be queried to obtain some dataset traits as demonstrated.

Furthermore, new MLDs can be created from any existing data, whether it is from a real domain or produced by any formula. This functionality, provided by the `mldr_from_dataframe()` function, allows creating synthetic MLDs. This function needs as inputs a `data.frame` containing all the features, a vector stating


Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/

```
> yeast <- mldr('yeast-5x2x1-1tra', xml_file = 'yeast.xml')
> summary(yeast)
  num.attributes num.instances num.inputs num.labels num.labelsets
num.single.labelsets max.frequency cardinality
1            117          1933        103         14            174
                  66          189    4.222969
    density  meanIR  scumble scumble.cv      tcs
1 0.3016407 7.41181 0.105536   1.046271 12.43284
> |
```

Fig. 9.11 Loading an MLD from an external ARFF file


Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/

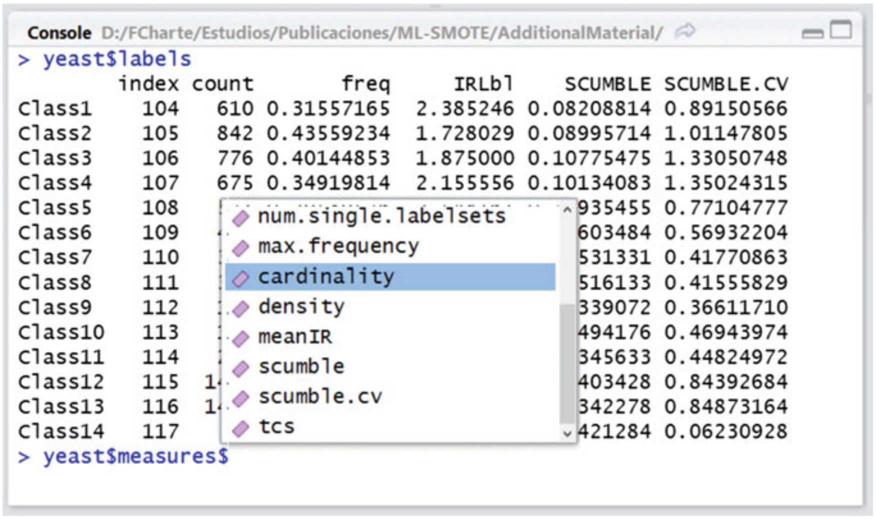
```
> features <- data.frame(matrix(rnorm(500), ncol = 5))
> genLabels <- function() c(sample(c(0, 1), 100, replace = TRUE))
> features$labelA <- genLabels()
> features$labelB <- genLabels()
>
> syntheticMLD <- mldr_from_dataframe(
+     features, labelIndices = c(6, 7),
+     name = 'SyntheticMLD')
>
> summary(syntheticMLD)
  num.attributes num.instances num.inputs num.labels
1              7           100          5          2
  num.labelsets num.single.labelsets max.frequency cardinality
1             4                    0            37        0.79
  density   meanIR      scumble scumble.cv      tcs
1   0.395 1.097222 0.0006293422   2.302831 3.688879
> |
```

Fig. 9.12 New MLDs can be generated on the fly from any formula

which ones of them will be labels, and optionally a name to assign to the new MLD. The result, as is shown in Fig. 9.12, is an `mldr` object that can be used as any other MLD.

9.3.2.2 Querying Data Characterization Metrics

Independently of the origin of the MLD's data, all `mldr` objects have the same structure and can be processed with the same set of functions. Many of these are aimed to compute and provide several characterization metrics. The supplied metrics can be grouped into four categories:



```
Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/
> yeast$labels
        index count       freq    IRLbl    SCUMBLE SCUMBLE.CV
Class1    104   610 0.31557165 2.385246 0.08208814 0.89150566
Class2    105   842 0.43559234 1.728029 0.08995714 1.01147805
Class3    106   776 0.40144853 1.875000 0.10775475 1.33050748
Class4    107   675 0.34919814 2.155556 0.10134083 1.35024315
Class5    108                                 935455 0.77104777
Class6    109                                 603484 0.56932204
Class7    110                                 531331 0.41770863
Class8    111                                 516133 0.41555829
Class9    112                                 339072 0.36611710
Class10   113                                 494176 0.46943974
Class11   114                                 345633 0.44824972
Class12   115   1                             403428 0.84392684
Class13   116   1                             342278 0.84873164
Class14   117                                 421284 0.06230928
> yeast$measures$
```

Fig. 9.13 A large set of traits are computed for each MLD

- **Basic traits**: The number of instances, number of input attributes, number of output labels, and number of labelsets are in this group. All of them can be queried with the syntax `mldrobject$measures$num.XXX`. The `summary()` function also returns this information.
- **Label distribution data**: Metrics such as label cardinality, label density, and the frequency of each individual label are available through the `measures` and `labels` members, as demonstrated in Fig. 9.13.
- **Label relationship metrics**: The relationships among the labels in the dataset can be inspected through metrics such as the total number of unique labelsets (`measures$num.labelsets`) in the MLD, the number of single labelsets (`measures$num.single.labelsets`), and analyzing the values for the global and by label *SCUMBLE* measures (`measures$scumble` and `labels$SCUMBLE`).
- **Metrics related to label imbalance**: The individual imbalance level for each label is provided by the `labels$IRLbl` member. The average imbalance level for the MLD can be obtained through the `measures$meanIR` member.

Additional information about the MLD is provided in the members of the `mldr` object such as `labelsets` (signature and counter of each labelset in the MLD), `attributes` (name and domain of each attribute), and `measures$tcs` (the theoretical complexity score of the MLD). All of them are properly documented in the electronic help included in the package.

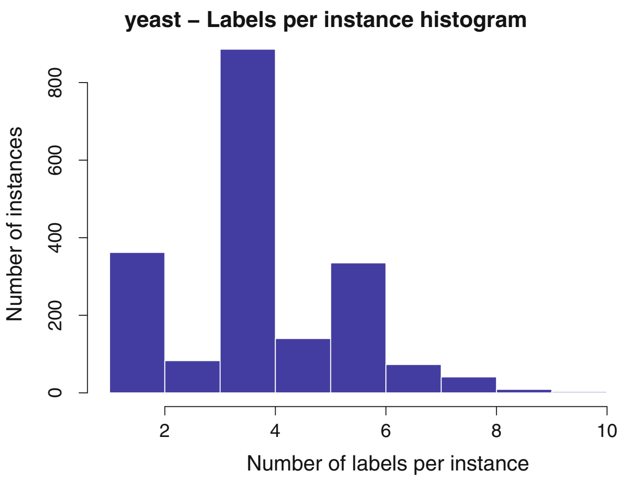


Fig. 9.14 Number of labels per instance histogram

9.3.2.3 mldr Custom Plots

The mldr package delivers a custom `plot()` function for `mldr` objects, able to produce seven specific plots from the data contained in those objects. The arguments to this function are usually two, the `mldr` object to analyze and the `type` parameter specifying what type of plot is desired. Some kinds of visualizations accept additional parameters, for instance to restrict the set of plotted labels.

Three of the custom plots are histograms designed to depict how labels and labelsets are distributed among the instances. The one shown in Fig. 9.14 is produced by the `plot(yeast, type = "CH")` call, showing the distribution of label cardinality among the instances. The `yeast` dataset has a *Card* value of 4.223. That most instances have four labels can be observed in this plot.

Two more of the available types, `"LB"` and `"LSB,"` are bar plots depicting how many instances each label and labelset appear. This way, the differences between frequencies of labels and labelsets can be explored. For instance, Fig. 9.15 shows that there are two majority labels (`Class12` and `Class13`) and several minority labels (`Class14` and `Class9` to `Class11`).

Another kind of plot is denoted as `"AT."` It is a regular pie chart showing the proportion of each type of features, continuous, nominal, and labels. The seventh visualization option is the one generated by default when the `type` argument is not provided. It is a circular plot like the one described in Chap. 8 (see Fig. 9.16), illustrating how the labels interact among them. This kind of plot, as well as those presenting label frequencies, accepts the optional parameters `labelCount` and `labelIndices`, whose goal is to restrict the set of drawn labels. All plot types also take other optional arguments, such as `title` to set the title of the plot and `col` to set the color, or the standard graphical parameters usually given to R functions such as `barplot()` and `hist()`.

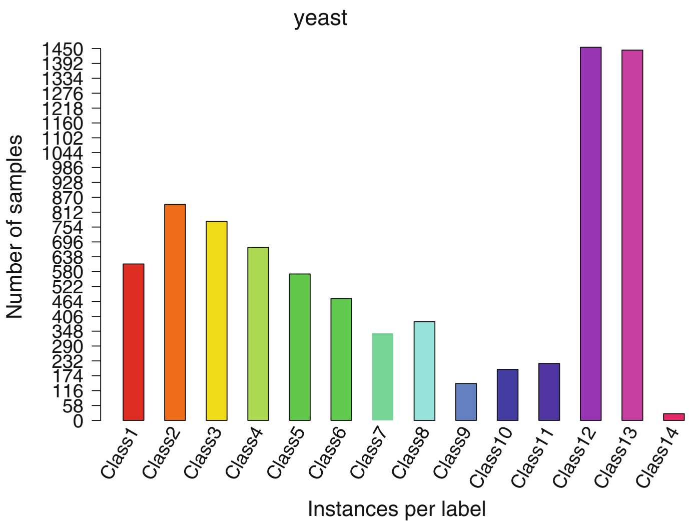


Fig. 9.15 The differences among label frequencies are easily inferred in this bar plot

9.3.2.4 Automated Reports

The mldr package includes some functions able to generate reports by means of automated analysis of the label measurements. The functions in charge of producing these reports only need the `mldr` object to be analyzed as argument. The resulting report is printed to the console.

With the `labelInteractions()` function, an analysis of which labels are in minority and how they interact with others is generated. The reported result includes the indexes of the minority labels, at the beginning, and for each one of them the list of labels they have interactions with, along with the number of samples in which they appear together (see Fig. 9.17).

The second report is produced by the `concurrenceReport()` function. As its name suggests, the report analyzes the concurrence among labels, but in a more elaborated way than the `labelInteractions()` function. As is shown in Fig. 9.18, the report includes the global *SCUMBLE* measurement and its CV, as well as *SCUMBLE* values for each individual label. For each minority label, a list of the labels it interacts with is also provided, including label names and indexes, the number of times they appear together, and their respective *SCUMBLE* values.

The information given by these functions can be useful in different scenarios. For instance, the indexes of minority labels and the labels they interact with would be convenient to customize the interaction plot previously described.

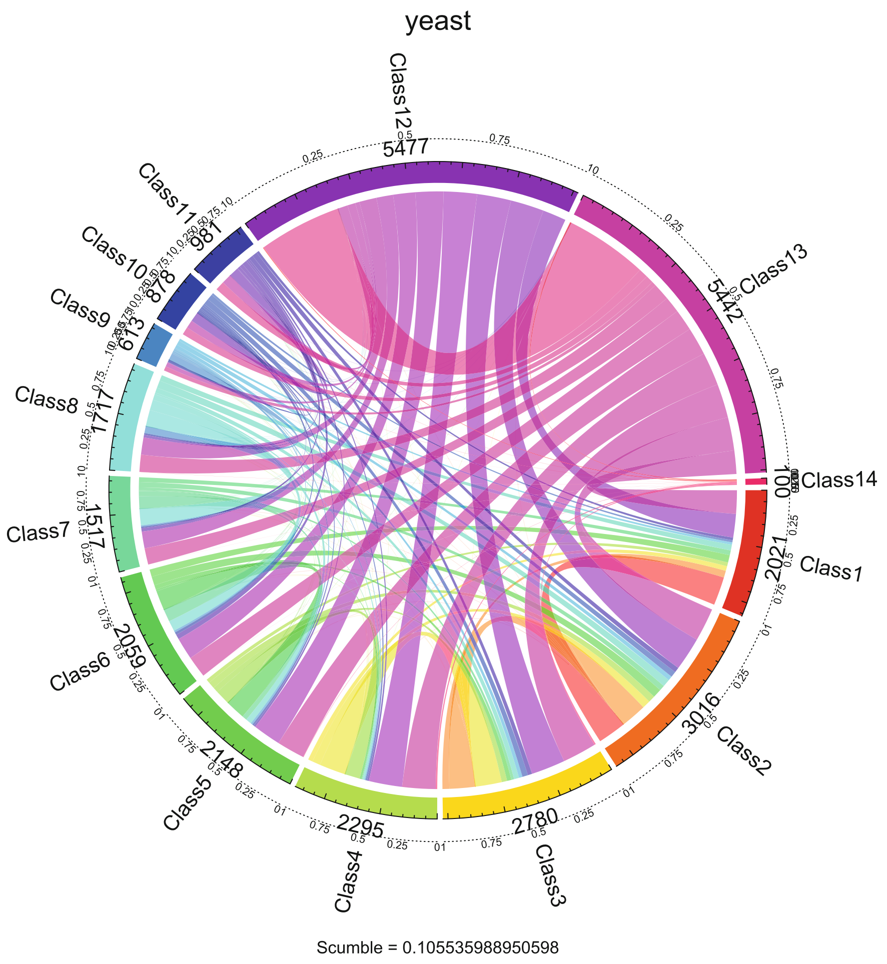
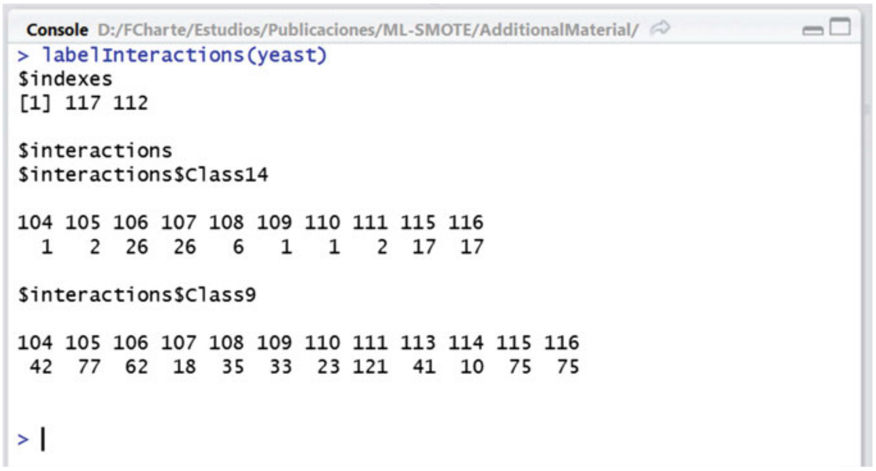


Fig. 9.16 Label interactions in the yeast MLD

9.3.2.5 The mldr User Interface

All the exploratory functions in the mldr package are accessible through the command line, so that R scripts can be written to perform reproducible analysis procedures. However, all the functionality already described is also reachable through the integrated GUI. To open it, the user has to enter the `mldrGUI()` sentence. The GUI will be launched inside the default Web browser, consisting of several sections.

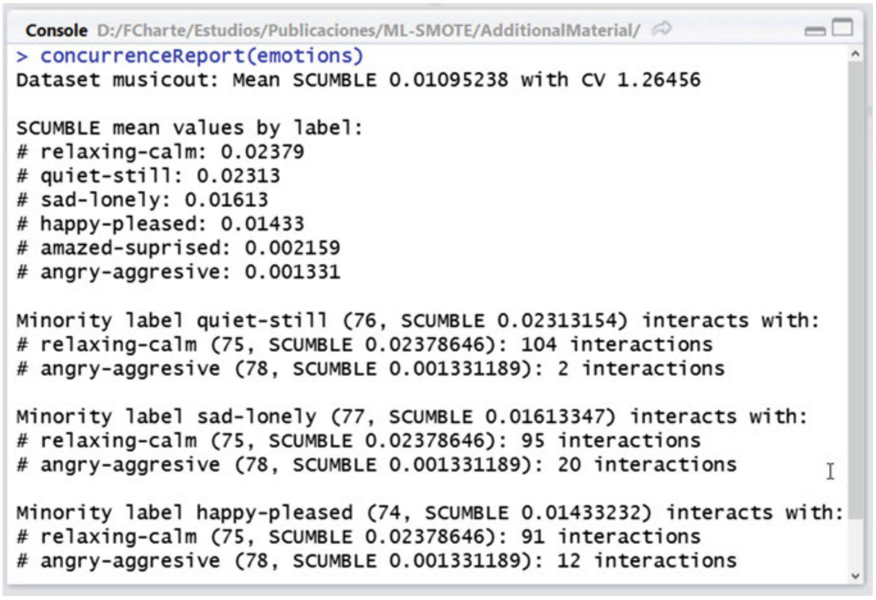
Just after the GUI has been launched, the main section is shown and the first MLD available in the working environment is selected. The name of the active MLD is always shown at the top of the GUI. A panel in this section (see Fig. 9.19) allows the user to choose from the loaded MLDs, as well as to load any others from their files.



Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/

```
> labelInteractions(yeast)
$indexes
[1] 117 112

$interactions
$interactions$Class14

104 105 106 107 108 109 110 111 115 116
  1   2  26  26   6   1   1   2  17  17

$interactions$Class9

104 105 106 107 108 109 110 111 113 114 115 116
 42  77  62  18  35  33  23 121  41  10  75  75


> |
```

Fig. 9.17 Report about how minority labels interact with other labels

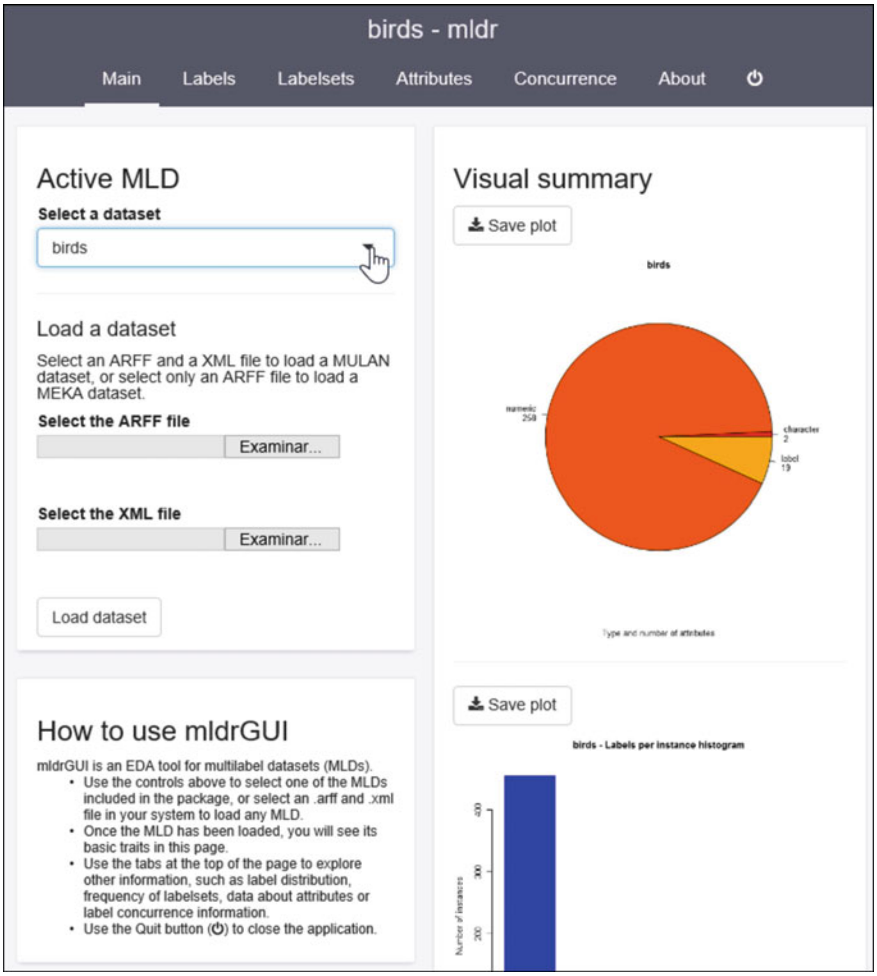


Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/

```
> concurrenceReport(emotions)
Dataset musicout: Mean SCUMBLE 0.01095238 with CV 1.26456

SCUMBLE mean values by label:
# relaxing-calm: 0.02379
# quiet-still: 0.02313
# sad-lonely: 0.01613
# happy-pleased: 0.01433
# amazed-suprised: 0.002159
# angry-aggresive: 0.001331

Minority label quiet-still (76, SCUMBLE 0.02313154) interacts with:
# relaxing-calm (75, SCUMBLE 0.02378646): 104 interactions
# angry-aggresive (78, SCUMBLE 0.001331189): 2 interactions

Minority label sad-lonely (77, SCUMBLE 0.01613347) interacts with:
# relaxing-calm (75, SCUMBLE 0.02378646): 95 interactions
# angry-aggresive (78, SCUMBLE 0.001331189): 20 interactions

Minority label happy-pleased (74, SCUMBLE 0.01433232) interacts with:
# relaxing-calm (75, SCUMBLE 0.02378646): 91 interactions
# angry-aggresive (78, SCUMBLE 0.001331189): 12 interactions
```

Fig. 9.18 Report about concurrence of labels in the MLD

A visual summary of the selected MLD is provided at the right, in the same section, including plots that show the proportion of each kind of attributes and how labels and labelsets are distributed. These plots can be saved for further use, for instance including them in any study.


birds - mldr

Main Labels Labelsets Attributes Concurrence About

Active MLD

Select a dataset

birds

Load a dataset

Select an ARFF and a XML file to load a MULAN dataset, or select only an ARFF file to load a MEKA dataset.

Select the ARFF file

Examinar...

Select the XML file

Examinar...

Load dataset

How to use mldrGUI

mldrGUI is an EDA tool for multilabel datasets (MLDs).

- Use the controls above to select one of the MLDs included in the package, or select an .arff and .xml file in your system to load any MLD.
- Once the MLD has been loaded, you will see its basic traits in this page.
- Use the tabs at the top of the page to explore other information, such as label distribution, frequency of labelsets, data about attributes or label concurrence information.
- Use the Quit button (⏻) to close the application.

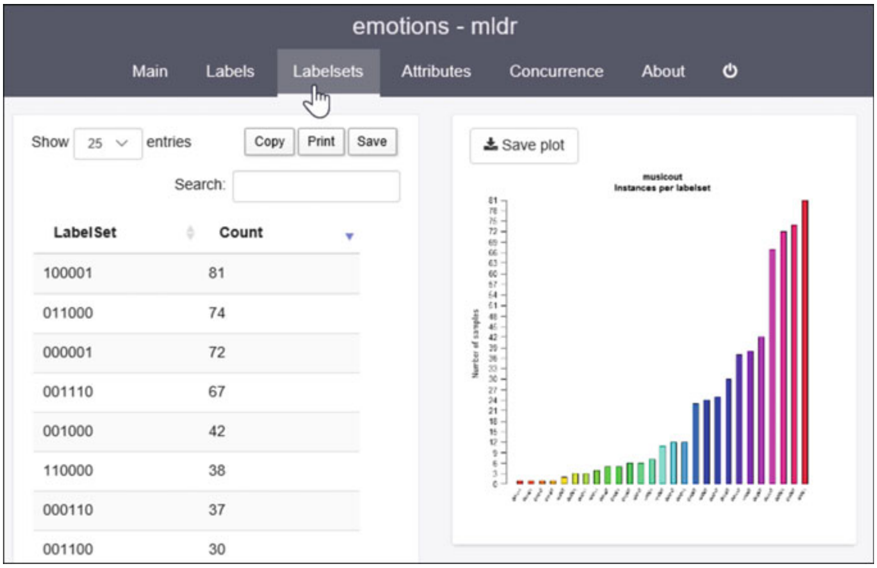
Visual summary

Fig. 9.19 Main page of mldr's GUI

The tabs located at the top of the user interface, below the selected MLD name, are the doors to the other GUI sections. The **Labels** and **Labelsets** pages are quite similar. Both provide a table with names and frequencies, as well as the associated bar plot. The labels table includes additional data, such as the feature index, imbalance levels, and concurrence levels. These tables can be sorted and filtered, and their content can be printed and exported, as is shown in Fig. 9.20.

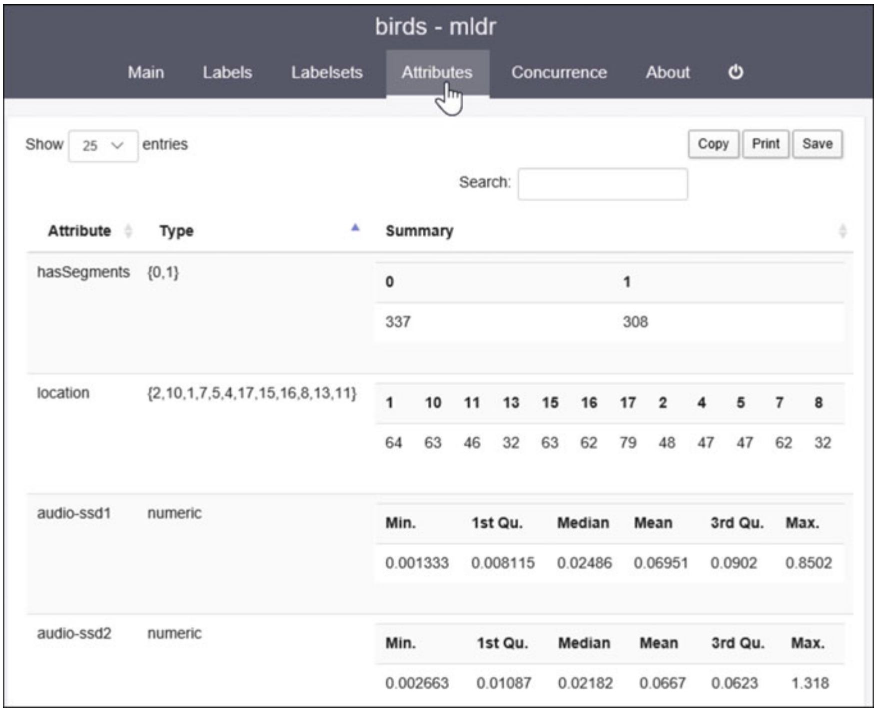
In the **Attributes** page, the domain of each input attribute is summarized. The frequency of each value is computed for nominal attributes, while for numeric ones, some statistical measures are computed (see Fig. 9.21).

By opening the **Concurrence** section, the user can access a concurrence analysis report, along with a customizable circular plot. The labels presented in it can be inter-



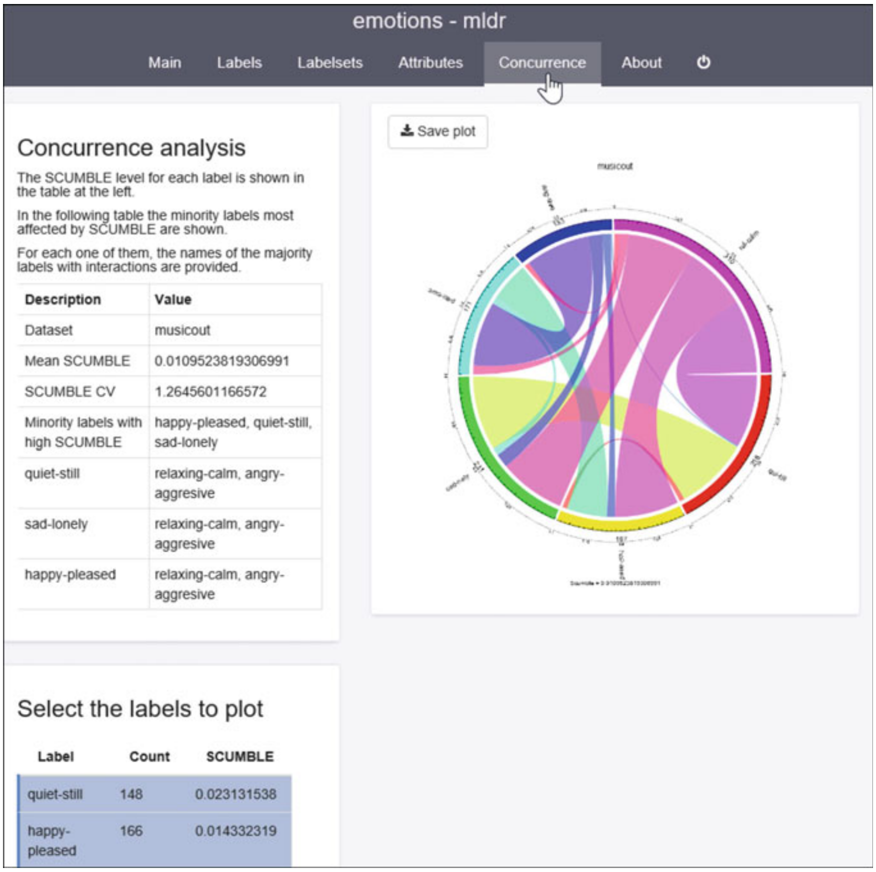
LabelSet	Count
100001	81
011000	74
000001	72
001110	67
001000	42
110000	38
000110	37
001100	30

Fig. 9.20 Table with all the labelsets and their frequencies and the corresponding plot



Attribute	Type	Summary
hasSegments	{0,1}	**0**: 337; **1**: 308
location	{2,10,1,7,5,4,17,15,16,8,13,11}	**1**: 64; **10**: 63; **11**: 46; **13**: 32; **15**: 63; **16**: 62; **17**: 79; **2**: 48; **4**: 47; **5**: 47; **7**: 62; **8**: 32
audio-ssd1	numeric	**Min.** 0.001333; **1st Qu.** 0.008115; **Median** 0.02486; **Mean** 0.06951; **3rd Qu.** 0.0902; **Max.** 0.8502
audio-ssd2	numeric	**Min.** 0.002663; **1st Qu.** 0.01087; **Median** 0.02182; **Mean** 0.0667; **3rd Qu.** 0.0623; **Max.** 1.318

Fig. 9.21 The Attributes page provides a summary of each attribute domain



Concurrence analysis

The SCUMBLE level for each label is shown in the table at the left.

In the following table the minority labels most affected by SCUMBLE are shown.

For each one of them, the names of the majority labels with interactions are provided.

Description	Value
Dataset	musicout
Mean SCUMBLE	0.0109523819306991
SCUMBLE CV	1.2645601166572
Minority labels with high SCUMBLE	happy-pleased, quiet-still, sad-lonely
quiet-still	relaxing-calm, angry-aggresive
sad-lonely	relaxing-calm, angry-aggresive
happy-pleased	relaxing-calm, angry-aggresive

Select the labels to plot

Label	Count	SCUMBLE
quiet-still	148	0.023131538
happy-pleased	166	0.014332319

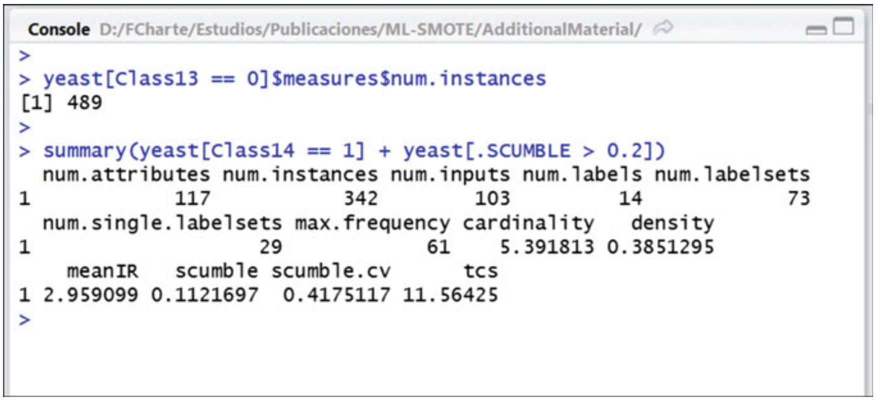
Fig. 9.22 The concurrence report along with the customizable interaction plot

actively selected, updating the visualization until the desired result is achieved. The report is based on the concurrence and imbalance metrics, indicating which minority labels have high interactions and which labels interact with others (Fig. 9.22).

Overall, the exploratory functions and GUI of the mldr package supply the user with an extensive range of useful information. In addition, the GUI allows customizing the tables and plots, and then exporting them, easing the process of documenting any report or study.

9.3.2.6 Other mldr Functions

In addition to the exploratory analysis functionality, the mldr package also offers the user functions to transform and filter the MLDs, as well as to evaluate the predictive performance from the outputs obtained by any classifier. Some of these functions are



```
Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/
>
> yeast[Class13 == 0]$measures$num.instances
[1] 489
>
> summary(yeast[Class14 == 1] + yeast[.SCUMBLE > 0.2])
  num.attributes num.instances num.inputs num.labels num.labelsets
1            117           342        103         14            73
  num.single.labelsets max.frequency cardinality   density
1                   29            61    5.391813 0.3851295
    meanIR    scumble scumble.cv       tcs
1 2.959099 0.1121697  0.4175117 11.56425
>
```

Fig. 9.23 Filtering data instances and joining two subsets

implemented as operators, preserving the natural syntax used in the R language to accomplish similar operations.

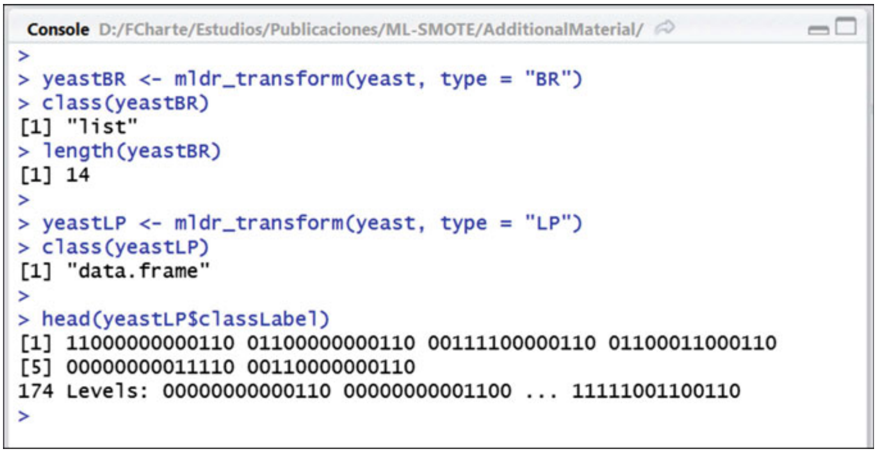
By means of the == and + operators, two `mldr` objects can be compared and combined. Two MLDs are equal as long as they have the same structure and content. To join two MLDs, they have to be structurally identical. With the `[]`,[3] the instances in the MLD can be filtered according to any valid expression. An example on how to use the latter operator to get the samples in which a certain label appears is shown in Fig. 9.23. The result returned by the operator is also an `mldr` object; thus, it can be used as any other mldr dataset. The same is applicable to the + operator.

The basic BR and LP multilabel data transformations can be applied to any `mldr` object through the `mldr_transform()` function. The only arguments needed are the MLD to transform and the `type` parameter indicating which transformation to apply. The valid values are `"BR"` and `"LP."` Optionally, a third parameter can be given stating the labels to use. This way, the transformation can be limited to a subset of labels. The value returned by this function will depend on the kind of transformation requested. For BR, it will be a list with as many `data.frame` objects as labels, each containing a binary dataset. For LP, only a `data.frame` is produced as output.

As is shown in Fig. 9.24, a column named `classLabel` is introduced in the `data.frame` instead of the original labels. This way, the resulting `data.frame` can be used with any of the binary or multiclass classifiers available in R, using `classLabel` as the class to predict.

Although the mldr package does not provide any multilabel classifier, a function to evaluate predictions obtained by any other means is supplied. This function, named `mldr_evaluate()`, takes two arguments, the `mldr` object with the instances

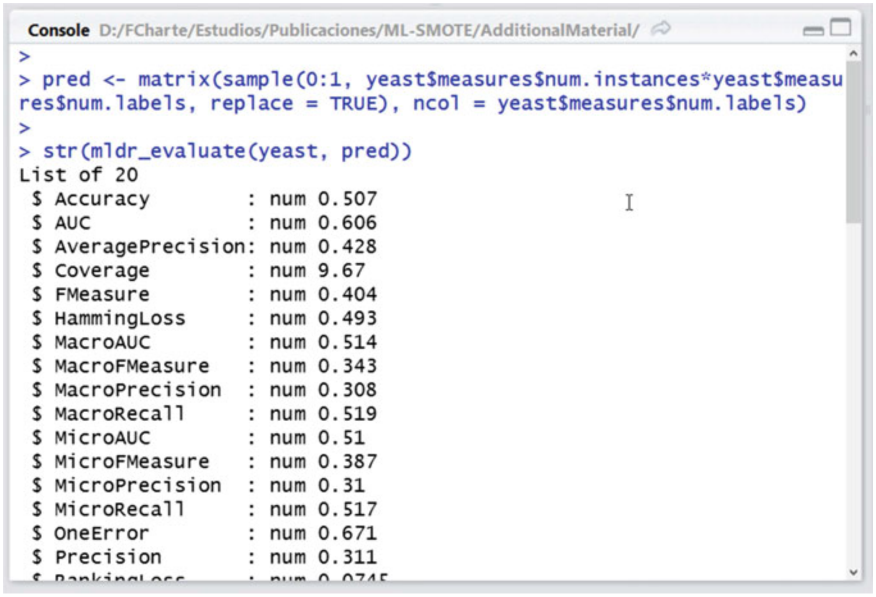
[3]The `[]` operator defined in the mldr package is designed to work with `mldr` objects only. The standard `[]` R operator can be used over the `mldr$dataset` member to manipulate the raw multilabel data.


Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/

```
>
> yeastBR <- mldr_transform(yeast, type = "BR")
> class(yeastBR)
[1] "list"
> length(yeastBR)
[1] 14
>
> yeastLP <- mldr_transform(yeast, type = "LP")
> class(yeastLP)
[1] "data.frame"
>
> head(yeastLP$classLabel)
[1] 11000000000110 01100000000110 00111100000110 01100011000110
[5] 00000000011110 00110000000110
174 Levels: 00000000000110 00000000001100 ... 11111001100110
>
```

Fig. 9.24 The mldr package provides a function to perform the usual BR and LP transformations

being assessed and the predictions obtained for them. The function returns a list containing about twenty performance metrics (see Fig. 9.25), along with an object containing all the data needed to plot the ROC curve. The example shown in Fig. 9.25 generates random predictions for all the labels in the yeast dataset and then evaluates the performance. As expected, the accuracy of this random classifier is around 50 %.


Console D:/FCharte/Estudios/Publicaciones/ML-SMOTE/AdditionalMaterial/

```
>
> pred <- matrix(sample(0:1, yeast$measures$num.instances*yeast$measu
res$num.labels, replace = TRUE), ncol = yeast$measures$num.labels)
>
> str(mldr_evaluate(yeast, pred))
List of 20
 $ Accuracy        : num 0.507
 $ AUC             : num 0.606
 $ AveragePrecision: num 0.428
 $ Coverage        : num 9.67
 $ FMeasure        : num 0.404
 $ HammingLoss     : num 0.493
 $ MacroAUC        : num 0.514
 $ MacroFMeasure   : num 0.343
 $ MacroPrecision  : num 0.308
 $ MacroRecall     : num 0.519
 $ MicroAUC        : num 0.51
 $ MicroFMeasure   : num 0.387
 $ MicroPrecision  : num 0.31
 $ MicroRecall     : num 0.517
 $ OneError        : num 0.671
 $ Precision       : num 0.311
 $ RankingLoss     : num 0.0745
```

Fig. 9.25 If a set of predictions is available, the performance of the classifier can be assessed

Exploratory analysis is a fundamental step to understand the data we are working on, providing the necessary knowledge to decide which preprocessing and learning methods should be applied. MEKA offers a GUI with some general tools, based on the popular WEKA. On the other hand, the mldr package delivers a plethora of specific multilabel metrics, plots, and reports, along with several functions to manipulate these kinds of data.

9.4 Conducting Multilabel Experiments

Once the user has learned what the structure of their MLDs is, how labels are distributed and correlated, etc., it is time to conduct some experiments using the multilabel data. This goal can imply applying some preprocessing method to the data, train a classifier using it, obtain predictions for the test set, and eventually evaluate these predictions to assess the performance.

Several software tools and packages for different languages are available to accomplish these tasks. However, there are two applications that stand out among everything else, MEKA and MULAN. Both have been developed by experts on the multilabel field, and they provide reference implementations for a large variety of MLC algorithms.

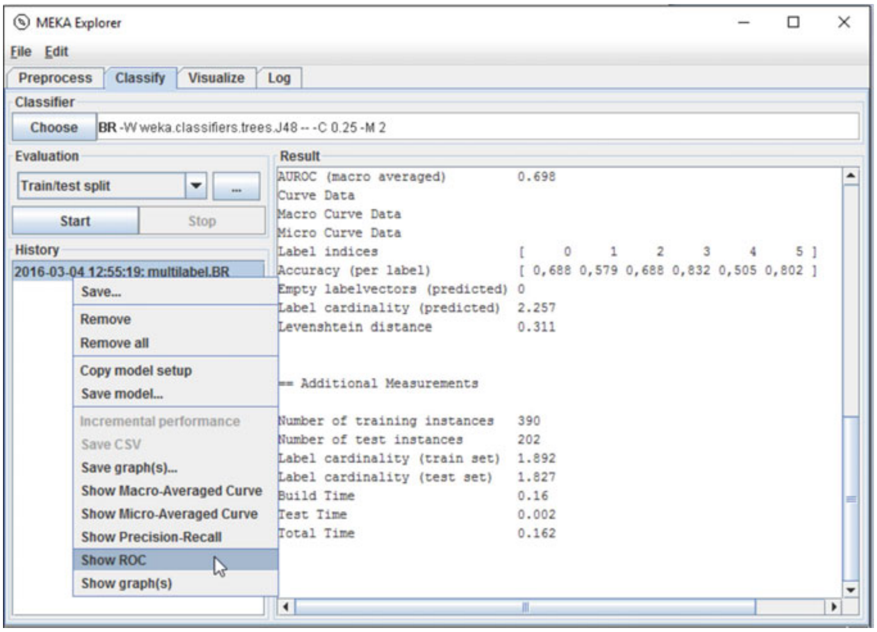
In this final section, how to use MEKA and MULAN to run simple multilabel experiments is explained. Although the former tool can be used from the command line, it is supplied with a GUI which eases the user's work. On the contrary, the latter only is accessible programmatically, as will be later shown. Furthermore, both software packages have been programmed in Java language, so that the latest JRE installed in the system is assumed.

9.4.1 MEKA

The MEKA user interface was previously introduced. Specifically, the tool known as MEKA Explorer was used to make some exploratory analysis on the data. This section starts from this earlier knowledge, firstly describing how to conduct interactive experiments and then designing more complex ones.

9.4.1.1 Running Experiments Interactively

The **Classify** page on this application allows the user to choose among an extensive list of classifiers, training them with the loaded MLD and evaluating them following



BR -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

2016-03-04 12:55:19: multilabel.BR

```
AUROC (macro averaged)        0.698
Curve Data
Macro Curve Data
Micro Curve Data
Label indices                 [     0     1     2     3     4     5 ]
Accuracy (per label)          [ 0,688 0,579 0,688 0,832 0,505 0,802 ]
Empty labelvectors (predicted) 0
Label cardinality (predicted)  2.257
Levenshtein distance          0.311


== Additional Measurements

Number of training instances   390
Number of test instances       202
Label cardinality (train set)  1.892
Label cardinality (test set)   1.827
Build Time                     0.16
Test Time                      0.002
Total Time                     0.162
```

Fig. 9.26 The MEKA Explorer allows the user to run any classifier interactively

several validation strategies. The first step is to load the MLD of interest. Then, by means of the **Choose** button in the **Classify** page, the classifier is selected, setting its parameters as desired. The drop-down list in the **Evaluation** section configures the validation scheme.

Once the experiment has been configured, it will be run by clicking the **Start** button. As soon as it finishes displaying a summary of results, as is shown in Fig. 9.26. Each experiment execution is independently stored in the **History** list, so the user can compare the results produced in each case.

The pop-up menu associated with the items in the **History** list can be used to save the obtained results, delete the experiment, copy its parameters, and show several performance evaluation plots. This menu appears opened in Fig. 9.26. By selecting the **Show ROC** option, a new window like the one shown in Fig. 9.27 is opened. It contains several pages, one per label, each one with the ROC plot.

Overall, the MEKA Explorer offers the user with a very simple way to run individual experiments. However, they are limited to using one MLC algorithm over one MLD.

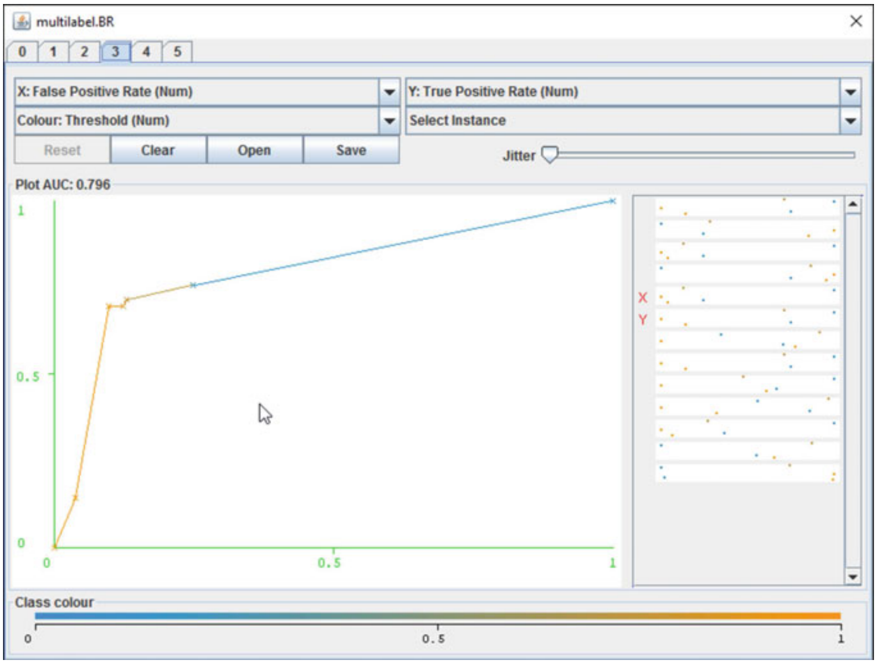
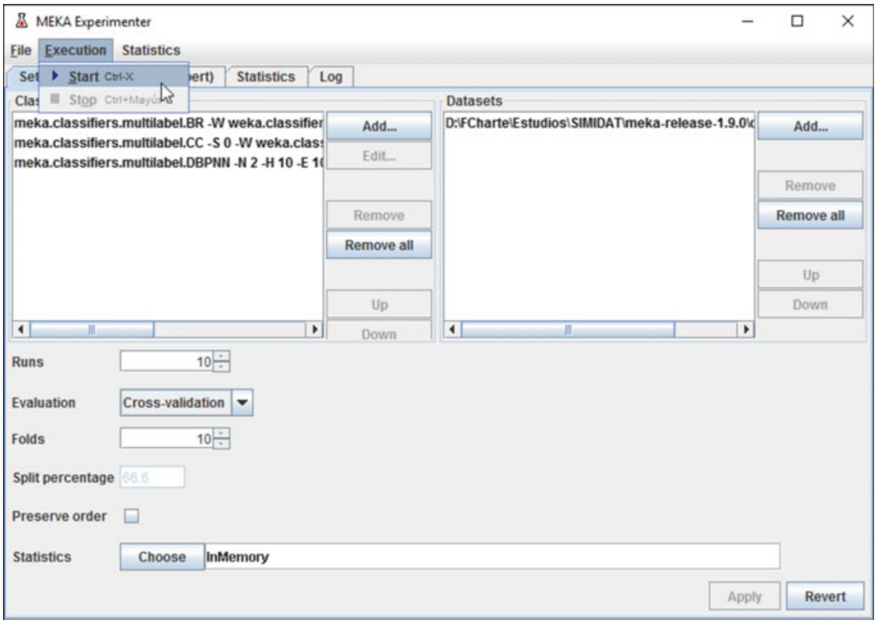


Fig. 9.27 The ROC curve for the selected experiment is shown in a window in its own

9.4.1.2 Designing Complex Experiments

Though running individual MLC algorithms with one MLD can be useful many times, for instance while analyzing the algorithm behavior, a real multilabel experimentation usually implies applying several methods to a group of MLDs. The MEKA Experimenter, another tool accessible from the MEKA main window, better suits this kind of job. As the MEKA Explorer, the Experimenter user interface also consists of several pages. The first one is where the user will design the experiment, fulfilling the following steps:

1. Adding the MLC algorithms to be used to the left list. The buttons at the right of this list allow the user to add and remove methods, configure them, and change the order they will be run.
2. Adding the MLDs which the algorithms will be applied to the right list. The buttons at the right of this list are similar to the previous ones.
3. Configuring how many times the experiment will be run, as well as how the results will be evaluated. For instance, in Fig. 9.28, a configuration with 10-fold cross-validation and 10 repetitions has been established.
4. Launching the experimentation by choosing the **Start** options in the **Execution** menu. Previously, all the configuration can be saved.



Classifiers:
meka.classifiers.multilabel.BR -W weka.classifier
meka.classifiers.multilabel.CC -S 0 -W weka.clas
meka.classifiers.multilabel.DBPNN -N 2 -H 10 -E 1

Datasets:
D:\FCharte\Estudios\SIMIDAT\meka-release-1.9.0\

Runs: 10
Evaluation: Cross-validation
Folds: 10
Split percentage: 66.6
Preserve order: ☐
Statistics: InMemory

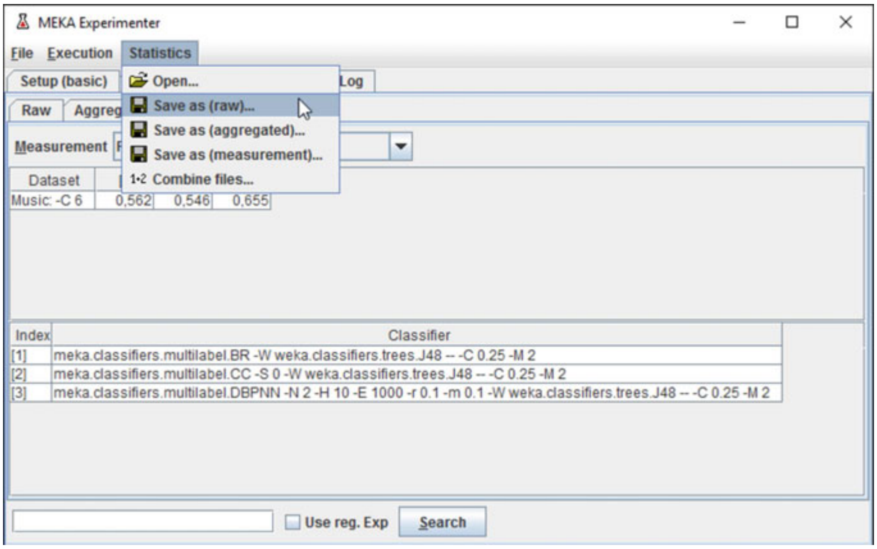
Fig. 9.28 The MEKA Experimenter is able to run several classifiers using disparate MLDs

Once the batch of runs defined in the experimentation finishes, the obtained evaluation results can be analyzed through the options in the **Statistics** page of the MEKA Experimenter. An extensive set of performance metrics is provided, and they can be viewed separately or aggregating them. For instance, it is possible to choose a specific metric and get the average values for each algorithm and dataset. This way, a comparative study can be easily performed.

The results produced by the experiments can be also written to a file, whether the user is interested in raw or aggregated data. As shown in Fig. 9.29, the options relating to exporting functions can be found in the **Statistics** menu.

9.4.2 MULAN

As MEKA, MULAN [1] is a multilabel software framework built on top of WEKA. However, it does not bring the user with a GUI. All the tasks have to be accomplished by writing Java code. Therefore, some experience with this programming language is essential. In addition to the aforementioned JRE, to use MULAN the Java Development Kit (JDK) is also needed. The JDK contains the compiler, among other Java tools and utilities.



Dataset			
Music: -C 6	0,562	0,546	0,655

Index	Classifier
[1]	meka.classifiers.multilabel.BR -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
[2]	meka.classifiers.multilabel.CC -S 0 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
[3]	meka.classifiers.multilabel.DBPNN -N 2 -H 10 -E 1000 -r 0.1 -m 0.1 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

Fig. 9.29 The experimental results can be viewed in MEKA Experimenter and exported to a file

In this section, the procedures to design a multilabel experiment using MULAN are briefly described. Additional example code can be found both in the MULAN Web site (http://mulan.sourceforge.net) and in this book's GitHub repository.

9.4.2.1 Obtaining MULAN

The first step of a MULAN user is obtaining the software itself. The last version can be downloaded from http://mulan.sourceforge.net/download.html as a single compressed file. After decompressing it, a folder named `mulan` will be found, containing the folders and files shown in Fig. 9.30. Nearly all the folder names are self-explanatory. The Java JAR package holding the MULAN classes is in the `dist` folder, while the WEKA package needed to run MULAN is provided in the `lib` folder.

9.4.2.2 Compiling a MULAN Experiment

Once MULAN is installed in the user system, to conduct any experiment a Java source file has to be created. It will contain all the code to load the MLDs, to select and configure the MLC algorithms to run, to evaluate the obtained results, etc. These tasks imply using some MULAN classes; therefore, the proper dependencies have to be imported at the beginning of the source code. Assuming the user needs to load an MLD and wants to use the ML-kNN classifier, the following sentences will import the corresponding classes:

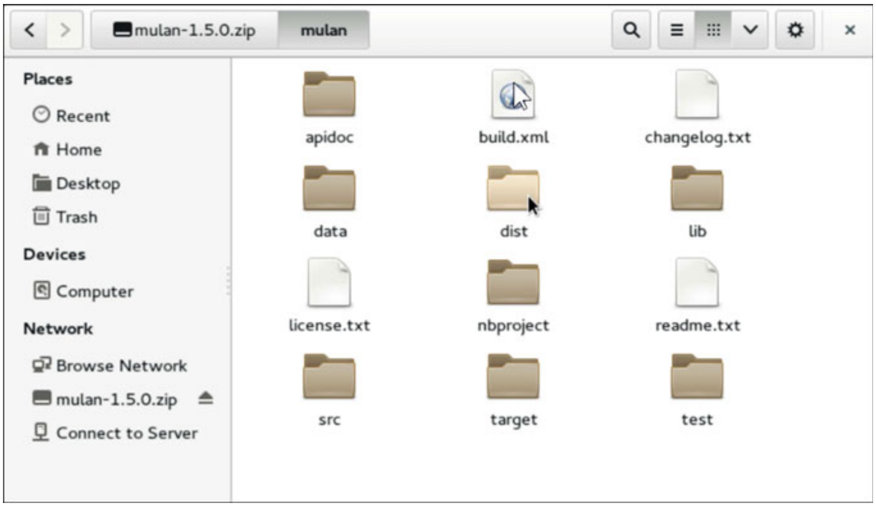
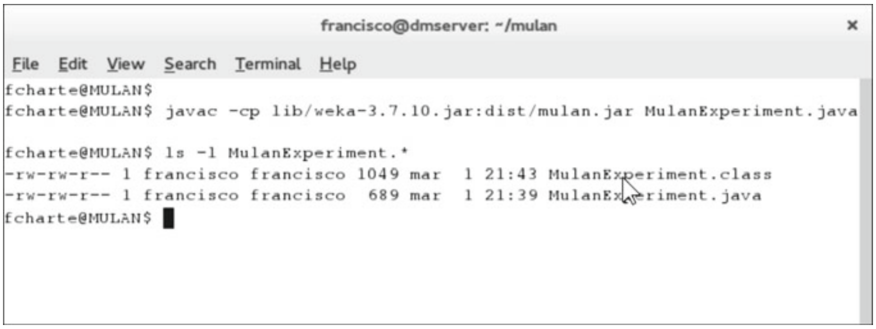


Fig. 9.30 The MULAN software packages include source code, example data files, and documentation

```
import mulan.data.MultiLabelInstances;
import mulan.classifier.lazy.MLkNN;
```

The way some of these MULAN classes are used is a matter which will be further described. Considering the code is already written and stored in a file named `MulanExperiment.java`, the compilation process involves calling the Java compiler providing the path to libraries and the aforementioned filename. The example in Fig. 9.31 shows a GNU/Linux terminal with the command line used to compile this hypothetical program. After compilation, a `.class` file with the compiled version is generated. To run this program, a similar command would be used, but changing `javac` for `java`.


francisco@dmserver: ~/mulan
File Edit View Search Terminal Help

```
fcharte@MULAN$
fcharte@MULAN$ javac -cp lib/weka-3.7.10.jar:dist/mulan.jar MulanExperiment.java

fcharte@MULAN$ ls -l MulanExperiment.*
-rw-rw-r-- 1 francisco francisco 1049 mar  1 21:43 MulanExperiment.class
-rw-rw-r-- 1 francisco francisco  689 mar  1 21:39 MulanExperiment.java
fcharte@MULAN$ 
```

Fig. 9.31 To compile a MULAN program, the paths to the mulan.jar and weka.jar files have to be provided

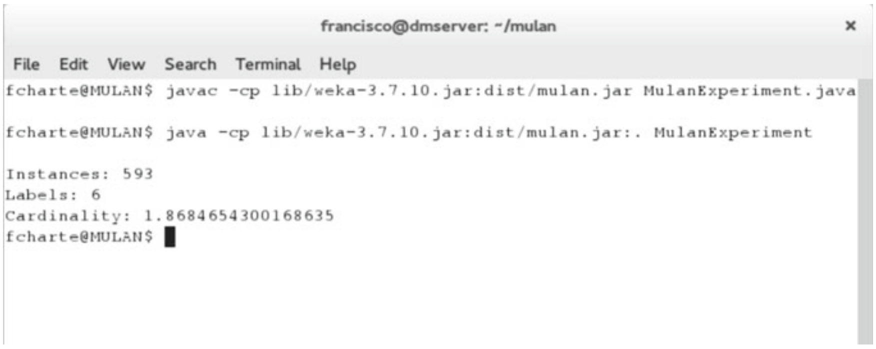
9.4.2.3 Loading Data Files

The `MultiLabelInstances` class is able to load a set of multilabel instances from a file. This class constructor usually is given two parameters, the path of the ARFF file and the path of the associated XML file. Other ways to obtain the data are considered, for instance from existing WEKA data samples and an array stating which ones are labels.

An initialized `MultiLabelInstances` object can be passed as argument to different methods. It also offers several functions which return data traits, such as the number of instances or labels, and label cardinality.

Assuming the user is working in the `mulan` folder, so a `data` subfolder with some example data is available, and the code below will load the emotions dataset and then print the number of instances and labels and the label cardinality. Figure 9.32 shows how the program is compiled and executed.

```
import mulan.data.MultiLabelInstances;

public class MulanExperiment {
  public static void main(String[] args)
  throws Exception {
    MultiLabelInstances emotions =
      new MultiLabelInstances("data/emotions.arff",
                              "data/emotions.xml");

    System.out.println(
       "\nInstances:" + emotions.getNumInstances() +
       "\nLabels:" + emotions.getNumLabels() +
       "\nCardinality:" + emotions.getCardinality());
  }
}
```


francisco@dmserver: ~/mulan
File Edit View Search Terminal Help

```
fcharte@MULAN$ javac -cp lib/weka-3.7.10.jar:dist/mulan.jar MulanExperiment.java

fcharte@MULAN$ java -cp lib/weka-3.7.10.jar:dist/mulan.jar:. MulanExperiment

Instances: 593
Labels: 6
Cardinality: 1.8684654300168635
fcharte@MULAN$
```

Fig. 9.32 The program loads the MLD and outputs some data about it to the console

The name of the files to load could obviously be supplied in the command line, instead of being hardwired in the Java code as in this simple example.

9.4.2.4 Configuring MLC Algorithms

Once the data are already in a `MultiLabelInstances` object, the next step is to configure the MLC algorithms this object is going to be given as input. MULAN provides a large set of classification algorithms, they are held in the `mulan.classifier` namespace, and some partitioning and preprocessing methods spread out several namespaces. All of them can be easily found in the electronic help of the program.

Many of the algorithms included in MULAN can work using default values, so the corresponding object is created without needing any parameters. For instance, to work with the ML-kNN algorithm (see Chap. 5), all the user has to do is to create an `MLkNN` object, as follows:

```
MLkNN kNNClassifier = new MLkNN();
```

To change the default values for the algorithm, a different constructor accepting them should be called. Other classifiers, such as the ones based on transformation methods, always need at least one parameter, specifically the base binary or multiclass method to be used as underlying classifier. Any classifier available as a WEKA class can be in charge of this task.

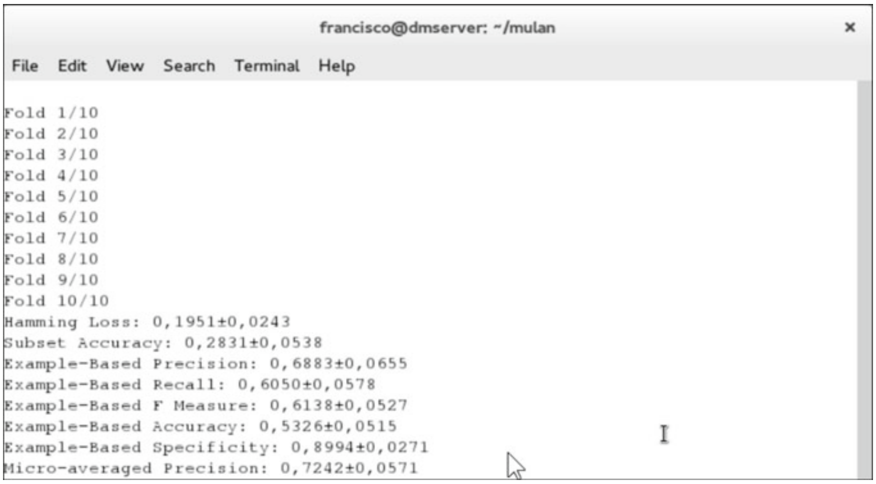
The following sentences would create two classifier instances. The first one is ML-kNN with 5 nearest neighbors, while the second one is a BR transformation using the standard C4.5[4] algorithm as base classifier. The last sentence prints in the standard output the basic information about the ML-kNN algorithm.

```
MLkNN kNNClassifier = new MLkNN(5, 1);
BinaryRelevance BRClassifier =
     new BinaryRelevance(new J48());

System.out.println(
    "\nkNNClassifier:" +
    kNNClassifier.getTechnicalInformation().toString());
}
```

9.4.2.5 Training and Evaluating the Classifier

MULAN has a class named `Evaluator` able to take a full dataset, partition it, and conduct a cross-validation evaluation. The method to accomplish the full procedure

[4]The C4.5 algorithm is implemented in WEKA by the `J48` class.


francisco@dmserver: ~/mulan
File Edit View Search Terminal Help

```
Fold 1/10
Fold 2/10
Fold 3/10
Fold 4/10
Fold 5/10
Fold 6/10
Fold 7/10
Fold 8/10
Fold 9/10
Fold 10/10
Hamming Loss: 0,1951±0,0243
Subset Accuracy: 0,2831±0,0538
Example-Based Precision: 0,6883±0,0655
Example-Based Recall: 0,6050±0,0578
Example-Based F Measure: 0,6138±0,0527
Example-Based Accuracy: 0,5326±0,0515
Example-Based Specificity: 0,8994±0,0271
Micro-averaged Precision: 0,7242±0,0571
```

Fig. 9.33 Partial output produced by printing the information returned by crossValidate()

is `crossValidate()`, and it needs three parameters: the classification model, the dataset, and the number of folds to be used.

Assuming the proper namespaces have been imported and the `emotions` variable is a `MultiLabelInstances` object with the MLD, the following sentence would produce an output similar to that shown in Fig. 9.33. A 10-fold cross-validation is performed using the ML-kNN algorithm, and average values for a large set of evaluation metrics are returned.

```
System.out.println(
   new Evaluator().crossValidate(
      new MLkNN(), emotions, 10));
```

Instead of relying on an automated partitioning, training, and evaluation process, the users can separately run each step on their own. The training and test partitions, maybe generated by the functions in the mldr.datasets package previously described, can be individually loaded from their files. The full dataset can be also partitioned using the `IterativeStratification` class. In any case, two or more `MultiLabelInstances` objects will be used.

The training of any MULAN classifier is accomplished by calling its `build()` method. It needs only one argument, and the `MultiLabelInstances` objects with the samples aimed to train the model. Once trained, the model can be evaluated or used to get predictions for new instances. The former task is handled by the `evaluate()` method of the `Evaluator` class, taking as input the model and the test instances. The latter is in charge of the `makePrediction()` method of the model itself. It takes a `MultiLabelInstance` object as input, with the instance the prediction is aimed for.

Table 9.1 Input parameters for the RunMLClassifier utility

Name	Description	Example
`-path`	Establishes the path where the data files are located	`-path ~/data`
`-dataset`	Indicates the root name shared by all the dataset partitions. Training files have to be named as `MLD-strategy-Ntra.arff`, while test files will be `MLD-strategy-Ntst.arff`. The `strategy` part can be any user identifier for the partitioning strategy. `N` will be a sequential partition number. That an XML file with the name `MLD.xml` file is also in the folder is assumed	`-dataset emotions-10cv`
`-folds`	Sets the number of folds to iterate. The `N` part of the dataset name will take values from `1` to the number specified by this argument	`-folds 10`
`-algorithm`	Chooses the MLC algorithm to be used. The list of values accepted by this parameter is shown in Table 9.2	`-algorithm HOMER`
`-debug`	If included, this optional parameter will change the output produced by the program, increasing the information printed to the console	`-debug`

9.4.3 The RunMLClassifier Utility

The main drawback in using MULAN is the need to have some Java language expertise. The authors of this book provide in the companion repository an utility, named `RunMLClassifier`, aimed to help users lacking this competence. It is a Java program which eases running many of the classifiers implemented in MULAN. The source code of this program, along the specific versions of the MULAN and WEKA libraries and a compilation script, can be found in the `RunMLClassifier` folder of the repository.

To run the `RunMLClassifier` program, assuming the user is located in the same directory that the `.jar` file is obtained once it is compiled, the following sentence[5] has to be entered at the command line. The meaning of each parameter is detailed in Table 9.1.

```
java -jar RunMLClassifier.jar -path P -dataset D -folds
F -algorithm A [-debug]
```

[5]Although due to the page width limit the sentence appears in the text divided into two lines, it has to be entered as only one.

Table 9.2 Valid values for the `-algorithm` parameter of the RunMLClassifier utility

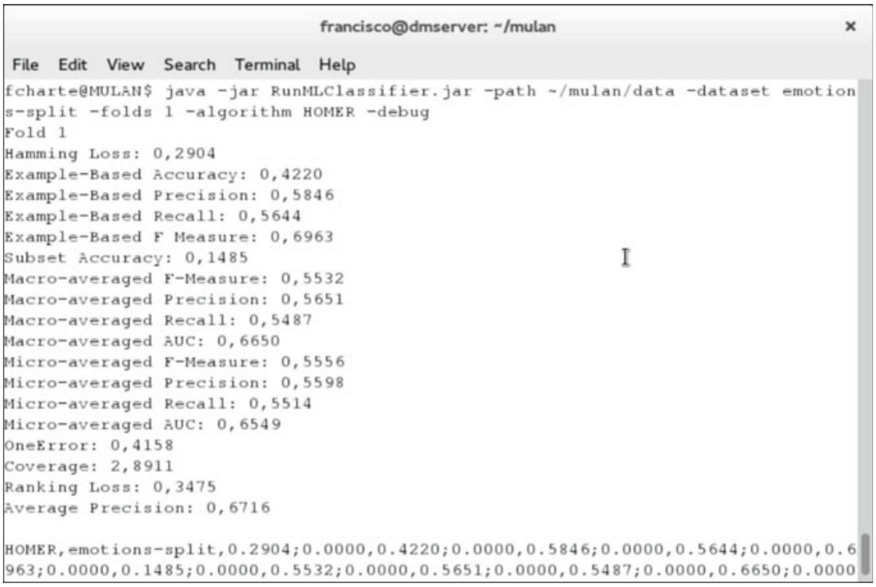
Value	MULAN class instantiated as classifier
`BPMLL`	`BPMLL()`
`BR-J48`	`BinaryRelevance(new J48())`
`BRkNN`	`BRkNN(10)`
`CC-J48`	`ClassifierChain(new J48())`
`CLR`	`CalibratedLabelRanking(new J48())`
`ECC-J48`	`EnsembleOfClassifierChains(new J48(), 10, true, false)`
`EPS-J48`	`EnsembleOfPrunedSets(80, 10, 0.2, 2, PrunedSets.Strategy.values()[0], 2, new J48())`
`HOMER`	`HOMER(new BinaryRelevance(new J48()), (numLabels < 4 ? numLabels : 4), Method.Random)`
`IBLR-ML`	`IBLR_ML()`
`LP-J48`	`LabelPowerset(new J48())`
`MLkNN`	`MLkNN(10, 1.0)`
`PS-J48`	`PrunedSets(new J48(), 2, PrunedSets.Strategy.values()[0], 2)`
`RAkEL-BR`	`RAkEL(new BinaryRelevance(new J48()))`
`RAkEL-LP`	`RAkEL(new LabelPowerset(new J48()))`

The utility instances each classifier using default or recommended values. As can be seen, only one classifier can be chosen to be run over the MLD partitions. The `RunMLClassifier` is designed to be launched independently, maybe in parallel, for each algorithm the user is interested in. An example run with this utility is shown in Fig. 9.34. Without the `-debug` parameter, only the final line which summarizes the average results and standard deviations would be printed.

MEKA and MULAN are the two main frameworks to conduct multilabel experiments, since they provide reference implementations of many MLC methods. The former has a GUI which eases the design of such experiments, while the latter only can be used while writing some code. The `RunMLClassiier` utility aims to make the use of MULAN more comfortable, establishing the methods and datasets to process through command line parameters, instead of writing and compiling a Java program.

9.5 Summarizing Comments

Learning from multilabeled data is a quite challenging task. Datasets of this kind come in diverse file formats and are distributed among a few Web repositories. Once the data files have been obtained, they have to be imported to the learning tool, some


francisco@dmserver: ~/mulan
File Edit View Search Terminal Help

```
fcharte@MULAN$ java -jar RunMLClassifier.jar -path ~/mulan/data -dataset emotion
s-split -folds 1 -algorithm HOMER -debug
Fold 1
Hamming Loss: 0,2904
Example-Based Accuracy: 0,4220
Example-Based Precision: 0,5846
Example-Based Recall: 0,5644
Example-Based F Measure: 0,6963
Subset Accuracy: 0,1485
Macro-averaged F-Measure: 0,5532
Macro-averaged Precision: 0,5651
Macro-averaged Recall: 0,5487
Macro-averaged AUC: 0,6650
Micro-averaged F-Measure: 0,5556
Micro-averaged Precision: 0,5598
Micro-averaged Recall: 0,5514
Micro-averaged AUC: 0,6549
OneError: 0,4158
Coverage: 2,8911
Ranking Loss: 0,3475
Average Precision: 0,6716

HOMER,emotions-split,0.2904;0.0000,0.4220;0.0000,0.5846;0.0000,0.5644;0.0000,0.6
963;0.0000,0.1485;0.0000,0.5532;0.0000,0.5651;0.0000,0.5487;0.0000,0.6650;0.0000
```

Fig. 9.34 The utility trains and evaluates the MLD partitions with the specified classifier

times with a preliminary conversion step. A large set of multilabel characterization metrics exist in the literature, most of them were described in Chap. 3, and dozens of methods have been defined to preprocess and classify multilabel data, as is shown in Chaps. 4 to 8.

In this chapter, the Web sites from which the MLDs can be downloaded, as well as the existing file formats, have been thoroughly detailed. Tools such as the mldr.datasets R package, along with the RUMDR repository, can automate most of the tasks associated with MLDs, including getting, citing, partitioning, and exporting them to several learning software frameworks. These MLDs can be analyzed by means of the functionality found in MEKA and the mldr R package. Both provide EDA tools, comprehensively explained through this chapter.

The last section has been focused on the use of tools aimed at conducting multilabel experiments, specifically MEKA and MULAN. Although there are a few multilabel algorithms implemented outside of these two frameworks, currently they are the most prominent and widely used.

References

1. Tsoumakas, G., Xioufis, E.S., Vilcek, J., Vlahavas, I.: MULAN: A Java library for multi-label learning. J. Mach. Learn. Res. **12**, 2411–2414 (2011)
2. Read, J., Reutemann, P.: MEKA multi-label dataset repository. http://sourceforge.net/projects/meka/files/Datasets/

3. Chang, C.C., Lin, C.J.: Libsvm: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 1–27 (2011)
4. Alcala-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., Herrera, F.: KEEL data-mining software tool: data set repository and integration of algorithms and experimental analysis framework. J. Mult-Valued Log. Soft Comput. **17**(2–3), 255–287 (2011)
5. Charte, F., Charte, D., Rivera, A.J., del Jesus, M.J., Herrera, F.: R Ultimate Multilabel Dataset Repository. https://github.com/fcharte/mldr.datasets
6. Tsoumakas, G., Xioufis, E.S., Vilcek, J., Vlahavas, I.: MULAN multi-label dataset repository. http://mulan.sourceforge.net/datasets.html
7. Chang, C.C., Lin, C.J.: LIBSVM data: multi-label classification repository. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html
8. Alcala-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sánchez, L., Herrera, F.: KEEL multi-label dataset repository. http://sci2s.ugr.es/keel/multilabel.php
9. Charte, F., Charte, D., Rivera, A.J., del Jesus, M.J., Herrera, F.: R ultimate multilabel dataset repository. In: Proceeedings of 11th International Conference on Hybrid Artificial Intelligent Systems, HAIS'16, vol. 9648, pp. 487–499. Springer (2016)
10. Bhatia, K.H., Jain, P.J., Varma, M.: The extreme classification repository: multi-label datasets & code. http://research.microsoft.com/en-us/um/people/manik/downloads/XC/XMLRepository.html
11. R Core Team: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria (2014). http://www.R-project.org/
12. Tomás, J.T., Spolaôr, N., Cherman, E.A., Monard, M.C.: A framework to generate synthetic multi-label datasets. Electron. Notes Theor. Comput. Sci. **302**, 155–176 (2014)
13. Read, J., Pfahringer, B., Holmes, G.: Generating synthetic multi-label data streams. In: Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML PKDD'09, pp. 69–84 (2009)
14. Charte, F., Charte, D.: Working with multilabel datasets in R: the mldr package. R J. **7**(2), 149–162 (2015)