IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 3, 2019, accepted March 2, 2020, date of publication March 10, 2020, date of current version March 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2979787

A Comprehensive and Didactic Review on Multilabel Learning Software Tools

FRANCISCO CHARTE, (Member, IEEE)
Department of Computer Science, Universidad de Jaén, 23071 Jaén, Spain
e-mail: fcharte@ujaen.es

This work was supported by the Spanish Ministry of Economy and Competitiveness under Project TIN2015-68854-R (FEDER Founds).

ABSTRACT Machine learning has become an everyday tool in so many fields that there is plenty of software to run many of these algorithms in every device, from supercomputers to embedded appliances. Most of these methods fall into the category known as standard learning, being supervised models (guided by pre-labeled examples) aimed to classify new patterns into exactly one category. This way, machine learning is in charge of getting rid of junk emails, labeling people in a picture, or detecting a fraudulent transaction when using a credit card. Aside from unsupervised learning methods, which are usually applied to group similar patterns, infer association rules and similar tasks, some non-standard supervised machine learning problems have been faced in late years. Among them, multilabel learning is arguably the most popular one. These algorithms aim to produce models in which each data pattern may be linked to several categories at once. Thus, a multilabel classifier generates a set of outputs instead of only one as a standard classifier does. However, software tools for multilabel learning tend to be scarce. This paper provides multilabel researchers with a comprehensive review of the currently available multilabel learning software. It is written following a didactic approach, focusing on how to accomplish each task rather than simply offering a list of programs and websites. The goal is to help finding the most appropriate resource to complete every step, from locating datasets and partitioning them to running many of the multilabel algorithms proposed in the literature until now.

INDEX TERMS Machine learning, multilabel, non-standard learning, software, tools.

I. INTRODUCTION

The availability of software such as R's `caret` package [1], Matlab's `Machine Learning` toolbox [2], Java's `WEKA` application [3] and Python's `scikit-learn` package [4], to mention only a few of the existing alternatives, puts data analysis and data mining capabilities at the fingertips of researchers, students and practitioners. Exploratory data analysis (EDA) tools are essential to understand data traits, compute diverse characterization metrics and visualize the data in proper ways. Machine Learning (ML) software, and specifically Data Mining (DM) tools, provide the means to apply proven algorithms to these data, aiming to transform or clean the data, to extract hidden knowledge or to create predictive models, among other tasks.

A large portion of the data used nowadays falls into the category of labeled data, e.g. e-mails are classified as spam or legitimate, news are grouped into topics, people are tagged as appearing in photos, etc. EDA and DM tools can take advantage of the label assigned to each data pattern, for instance by differentiating the points in a plot according to their labels, or using the labels to infer a classifier by means of supervised learning algorithms.

A typical assumption is that each data pattern is linked to only one label. Sometimes this label can take one of two values, i.e. the mail is spam or it is not. This is the binary case. If the label can belong to a limited set of values, having this set more than two elements, then it is known as the multiclass case. Most EDA and DM software tools available nowadays are aimed to work with binary and multiclass data.

Single-label learning, also known as standard learning, is probably the most common scenario when working in EDA and DM tasks, but it is certainly not the only one. There are other non-standard modalities [5], such as multilabel learning [6] (MLL), multiinstance learning [7], multiview learning [8], etc. Here we are particularly interested in MLL, since it is the most common case of non-standard learning. Currently, MLL is being used in fields such as automatic tagging of new entries in question-answering forums [9], classification of aviation safety narratives [10], content-based

The associate editor coordinating the review of this manuscript and approving it for publication was Hui Liu.

retrieval of remote sensing images [11] and prediction of chemical toxicity effects [12], among many others. Some generic reviews and tutorials on MLL learning can be found in [13]–[15].

Unlike standard learning, for which there is available a plethora of tools, software for MLL is far scarcer and also more specific. For those starting research in this field, it may be difficult to locate suitable data sets. These can be in disparate file formats, and most EDA tools are only capable to deal with a few of them. In the same way, running the most popular transformation methods and learning algorithms implies looking for the software tool that implements them. Sometimes, a certain model is only available within one of these tools, hence the value of knowing which MLL programs there are available, what are their capabilities, where to find them, and how to use them. The goal of this paper is to provide a quick answer to those questions.

This paper is structured as follows. Section II briefly discusses the most salient aspects of MLL versus traditional learning. The topic of Section III is where to find and how to produce MLL data. Section IV is concerned with EDA software. The tools for partitioning and transforming these data are described in Section V, whereas tools for applying MLL algorithms are covered in Section VI. Section VII describes how to assess predictive performance of MLL methods. After some final words in Section VIII, a set of appendices provides basic instructions on how to download, install and configure the tools[1] used in the sections listed above.

II. SPECIFICS OF MULTILABEL LEARNING VS. STANDARD LEARNING

Before digging deeper into how to face each possible task, in this section the main special characteristics of MLL are going to be outlined.

The essential difference between MLL and standard single-label learning (SLL) lies in the nature of multilabel data itself. Let $X^1, \dots X^f$ be the domains of the f features in a dataset and L the set of distinct labels. The *i-th* data pattern in an SLL dataset can be defined as in (1), whereas the definition of the same data point in MLL would be that of (2).

$$I_i = (X_i, y_i) \mid X_i \in X^1 \times X^2 \times \cdots \times X^f, \quad y_i \in L \quad (1)$$

$$I_i = (X_i, Y_i) \mid X_i \in X^1 \times X^2 \times \cdots \times X^f, \quad Y_i \subseteq L \quad (2)$$

As can be seen, there is no difference in the definition of the set of input features X_i, it being a sequence of values taken from each attribute domain both in SLL and MLL. The changes are found in the second part of the tuple. In SLL, y_i denotes a single label taken from L, whereas in MLL Y_i can be any subset of L, including the empty set and the full set of labels. Y_i usually is represented as a binary vector, made up of zeroes and ones, each component corresponding to a label in L and stating if it is relevant to the instance I_i or not.

[1]The work sessions shown in these appendices are available for download at github.com/fcharte/MLC-Tools-Sessions.

A. MULTILABEL DATA CHARACTERISTICS

Since each data pattern in MLL can be linked to a set of labels instead of exactly one, certain traits specific to multilabel datasets arise. These can be summarized as follows:

1) ACTIVE LABELS

The most basic characterization metrics are those that evaluate how many active labels in average there are in the data. Assuming that each label relevant to an instance is represented as 1 and the remainder ones are set to 0, the number of active labels in I_i can be obtained by simply adding the elements in Y_i. From here, the mean number of active labels throughout the dataset, dubbed label cardinality, can be easily computed. The other common measurement, known as label density, is calculated from the previous one by dividing it by the total number of elements in L.

2) LABEL SETS

Assuming that there are $|L|$ elements in L, Y_i could be any of the $2^{|L|}$ potential combinations. Each one of those is known as labelset, and there are several characterization metrics related to them. Since $2^{|L|}$ can be a huge number depending on the size of L, most labelsets do not occur in a dataset unless it also has a huge amount of instances, hence the interest in knowing how many distinct labelsets there are, how frequent they are, etc.

3) LABEL FREQUENCY

As it happens in traditional data, multilabel datasets can suffer from imbalance, i.e. unequal label distribution. Therefore, several metrics involved in the analysis of label frequency exist. An individual imbalance ratio for each label, with respect to the most frequent one, can be computed. Averaging the ratios from all labels yields a mean imbalance ratio.

4) LABEL RELATIONSHIP

Among the specifics of multilabel data, maybe the most studied topic in the literature is how to measure and take advantage of potential relationships among labels. As a consequence, measurements on how frequently two or more labels appear together, whether the occurrence of one label implies the presence of others, and similar metrics have been proposed.

The formal definition of most MLL characterization metrics, including those in the previous four groups, can be found in the book by Herrera *et al.* [6]. Specific imbalance measures were introduced in [16] and [17].

Observe that all the metrics mentioned above are computed only from Y_i and L, which is where multilabel data differ from traditional data. There are others that combine basic statistics, such as the total number of attributes, amount of labels, labelsets, etc., in order to evaluate the complexity [18] of MLL data.

B. MULTILABEL LEARNING THROUGH TRANSFORMATION METHODS

Since SLL and MLL data have an identical set of input attributes, as seen in (1) and (2), unsupervised ML methods can be applied in exactly the same way to both. By contrast, SLL supervised methods are not able to deal with Y_i, a set of relevant labels, as they expect only one label instead. Therefore, most of the classic regression and classification algorithms cannot be directly applied to MLL data. However, there are dozens of proven SLL methods that would be useful if there were a way to transform (2) to (1).

The first approach to solve MLL was based on this kind of data transformation methods. Many of them are detailed in [6] and [19] and have been implemented in several of the tools later described. Among them, the most popular ones are known as Binary Relevance (BR) [20] and Label Powerset (LP) [21]. They are very straightforward and easy to understand.

BR relies on a binarization process, taking each label in Y_i one by one so that $|L|$ different versions of the dataset are produced. These can be given as input to any SLL algorithm, obtaining a set of binary predictions that must be joined in the end to construct the predicted labelset. The main drawback of this option is that a large set (of size $|L|$) of binary models has to be produced, increasing the time needed to perform the task.

The approach followed by LP is even simpler. Each existing combination of labels Y_i is taken as a class identifier. Therefore, a single multiclass model is enough to deal with the data. The predicted output is easily interpreted as a labelset, splitting it in the correct set of labels. Some drawbacks of this alternative are the potentially huge number of different classes, up to $2^{|L|}$ of them, and the inability to predict label combinations that do not exist in the training set.

C. NATIVE MULTILABEL LEARNING MODELS

Designing models able to learn from multilabel data would allow to deal with some of the specifics previously outlined. However, this is not a trivial task, as the plethora of potential solution proposals published in the literature (see [6], [13], [14]) in late years demonstrates.

Some learning models, such as many kinds of neural networks, are inherently able to deal with any number of outputs, hence the large amount of proposals based on these models [22]–[25]. By having an output neuron for each label in L, the values produced by the neural network can be ordered to provide a label ranking. Then, adjusting a cut-off threshold, the subset of relevant labels is retrieved.

The adaptation process to work with multilabel data has to be designed taking into account the specific architecture of each learning model. For instance, in [26] the classic decision tree is modified so that each leaf contains a set of labels instead of only one. Besides this structural change in the tree, the gain function used to define branching points has to be adjusted as well, considering the existence of several labels per leaf. Similarly, the study of [27] adapts the popular k-nearest neighbors classification algorithm by predicting a subset of L from the labelsets of instances closest to the one being classified.

In addition to the pure method adaptation approach, with results such as the ones referenced above, there are also many proposals mixing the use of SLL models with data transformation. These use the multilabel data to produce ensembles of models [28]–[32], usually through one-vs-all and one-vs-one techniques.

D. ASSESSING MULTILABEL LEARNING PERFORMANCE

Another of the specifics of MLL concerns the way the results produced by any method are assessed. In SLL, the prediction z_i produced by a classifier only can be correct ($z_i = y_i$) or wrong ($z_i \neq y_i$). Conversely, a MLL prediction Z_i can be fully correct ($Z_i = Y_i$), totally wrong ($Z_i \cap Y_i = \oslash$) or partially correct/wrong ($\oslash \neq Z_i \cap Y_i \neq Y_i$).

The usual performance metrics in SLL, such as accuracy, precision, recall, etc., are computed from a confusion matrix formed by the number of true positives, true negatives, false positives and false negatives. The same computations are made in MLL, but having a confusion matrix per label. Due to this, the results can be accumulated and averaged in different ways. The metric can be computed by sample, accumulated and averaged (sample-based metrics) or it can be calculated by label (label-based metrics). In the latter case there are two approaches to aggregate the counters, named micro-averaging and macro-averaging.

In addition to measurements based in the aforementioned confusion matrix, in MLL there is other group known as ranking-based metrics. These involve a ranking made from the confidence levels produced by the model for each label. Using this ranking, a threshold is applied to decide which labels are relevant to the instance and which ones are not.

A comprehensive list of most of the MLL performance metrics and their formulations can be found in [6], [33]. How to compute them using different software packages will be explained later.

E. FACING THE USUAL MULTILABEL LEARNING TASKS

Beyond the introduction to the basics that has just been offered, the rest of this work is focused on how to use the available software tools to complete each of the usual tasks described below in practice. The reader can use the bibliography already cited, especially [6] and the various reviews on the subject [13], [14], [19], to resolve any theoretical aspect. You will also find the guidelines provided in [34] helpful for completing studies in the MLL field.

1) OBTAINING MULTILABEL DATA

In order to conduct any MLL study we will need some multilabel data. Although these data could be self collected in some cases, in most occasions some of the already available datasets will be chosen. Existing multilabel data repositories will be enumerated in Section III, along with the amount

TABLE 1. Public multilabel data repositories.

Name	URL	Format	Type	MLDs
Cometa	cometa.ujaen.es	Several	Generic	> 70
Kdis	www.uco.es/kdis/mllresources/	ARFF1/ARFF3	Generic	> 70
LABIC	computer.njnu.edu.cn/Lab/LABIC/LABIC_Software.html	LABIC	Protein	27
MULAN	mulan.sourceforge.net/datasets-mlc.html	ARFF1	Generic	26
CLUS	dtai.cs.kuleuven.be/clus/hmcdatasets/	ARFF2	Hierarchical	24
MEKA	sourceforge.net/projects/meka/files/Datasets	ARFF3	Generic	15
XML	manikvarma.org/downloads/XC/XMLRepository.html	XML	Large	15
LibSVM	www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html	LibSVM	Generic	7

of datasets they offer and their file formats. Software tools for retrieval as well as generation of datasets will be also described.

2) EXPLORING MULTILABEL DATA TRAITS

A multilabel dataset will have specific characteristics depending on the way its labels are distributed among its data points. Exploratory data analysis tools, such as those shown in Section IV, will allow us to know if labels in L are equally distributed or there is imbalance, whether each instance has a large set of relevant labels (dense, high cardinality) or only a few (sparse, low cardinality), whether these labels are correlated or not, etc. The most appropriate datasets for each study can be chosen based on these traits.

3) PARTITIONING AND TRANSFORMING THE DATA

Once the datasets have been collected, usually they have to be partitioned and sometimes transformed to other file formats. Several of the MLL learning tools are able to partition the data dynamically, just before a learning model is created. In order to perform this task statically so that partitions are stored in files, thus easing further reproduction of the experiments, any of the tools outlined in Section V can be used.

4) LEARNING FROM MULTILABEL DATA

After the previous steps involving data selection and analysis are completed, it is time to use those data to train a model. The same SLL unsupervised methods we are used to will also work with MLL data, since class labels are not taken into account by these algorithms. On the other hand, supervised methods (Section VI) require applying data transformation techniques beforehand or choosing algorithms specifically designed for MLL.

5) EVALUATING MULTILABEL LEARNING PREDICTIVE PERFORMANCE

The outputs produced by MLL methods, predictions about the labels that should be relevant to each data instance, have to be evaluated in order to assess model performance. This step is usually included at the end of the learning task, as part of it, but it can also be conducted independently as will be explained in Section VII.

The following sections will describe how to complete these tasks using various tools. Details on how to download, install and configure these tools are provided in the corresponding appendices.

III. OBTAINING AND GENERATING MULTILABEL DATA

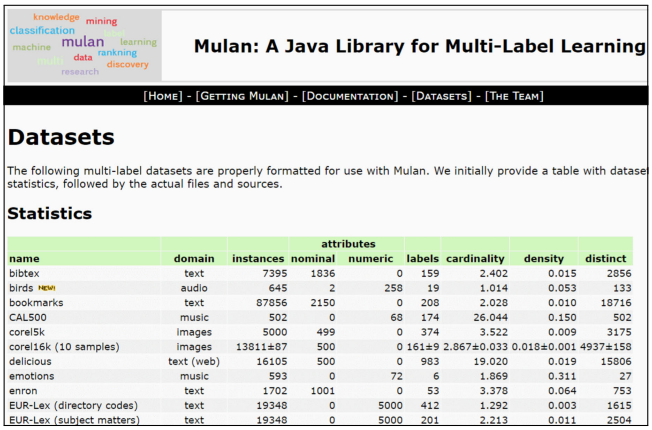
Sometimes, the data to be used in an ML experiment originate in a certain need arisen in a specific field. Therefore, the authors themselves are in charge of collecting, cleaning and formatting the data pieces which, in the end, will make up the dataset. They will need to know the details of the file format they want to generate, unless there is a tool able to produce the dataset from raw data values. In other cases, what researchers want to do is to test a new method they have developed. To do so, they need the proper datasets, which can usually be found in data repositories. If the data have to present specific traits, this selection implies analyzing the characteristics of the available datasets. Alternatively, synthetic data that fits these needs can be generated.

This section begins by providing a list of available MLL data repositories, i.e. web sites from where datasets can be downloaded. The amount of datasets, offered file formats and other details are also provided. Then, software tools specifically designed for managing multilabel datasets are described. The last subsection shows how to generate new datasets by means of different programs.

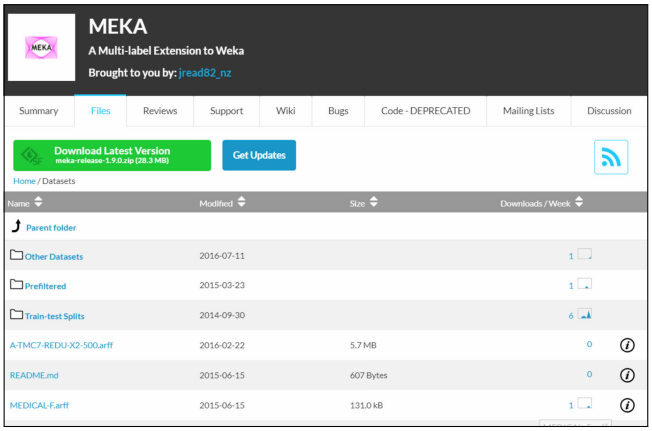
A. DATA REPOSITORIES

When it comes to SLL learning, every researcher knows the UCI Machine Learning Repository [35] as a primary resource of data to work with. Each dataset is tagged with data type, task it is used for, origin, number of samples and attributes, etc. However, only a few of the hundreds of datasets available in this repository correspond to MLL data. This has led to the emergence of specific repositories for multilabel data.

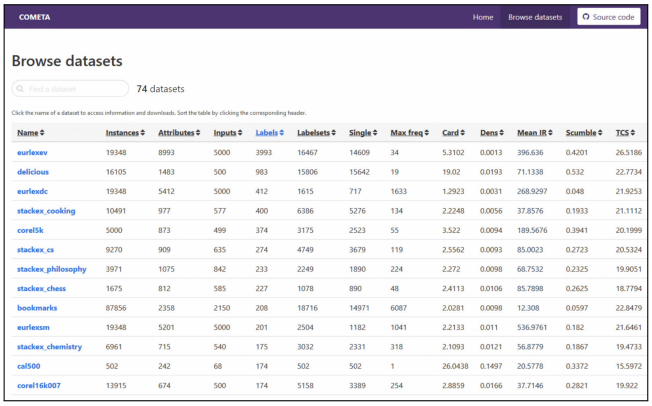
Table 1 enumerates the repositories where most of the publicly available multilabel datasets (MLDs) can be downloaded from. Rows have been sorted by the last column, so that repositories hosting a larger amount of MLDs appear first. Four of their download pages are shown in Fig. 1.



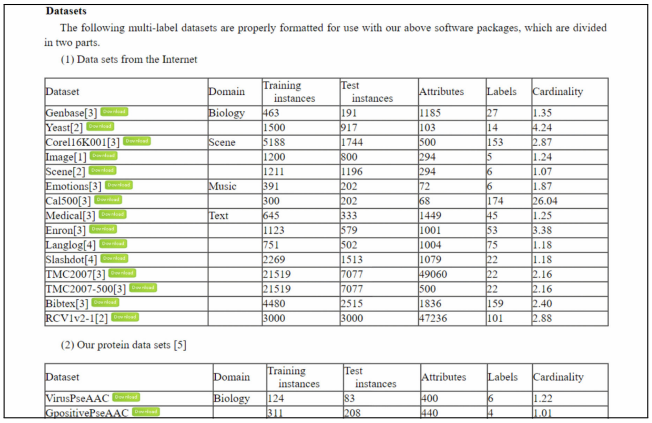
(a) MULAN repository



(b) MEKA repository



(c) COMETA Repository



(d) LABIC Repository

FIGURE 1. Download pages of four MLD repositories.

It should be taken into account that there exists some overlapping among the data available in these repositories. Most of the MLDs on MEKA are also available on MULAN, Kdis holds most of the datasets on MULAN and MEKA, and Cometa provides almost all of the available on the previous repositories together. Certain repositories, such as LABIC, CLUS and XML, provide specific types of datasets. Following, file formats, data type and structure of the MLDs are discussed.

1) FILE FORMATS

Aside from the quantity of MLDs provided, the main difference among the repositories lies on the file formats offered for these MLDs. With the exception of Cometa, a repository holding the same MLDs in disparate formats (MULAN ARFF, MEKA ARFF, Keel ARFF, LibSVM and mldr), the remaining ones are only provided in their own file format.

MLDs are mostly available in repositories linked to software tools such as MULAN [36], MEKA [37] and mldr.datasets [34]. Usually, they are provided in the file format used by the respective tool. For instance, both MULAN and MEKA use the ARFF[2] file format, but the former relies on an external XML file to define which attributes are the labels (dubbed ARFF1 in Table 1), so that they can appear in any position inside the data, whereas the latter uses the ARFF header to report the amount of labels (ARFF3 in Table 1), assuming that they are always at the beginning.

The Clus system used by [38] also has a modified version of ARFF, specific for hierarchical data in this case. LABIC software, such as the proposal in [39], has its own text-based file format, with the indexes of active labels at the beginning and the values of attributes noted as `index:value` pairs, all of them separated by a blank space. A very similar format is XML (Extreme Machine Learning), although in this case a header indicates the number of instances, features and labels. Both, along with LibSVM that is also similar, are sparse file formats. This means that for each instance, only attributes having non-zero values are specified. ARFF supports both dense and sparse representations of data.

2) TYPE OF DATASET

Among all MLDs available in the previous repositories, there is a subset of them usually included in most studies. These are mostly from generic fields, such as text and image classification, and they have heterogeneous traits regarding their dimensionality (number of features and labels), size

[2]Attribute-Relation File Format. This is the same format used in the popular software WEKA.

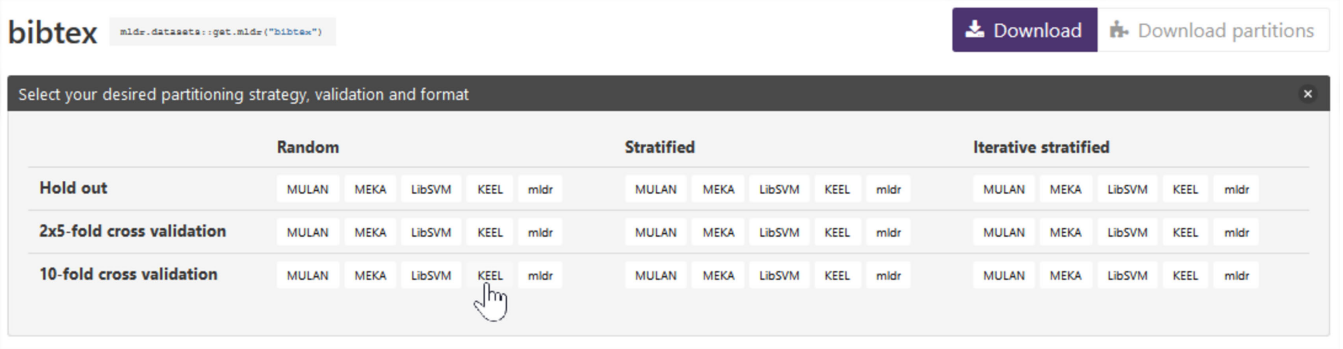


FIGURE 2. Download options offered in the cometa data repository.

(number of instances), imbalance level, etc. All of them can be easily obtained from the sites labeled as Generic in Table 1.

Specific types of MLDs are also available in some of the repositories, as marked in the **Type** column. In addition to generic datasets, the LABIC repository provides a dozen of MLDs related to proteins. These supplement `genbase` and `yeast`, the only two MLDs available in generic repositories which come from the biology field.

The datasets provided in the CLUS repository represent hierarchical multilabel data. That means that the labels in these MLDs conform a hierarchy, so explicit relationships among them exist.

Lastly, the special characteristic of the MLDs in the XML repository is their size. Several of them have millions of input features, millions of labels and millions of instances as well. As the name of the repository (Extreme Classification Repository) states, these datasets are aimed to test methods designed for extreme cases, where the classic ones cannot be applied.

3) STRUCTURE OF THE DATA

Another important aspect to take into account when using the previous data repositories is how the provided MLDs are structured. Sometimes the full datasets, stored in one file, are available, whereas in other cases only partitions (usually two files with train/test instances) can be downloaded.

MULAN and MEKA usually provide full and hold-out train/test partitions of the datasets, but not for all of them. The partitions usually correspond to those used in the papers were the MLDs were introduced. The MULAN repository stores the data files along an XML file needed to read the MLDs from MULAN.

Only hold-out train/test partitions are provided in the LABIC repository, formatted to be read from their software. The way the MLDs were partitioned is not stated, which leads to assume that they have been randomly processed. The XML repository also offers train/test partitions, but in this case they have been processed in a stratified fashion, so that labels keep a similar distribution in training and testing.

Regarding the structure of MLDs, the LibSVM repository is quite heterogeneous. Hold-out train/test partitions are provided for some MLDs, even the small ones such as `scene`, while five folds cross validation are available for others.

Cometa is the most flexible repository regarding the structure of MLDs, since the user can choose (see Fig. 2) among three different partitioning schemes as well as three partitioning strategies. For each dataset an individual page allows to get the full dataset or download it partitioned with hold-out, five folds cross validation or ten folds cross validation schemes. There are random and two stratified partitioning strategies available in all cases.

Depending on the tools we are going to use to conduct a hypothetical study, datasets should be in a specific file format and possibly be prepartitioned. Usually, the first step would be locating and downloading the proper datasets from the previous repositories. Afterward, some of the tools described later can convert these datasets to the desired format.

B. DATA MANAGING TOOLS

Searching for datasets in web repositories, manually downloading them, is just one option. There are some tools able to automate this process, such as the `mldr.datasets` R package introduced in [34]. A general overview of this package is provided in Appendix D. This package is tightly linked to the Cometa data repository, providing the commands needed to enumerate the available datasets, with function `available.mldrs()`, and download them, with `get.mldr()`.

`mldr.datasets` downloads the full datasets in an R file format specified by the `mldr` package [40]. This way the data can be explored directly from the R command line or an R script. Then, by means of other functions in the package, the user can partition and export the data to other file formats. For instance, any dataset (or its partitions) can be written in MULAN, MEKA, KEEL, LibSVM or CSV file formats by means of the `write.mldr()` function.

The Python `scikit-multilearn` [41] library also has built-in functions to access its own data repository. It offers 17 MLDs taken from the MULAN repository, the only considered file format. Function `available_data_sets()` returns a dictionary with the names of each dataset and

the available versions, undivided and split into train and test partitions. Any of them can be loaded onto memory, downloading it from the repository if necessary, by calling `load_dataset()` providing the name of the MLD as argument.

C. SYNTHETIC DATA GENERATION TOOLS

Sometimes machine learning methods are designed to tackle a very specific problem. Although this problem can be variably present in real data, a detailed analysis is not always possible if disparate traits interacting between them exist in the data patterns. This is the reason why so many researchers also include synthetic data in their studies. These data show exactly the characteristics the proposed method aims for, allowing a better adjustment of parameters. Once this work is done, real world datasets are usually also included in the experiments.

Most of the available tools to produce synthetic multilabel data can be grouped into one of two categories: generic or specialized. There are plenty of published articles that use the latter category to obtain an MLD with very specific characteristics, usually those the proposed MLL algorithm is supposed to solve. On the other hand, tools in the former group allow the users creating MLDs with disparate traits, depending on their needs. The following three are among the existing alternatives.

1) `mldr_from_dataframe()`

This function is provided by the R `mldr` [40] package. It can be used from the R command line, needing two parameters: an R `data.frame` containing the data and a vector stating which columns act as labels. Since instances are provided as a `data.frame`, they can have any desirable characteristics. All the computing and statistical power of R is available to model attribute values and label relevance. The generated MLD can be saved using several file formats (see the `mldr.datasets` package description).

2) MLDATAGEN

A very simple to use web tool (http://sites.labic.icmc.usp.br/mldatagen/) able to produce MLD instances following two strategies, named hypercube and hypersphere, as defined in [42]. In addition to the strategy, the user can choose the amount of relevant, irrelevant and redundant features, as well as the number of labels per instance and the level of noise. The generated MLD can be downloaded in MULAN format, so that most MLL algorithms can work with it.

3) `ml_generator()`

This is a MATLAB function which also relies on an hypercube strategy to produce the instances of the synthetic MLD. The tool is introduced in [43] and can be downloaded from http://www.aic.uniovi.es/ml_generator/. Once the function has been loaded into MATLAB, the user can call it stating the amount of input attributes, number of instances and labels, the desired label cardinality and label dependency levels, number of hyperplanes to use, etc. Some of these parameters, such as label cardinality and the dependency level, are only suggestions to the algorithm, which will try to get as closer as possible to these values.

IV. EXPLORATORY DATA ANALYSIS TOOLS

One of the key aspects in designing appropriate machine learning methods is understanding the data you are working with, thus the importance of having EDA tools at your disposal. When dealing with multilabel data, knowing in advance if the labels are balanced or not, if label cardinality and density are high or low, how many different labelsets there are, etc., may allow us to choose one among the available learning methods and properly adjust its parameters, as well as to decide if a preprocessing step is necessary.

EDA tools specifically designed for multilabel data are scarce when compared with the multitude available for dealing with classic single-label data. They can be grouped into three categories:

- Programmatic tools: A program has to be written to load and analyze the data before a set of characterization metrics can be retrieved or a plot can be generated.
- Command line tools: Loading and analysis of data can be performed in an interactive fashion, from a REPL (Read-Eval-Print-Loop) prompt, obtaining an immediate answer.
- GUI tools: Provides a higher-level interface aimed at non-experienced users, so that they can explore the data traits by simply clicking some options.

A single piece of software may provide several interfaces for accessing its EDA tools. For instance, the `mldr` [40] package has a command line interface, whose syntax can be used in a programmatic way inside scripts, and also a GUI, whereas MEKA [37] provides both programmatic and GUI EDA options. Table 2 summarizes the type of interface provided by each EDA tool.

TABLE 2. Type of interface provided by the EDA tools.

Lang.	Tool	Programmatic	Command line	GUI
Java	MEKA	✓	✗	✓
R	mldr	✓	✓	✓
Java	MULAN	✓	✗	✗
Python	scikit-multilearn	✓	✓	✗

Another fact that differentiates existing tools is the set of data traits they provide. There are very common characteristics, such as label density and cardinality, found in all software packages. However, some more specific metrics, such as those related to imbalance or concurrence levels, are not so common. Table 3 shows the EDA tools that can be used to compute each metric. The following subsections portrait the use of the major multilabel EDA tools.

TABLE 3. Data traits provided by the EDA tools reviewed in this section.

Metric	MEKA	mldr	MULAN
# inst./attr.	✓/✓	✓/✓	✓/✓
# labels	✓	✓	✓
# labelsets	✗	✓	✓
# unique labelsets	✗	✓	✗
Cardinality	✓	✓	✓
Density	✗	✓	✓
MeanIR	✗	✓	✗
Scumble	✗	✓	✗
TCS	✗	✓	✗
Label frequencies	✓	✓	✓
Label IR	✗	✓	✗
Label Scumble	✗	✓	✗
Labelset frequencies	✗	✓	✓

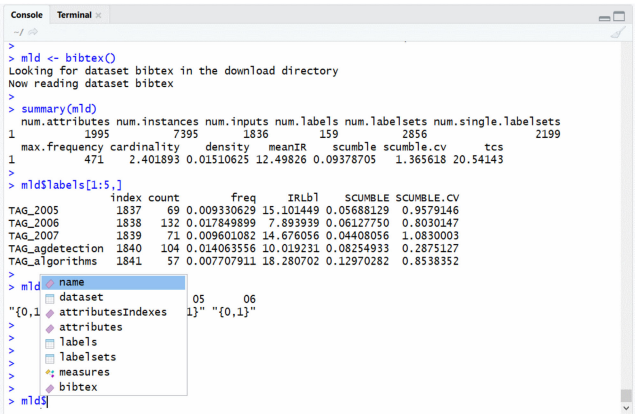


FIGURE 3. Sample EDA session from the R command line using the mldr package.

A. MULTILABEL EXPLORATORY DATA ANALYSIS JAVA TOOLS

MULAN [36] is arguably the most used MLL tool. It does not provide the user with a GUI as it is designed to be used programmatically. However, it is quite easy to write a program that loads an MLD and uses the `Statistics` class to calculate some statistics. This class is the main EDA option in MULAN. Once the `calculateStats()` method has been called, data disparate traits can be retrieved including label cardinality and density, label frequencies, distinct labelsets, etc. The `Statistics` class also offers methods aimed to compute label concurrence and label correlation matrices.

Although MEKA [37] has an easy to use GUI, it is mostly aimed to design experiments rather than to facilitate EDA tasks. The MEKA Explorer shows, once a dataset has been loaded, the number of attributes and labels as well as basic attribute statistics as they are chosen from the list. After running any experiment, the results panel informs the user about the label cardinality in the training and testing sets. Lastly, the Visualize panel provides paired plots for all the attributes, including the labels, so a basic intuition about their distribution can be obtained.

Another Java tool aimed to perform EDA tasks, although much less known than the previous ones, is MultiLabel Dataset Analyzer [44]. This Java program provides a GUI similar to that in MEKA. However, it is focused on providing MLD characterization metrics as well as several types of plots summarizing label frequencies, label co-occurrences, etc.

B. MULTILABEL EXPLORATORY DATA ANALYSIS R TOOLS

The main multilabel EDA tool for R is inside the `mldr` [40] package. It provides both a command line and a GUI interface to accomplish most tasks. Traits of the data are obtained through the function in charge of loading the MLDs. It is also able to deal with data sets obtained through the `mldr.datasets` [34] package or downloaded from the Cometa multilabel data repository.

Once an MLD has been loaded, it appears to R as an "`mldr`" class object. This object has several attributes containing data traits. A summary of some of them can be retrieved through the usual `summary()` function, as shown in Fig. 3. The remaining attributes[3] provide the following information:

- `attributes`: A character vector holding the name and a range of values for each attribute in the MLD, including the labels.
- `labels`: An R `data.frame` with as many rows as labels within the MLD. Each row provides the name of the label, its position (column), absolute and relative frequency, imbalance and concurrence levels.
- `labelsets`: A named array with an entry for each distinct label set in the MLD, stating the label combination and number of occurrences.
- `measures`: A list holding all basic metrics of the MLD, including number of attributes, labels, labelsets, label cardinality and density, average imbalance and concurrence levels, etc.

In addition to attributes holding data traits, an "`mldr`" object also has some methods useful to explore the MLD such as the `concurrenceReport()` method. It analyzes the most salient interactions between frequent labels and minority ones, producing a textual report and a circular plot showing these interactions. Another interesting method is `plot()`, able to produce up to seven types of plots from the MLD. These include histogram of labels, labelsets and label cardinality, label and labelset bar plots, attribute type pie chart, etc. All of them are thoroughly explained in [40].

All the functionality described above, from loading data to retrieve generic metrics, label and labelset details, the concurrence report and most available plots, is also accessible via the package's integrated GUI. Users are offered the option to print and save the information shown in the interface, both for tables and plots, as well as to filter and search the tables, as can be seen in Fig. 4.

[3]Aside from informative members, the "`mldr`" object also has a `dataset` attribute that contains the actual MLD data. It can be used to perform any other exploratory action over attribute values and labels.

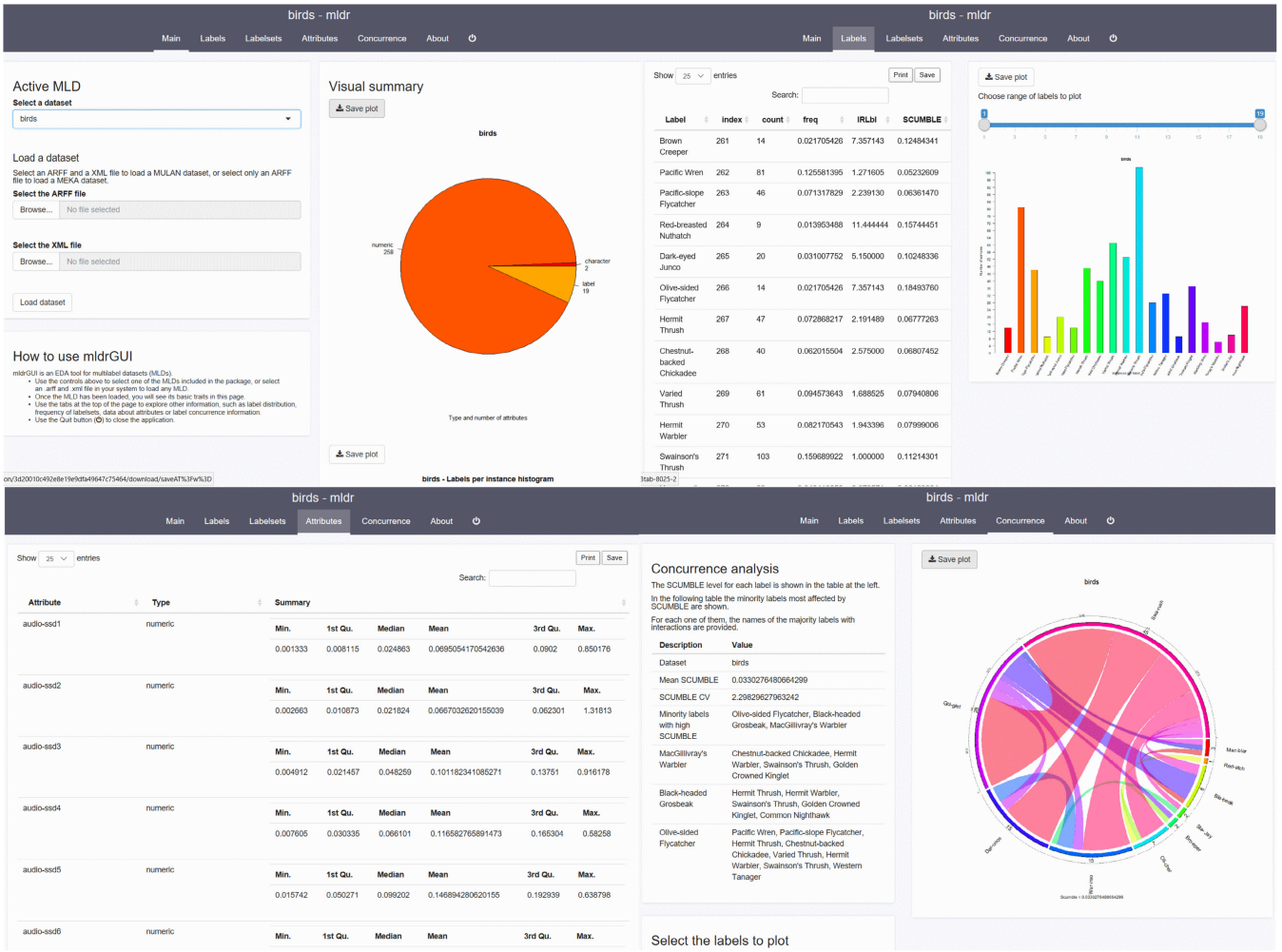


FIGURE 4. Some of the pages provided by the `mldr` package's GUI.

C. MULTILABEL EXPLORATORY DATA ANALYSIS PYTHON TOOLS

As far as we know, there are no specific tools for multilabel data exploration in Python. Although Python library `scikit-multilearn` [41] is capable of loading MLDs, using them to train different classifiers, it lacks features specific to perform exploratory analysis. Therefore, metrics such as label cardinality, label density, amount of distinct labelsets, imbalance ratios, etc., have to be manually computed. For most characterization metrics this is a simple procedure, but demands some Python programming knowledge from the user.

The `scikit-multilearn` `load_dataset()` function returns attribute and label values as well as attribute and label names. The former are a couple of sparse matrices, so the methods in the `numpy` library can be applied to compute sums per columns or rows. This way label frequencies can be obtained, in a first step, and then many other measurements as a result of simple arithmetic operations, as shown in Fig. 5.

On the other hand, what the `scikit-multilearn` package does offer is various functions for analyzing the relationships and co-occurrences between labels. For instance, a list of label pairs, stating the number of times each pair appear together in the MLD, can be generated with `LabelCoocurrenceGraphBuilder`. This information can be also plotted.

V. PARTITIONING AND TRANSFORMING MULTILABEL DATASETS

Once the data is locally available, usually other data manipulation operations are needed, mainly data partitioning and transformation. These tasks, which take a dataset as input and produce one or more output datasets, can be fulfilled through several of the software packages summarized in the appendices.

A. HOW TO PARTITION A MULTILABEL DATASET

Dividing an MLD into pieces, so that a fraction of the samples can be used to train a model and the remaining

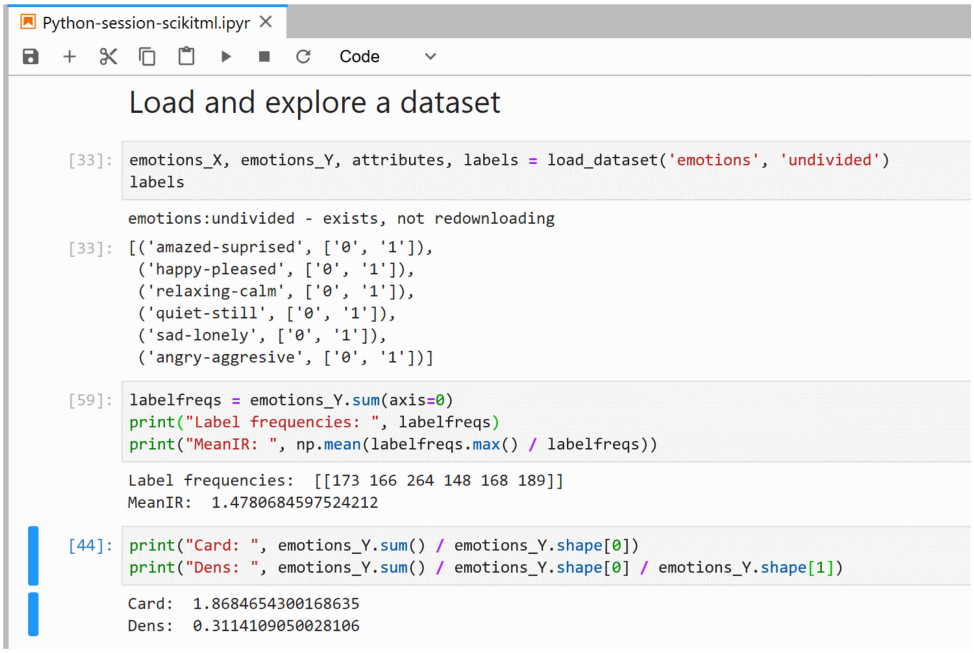


FIGURE 5. Computing some simple characterization metrics in Python.

ones to evaluate it, can be accomplished following different approaches. The simplest one, but nonetheless usual in many studies, consists in randomly picking each instance to either train or test the model. This way has a clear inconvenient, since the distribution of labels in the training set could be very different of that in the test set. In that case, the model would be biased to the more frequent labels in detriment of those rarer or even never seen in training.

Stratified partitioning of data samples is a straightforward way of balancing classes among training and testing partitions, but only for standard classification problems. Since each pattern is associated to only one class, it is trivial to distribute them among train and test subsets. However, in the multilabel case there are several class labels linked to most samples. Therefore, choosing one sample for training or testing incorporates not only the label of interest, but also all the other active labels in that sample. Because of this reason more sophisticated partitioning methods have been proposed in the literature, such as the ones proposed by [45] (iterative stratified) and [18] (stratified), able to deal with this complexity.

The three aforementioned algorithms, random, stratified and iterative stratified, are available in the `mldr.datasets` R package [34]. In order to apply them, the input dataset has to be previously loaded in R. Then, it is only a matter of calling one of the six available functions: `random.holdout()` or `random.kfolds()` for random partitioning, `stratified.holdout()` or `stratified.kfolds()` for the stratified approach, and `iterative.stratification.holdout()` or `iterative.stratification.kfolds()` to use the iterative stratification strategy. For further details, see the example in Appendix D.

MULAN [36] is built on top of Weka [3], and random partitioning can be performed through the `Filter` utility class of the latter. In addition, MULAN itself provides the `IterativeStratification` class in the `mulan.data` package. It conducts the iterative stratified partitioning described in [45] (see example session in Appendix A). The `crossValidate()` method (`Evaluator` class) automatically builds random partitions, so that cross validation can be accomplished without manually splitting the data.

The MEKA [37] GUI allows the user to choose between hold-out and cross validation random partitioning schemes, as well as getting prepartitioned train/test splits for any experiment. As far as can be inferred from the documentation, there is no way to partition a dataset and store the folds for further use, they can only be used to train and test a MEKA classifier through the `Evaluation.cvModel()` method.

The `utiml` R package [46] also provides the user with two partitioning functions, but only random hold-out (`create_holdout_partition()`) and k-folds (`create_kfold_partition()`) strategies are considered.

B. COMMON DATA TRANSFORMATIONS: BINARY RELEVANCE AND LABEL POWERSET

Data transformation is the most common approach to multilabel learning problems. Among the transformation methods proposed in the literature, BR [20] and LP [21] are

undoubtedly the best known. As a result, their availability in several software packages is expected.

For R users, these transformations can be found in the `mldr` [40] package. The `mldr_transform` method takes a dataset as its first parameter. The second argument establishes the transformation to be applied, "`BR`" or "`LP`". The following example applies both of them to the same dataset, obtaining a list of binary datasets and a multiclass dataset. Once the transformed data have been obtained, any of the classifiers available in the plethora of R packages, such as `caret` [1], may be used.

```
> library(mldr)

> emobr <- mldr_transform(emotions, type = "BR") # Returns list binary datasets
> emolp <- mldr_transform(emotions, type = "LP") # Returns multiclass dataset
```

Example 1. Transformation methods in the mldr package.

Some other R packages, such as `mlr` [47] and `utiml` [46], as well as the MEKA [37] software, also implement BR and LP transformations. However, their functions do not return the transformed data but use it to train the corresponding classifiers. The same is applicable for the `scikit-multilearn` [41] Python library. For instance, the following lines would use the BR transformation and a support vector machine to train a multilabel classifier using the `utiml` package.

```
> library(utiml)

> # Trains six SVM binary classifiers (one per label)
> brclassifier <- br(emotions$train, "SVM")

> # Uses the classifier to obtain predictions
> predictions <- predict(brclassifier, emotions$test)
```

Example 2. Transformation methods in the utiml package.

The `mulan.transformations` package in MULAN exports four classes aimed to perform different transformations. The first one is in charge of conducting the BR transformation, `BinaryRelevanceTransformation`. The second one, `LabelPowersetTransformation`, is responsible for the LP transformation. Both have a `transformInstances()` method that takes the original set of multilabel instances and return the transformed ones.

C. APPLYING OTHER TRANSFORMATIONS

In addition to BR and LP, there exist other data transformations for multilabel data. Some of them are available in certain software packages. This is the case of REMEDIAL [48], an algorithm aimed to resample a dataset so that instances having highly imbalanced labels are decoupled. The `remedial()` function in the `mldr` R package implements this algorithm, and can be used as follows:

As can be seen, the dataset has more samples once the method is applied, but the number of labelsets, label cardinality, label density and SCUMBLE metric have been reduced.

```
>
> summary(emotions)
  num.attributes num.instances num.inputs num.labels num.labelsets num.single.
          labelsets
1             78           593         72          6            27
                        4
  max.frequency cardinality   density   meanIR    scumble scumble.cv      tcs
1            81    1.868465 0.3114109 1.478068 0.01095238    1.26456 9.364262
>
> summary(remedial(emotions))
  num.attributes num.instances num.inputs num.labels num.labelsets num.single.
          labelsets
1             78           815         72          6            19
                        2
  max.frequency cardinality   density   meanIR      scumble scumble.cv      tcs
1           257    1.359509 0.2265849 1.478068 0.0005623093   2.566903 9.012865
>
```

Example 3. Emotions traits before and after applying REMEDIAL algorithm.

VI. RUNNING MULTILABEL LEARNING ALGORITHMS

A supervised algorithm relies on label data, the information that is not available at test time, to produce and/or adjust the learning model. Standard learning methods, designed to work with one output (class or target value) only, are not suitable for this task since multilabel patterns have multiple binary outputs.

Most MLL supervised methods fall into one of two approaches, as explained in subsections II-B and II-C. The creation of binary classifier ensembles to cope with this job might be the most usual technique. The major difference among the tools described below is in the set of algorithms they implement. Some of them are mainly focused on transformation methods, whereas others also include a large collection of native MLL methods. Table 4 (see abbreviations at the bottom) enumerates the algorithms provided by each software package. The details on how to use them are given in the following subsections.

A. MULTILABEL LEARNING WITH JAVA

The first general purpose MLL libraries were written using the Java language. It does not come as a surprise, since WEKA [3] was also written in Java and both MULAN [36] and MEKA [37] are built on top of WEKA's foundations. The main difference between these two software tools lies in the approach by which the user can access their functionality. MULAN is a purely programmatic tool, a library of classes to be used from the user's programs. By contrast, MEKA is more similar to WEKA and provides a GUI as well as an Application Programming Interface (API). Many of the existing MLL algorithms, following both the transformation and the adaptation approaches, can be found implemented in Java. On the contrary, as is shown in following sections, adaptation-based MLL algorithms are scarce in other programming languages such as R or Python.

1) THE MULAN LIBRARY

Designing an MLL experiment with MULAN implies writing a Java program, importing the MULAN packages and using the classes supplied by them. Then, the source code needs to be compiled and run from the command line, as usual with any other Java console application. A specific variation of the

TABLE 4. Summary of MLL algorithms implemented in each software package.

Algorithm	meka	mlr	mulan	scikit-ml	utiml
Transformation					
Adaboost.MH	✗	✗	✓	✗	✗
BR	✓	✓	✓	✓	✓
BR+	✗	✗	✗	✗	✓
BRS	✗	✓	✓	✗	✓
CC	✓	✓	✓	✓	✓
CDN	✓	✗	✗	✗	✗
CLR	✗	✗	✓	✗	✓
CT	✓	✗	✗	✗	✗
CTRL	✗	✗	✗	✗	✓
DBR	✗	✓	✗	✗	✓
EBR	✗	✗	✗	✗	✓
ECC	✓	✗	✓	✗	✓
EPS	✓	✗	✓	✗	✓
HOMER	✗	✗	✓	✗	✓
LIFT	✗	✗	✗	✗	✓
LP	✓	✗	✓	✓	✓
NS	✗	✓	✗	✗	✓
PRUDENT	✗	✗	✗	✗	✓
PS	✓	✗	✓	✗	✓
RAkEL	✓	✗	✓	✓	✓
RDBR	✗	✗	✗	✗	✓
RPC	✗	✗	✓	✗	✓
Adaptation					
BP-MLL	✓	✗	✓	✗	✗
BR-kNN	✗	✗	✓	✓	✗
DBPNN	✓	✗	✗	✗	✗
IBLR-ML	✗	✗	✓	✗	✗
ML-ARAM	✗	✗	✗	✓	✗
ML-kNN	✗	✗	✓	✓	✓
MLTSVM	✗	✗	✗	✓	✗
MMPLearner	✗	✗	✓	✗	✗
Random ferns	✗	✓	✗	✗	✗
Random forest	✗	✓	✗	✗	✗

BP-MLL=Back Propagation MLL, BR=Binary Relevance, BRS=BR Stacking, CC=Classifier Chains, CDN=Conditional Dependency Network, CLR=Calibrate Label Ranking, CT=Classifier Trellis, CTRL=Controlled Label Correlation, DBR=Dependent BR, DBPNN=Deep Back Propagation NN, EBR=Ensemble BR, ECC=Ensemble CC, EPS=Ensemble PS, HOMER=Hierarchy multilabel Classifiers, IBLR-ML=Instance Based and Logistic Regression, LIFT=Learning Specific Label Features, LP=Label Powerset, MMPLearner=Multiclass Multilabel Perceptron, NS=Nested Stacking, PRUDENT=Pruned Confident Stacking, PS=Pruned Sets, RAkEL=Random k Labelsets, RDBR=Recursive Dependent BR, RPC=Ranking Pairwise Comparison

ARFF file format is defined by MULAN, along with an XML file holding label information. The `mulan.data` package provides the classes in charge of loading and manipulating MLDs.

Once the data has been loaded into an object `MultiLabelInstances`, the steps to perform an MLL experiment using MULAN are the undermentioned:

1) Create an object from any of the classes derived from `MultiLabelLearnerBase`. All of them are grouped into several subpackages inside the `mulan.classifier` package. Depending on the type of classifier, the constructor will need a specific set of parameters, usually:
   - **Transformation methods:** The classes in `mulan.classifier.transformation` implement transformation-based models. They will require an underlying binary/multiclass classifier, a WEKA object that has to be created and given as parameter to the constructor.
   - **Adapted methods:** The remainder classes will take a series of specific arguments aimed to configure the MLL algorithm.
2) Call the `build()` method providing the set of training instances as argument. This will adjust the classifier's internal parameters according to the seen patterns.
3) Use the `makePrediction()` method to obtain label predictions for individual test instances.
4) To assess the classifier performance the `evaluate()` method in the `Evaluator` class is used. It takes the trained classifier, test instances and a list of metrics to compute as parameters.

The work session shown in Fig. 6 demonstrates how to use MULAN to train and evaluate the performance of two MLL methods with same data. Additional details on how to install and use MULAN can be found in Appendix A.

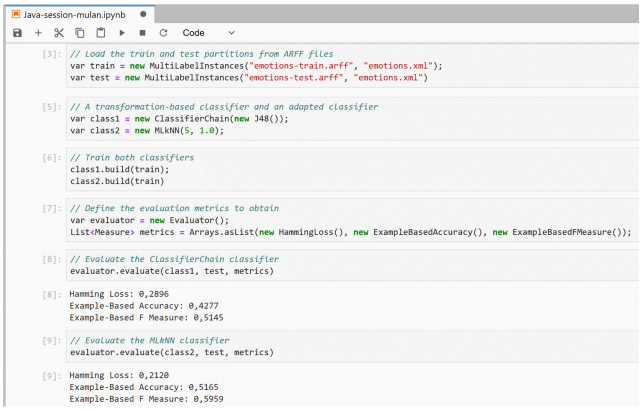


FIGURE 6. Train and evaluate two MULAN classifiers.

2) THE MEKA APPLICATION

Unlike MULAN, which is made mostly for Java programmers, MEKA [37] is a full-fledged application useful to any practitioner. Its GUI is made up of several components. The MEKA Explorer makes possible to load an MLD, choose a classifier, run it and obtain performance results. By means of the MEKA Experimenter, more complex experiments can be designed, involving several datasets and classifiers. All these tasks can also be accomplished from an OS terminal, simply by issuing a command with the appropriate options (see Appendix B).

The MEKA GUI is very similar to the one in WEKA [3], so WEKA users would get used to it in no time. After launching MEKA, a small window with a couple of buttons and a menu will appear. Assuming that the user is interested in performing a single experiment, it would open the MEKA Explorer and follow the steps detailed below:

1) Load any MLD in ARFF file format through the File>Open option. Unless it is a MEKA ARFF, state which attributes act as labels.

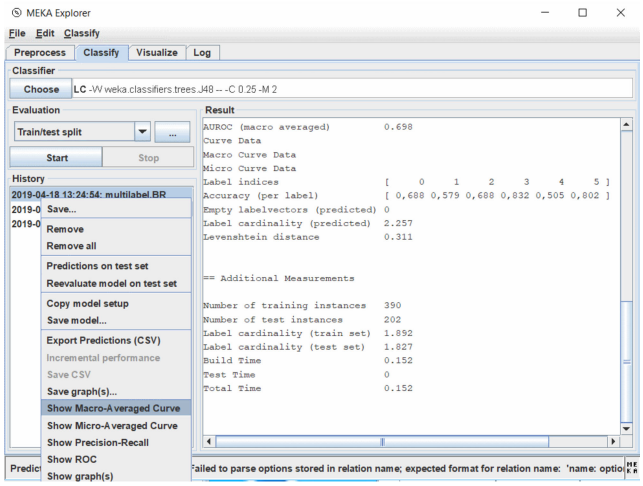


FIGURE 7. Running classifiers through the MEKA explorer.

2) Choose the classifier to be used, as well as the validation scheme, in the Classify page. The Start button in this page will run the experiment and show in-depth performance data in the Result panel.
3) Use the pop-up menu of any of the already run experiments (see Fig. 7) to save the model, export the predictions to a CSV file, show disparate plots, etc.

In addition to those pointed out in Table 4, MEKA also incorporates many CC-derived methods, such as Bayesian CC (BCC), MonteCarlo CC (MCC), Probabilistic CC (PCC), among others. Moreover, it provides a MULAN proxy which makes available to the user all MULAN algorithms. Table 4 only indicates the methods natively implemented in MEKA.

B. MULTILABEL LEARNING WITH R

Thousands of pre-built packages are available to R users. They can be installed from either CRAN, Bioconductor or other repositories. Many of these packages provide learning algorithms, specifically classification and regression methods. However, most of them aim to deal with standard tasks rather than multilabel ones. Some packages, such as `caret` [1] and `mlr` [47], act as wrappers around many of the former, offering a unified way of performing learning tasks.

Native MLL methods are somewhat scarce in R. Only a handful of them have been implemented. On the contrary, many of the transformation-based methods in the literature are available for R users. Most of them can be found in the `mlr` and `utiml` [46] packages described below.

1) THE MLR PACKAGE

This is one of the most popular R packages for ML tasks, including data preprocessing, data resampling, clustering, regression and classification methods, etc. Although recently it gained some MLL capabilities [49], the set of learning methods it provides is quite small as can be seen in Table 4. Only two adaptation methods and five transformation-based algorithms are implemented.

Performing an MLL experiment with `mlr` is straightforward. Once the multilabel data is stored into an R `data.frame` (see details about data types in Appendix E), the steps are as follows:

1) Create a task from the data by means of the `makeMultilabelTask()` function, stating which columns hold the labels.
2) Configure the learning algorithm. Two alternatives can be used:
   - **Native methods:** Call `makeLearner()` using either "`multilabel.randomForestSRC`" or "`multilabel.rFerns`" as the only parameter, depending on the algorithm to be used.
   - **Transformation methods:** Start by calling the `makeLearner()` function to instantiate any of the methods available in `mlr` to serve as binary classifier. Then, call one of the five `makeMultilabelXXXWrapper()` functions to configure the transformation.
3) Train the classifier by feeding it the data held in the task. The `train()` function takes the value returned by either `makeLearner()` or any of the `makeMultilabelXXXWrapper()` functions as first parameter. The second one would be the value returned by `makeMultilabelTask()`.
4) Obtain predictions for new instances through the `predict()` function. It accepts the trained classifier as first argument and the instances to be processed as second argument.

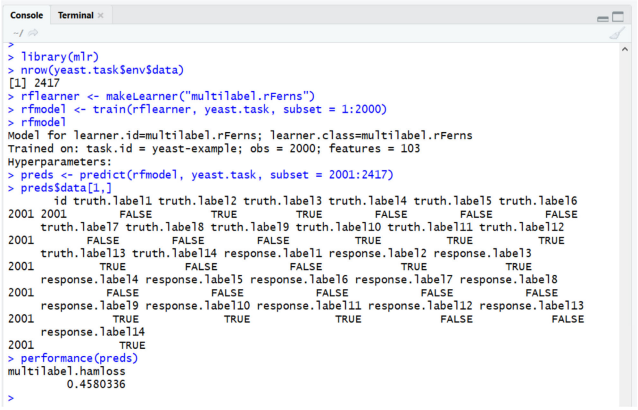


FIGURE 8. Sample sesion using the mlr R package.

The function in step 4 returns an object whose `data` member holds the predictions. For each instance, the set of true labels along with the predicted ones is provided, so that the performance of the model can be evaluated. The `performance()` function is in charge of this. A sample `mlr` work session is shown in Fig. 8. It uses the multilabel task `yeast.task` included as example in the package.

2) THE UTIML PACKAGE

This package [46] is the most recent addition to the R MLL learning portfolio. Unlike `mlr`, it has been designed from the beginning to accomplish MLL tasks. Aside from preprocessing and basic partitioning methods, `utiml` provides 22 functions to train different classifiers (see Table 4). All of them are transformation-based MLL algorithms, with the only exception of ML-kNN.

The usual steps to be taken to complete a predictive task with this package are as follows:

1) Get the MLD using the services in the `mldr` package and split it as desired.
2) Give the training instances to one of the functions which implement the classifiers, e.g. `br()`, `ecc()`, `mlknn()`, etc. A base classifier to process each binary problem can be also specified.
3) Obtain label predictions for each test sample by calling the `predict()` function, as usual. A threshold can be applied to the predicted values in order to generate a label bipartition.
4) Call the `multilabel_evaluate()` function to assess classifier performance. In addition to the set of test instances and the predictions returned by `predict()`, this function also takes as parameter a vector stating the evaluation metrics to be computed.

A sample `utiml` work session is shown in Fig. 9, training an ensemble of pruned sets classifier to process the `emotions` MLD. More details on how to use this package are provided in Appendix F.

```
Console  Terminal
~/
>
> library(utiml, quietly = TRUE)
> partitions <- create_holdout_partition(mldr::emotions, c(train = 0.7, test = 0.3))
> class1 <- eps(partitions$train, base.algorithm = "SVM")
> class1
Ensemble of Pruned Set Model

Call:
eps(mdata = partitions$train, base.algorithm = "SVM")

Models: 10
Instance by models:  312
Prune: 3
Strategy: A
B value: 2
> preds <- predict(class1, partitions$test)
> preds[1,]
  amazed-suprised happy-pleased relaxing-calm quiet-still sad-lonely angry-aggresive
11               0     2.120925     2.327697           0          0               0
> multilabel_evaluate(partitions$test, preds)
        accuracy average-precision              clp         coverage               F1      hamming-loss
       0.6194757         0.7911361       0.0000000        1.7752809        0.7038390         0.1853933
       macro-AUC          macro-F1  macro-precision     macro-recall     margin-loss         micro-AUC
       0.9051349         0.7170753       0.6733649        0.7809119        1.0842697         0.9103865
        micro-F1  micro-precision     micro-recall              mlp         one-error         precision
       0.7242340         0.6666667       0.7926829        0.0000000        0.3146067         0.6737828
    ranking-loss            recall  subset-accuracy              wlp
       0.1658864         0.7902622       0.3707865        0.0000000
>
```

FIGURE 9. Sample session using the utiml R package.

C. MULTILABEL LEARNING WITH PYTHON

In recent years Python has been rising among machine learning tools, largely thanks to the popularity of libraries such as `numpy`, `pandas` and, above all, `scikit-learn`. Although existing methods in the latter library could be used to face MLL, by applying basic data transformations and using binary or multiclass classifiers, there is no native support for MLL in `scikit-learn`. This gap has been filled by the `scikit-multilearn` library [41], built on top of the previous one.

Some datasets from MULAN are hosted in a specific `scikit-multilearn` repository. They can be downloaded from this repository and loaded onto memory by calling the `load_dataset()` function. Any other MLD, as long it is provided in ARFF format, can be read using the `load_from_arff()` function. Once the data is available, the usual steps to train and evaluate the MLL algorithms included in `scikit-multilearn` are as follows:

1) Use any of the functions within the `adapt` or `problem_transform` modules, such as `MLkNN()` or `BinaryRelevance()`, to configure the desired classifier.
2) Train the classifier by calling its `fit()` method. It will receive as parameters the features and labels of training instances.
3) Obtain label predictions for test data by means of the `predict()` method. It takes the features of test samples as input and returns the predicted outputs.
4) Compute performance metrics through the functions available in the `sklearn.metrics` module.

Although only a handful of transformation-based and adapted algorithms are implemented in this Python library, it also provides a wrapper that allows the user to call any MEKA classifier. The capability to use Keras in order to build deep learning-based multilabel models is also included. A sample `scikit-multilearn` work session is shown in Fig. 10, where two classifiers are trained with the `emotions` MLD. Additional details about how to install and use this Python library are given in Appendix G.

D. MULTILABEL LEARNING ALGORITHMS IMPLEMENTED IN OTHER LANGUAGES

Beyond the functionality found in the software packages already described, many authors provide their own implementations of many published MLL methods. Reference implementations of algorithms, those written by the authors themselves, can be found in disparate languages. Usually the link to these resources is given in the paper or the book where the algorithms are described. For instance, reference implementations of some techniques explained in [6] are hosted at https://github.com/fcharte/SM-MLC. Many of these links can be also found in MLL reviews such as [13]–[15], [19].

Two outstanding websites in this regard are those of research teams LABIC (http://computer.njnu.edu.cn/Lab/LABIC/LABIC_Software.html) and LAMDA (http://lamda.nju.edu.cn/Data.ashx). The former hosts MATLAB packages for performing multilabel linear feature extraction, and offers C++ packages with several MLL implementations relying on SVMs and kNN. The list of software methods provided in the latter page is quite long, mostly MATLAB source code modules although there also are Python, Java and C++ files. These include a remarkable amount of MLL classifiers, as well as MLL combined with other techniques such

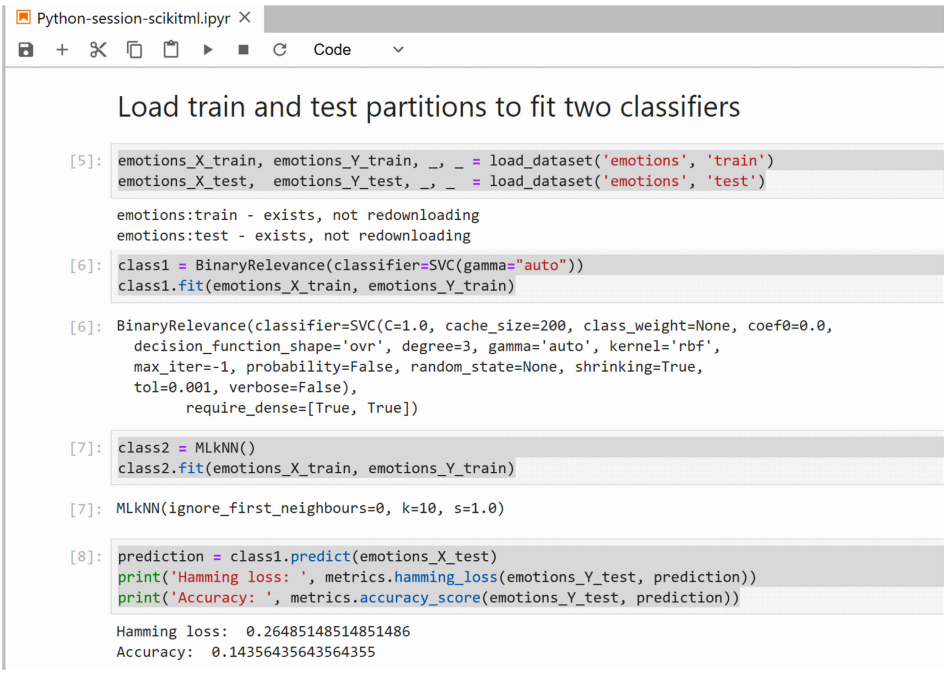


FIGURE 10. Training and testing of two classifiers using scikit-multilearn.

as multiinstance learning, multiview learning, reinforcement learning, etc.

VII. EVALUATING MULTILABEL LEARNING PREDICTIVE PERFORMANCE

Most of the learning algorithms specifically designed to work with multilabel data have a goal, predicting the set of labels for unseen instances as accurately as possible. As stated in Section II, this prediction can be fully correct or fully wrong, but also any point in between. Because of this a large collection of performance metrics, more than twenty, aimed to assessing the models exists.

In order to compute the evaluation metrics for a set of instances, we will usually need the sets of ground truth labels and of predicted ones. Predictions can be a binary partition, simply stating which labels are relevant or not for each instance, or real values, such as confidence levels. In the latter case, a label ranking can be obtained, and it can be converted to a binary partition by simply applying a threshold.

Although all performance metrics are defined in the literature, not all software tools implement them in the same way. The evaluation functions of a specific package could be able to process the representation of results produced only by itself. Moreover, not all existing metrics are provided by all tools. Table 5 summarizes the ones available in each software package among the most common metrics. There are others not reflected in this table due to being very basic and common, such as the amounts of true positives and true negatives, or being too specialized and quite rare in MLL. The following subsections portrait the procedures to assess predictive performance through several programs.

A. EVALUATING PERFORMANCE FROM JAVA

The `mulan.evaluation.measure` package in MULAN defines a large assortment of classes, aimed to compute a performance metric[4] each one of them. Since they derive from a few common classes, and implement the `Measure` interface, their behavior is always the same. Essentially, they provide an `update()` method that gets a prediction and a set of ground truth labels, a `reset()` method to clear previous computations, and a `getValue()` method which returns the current metric value.

MULAN does not facilitate a direct mechanism to compute a collection of metrics from a set of predictions, although the individual classes in the package `mulan.evaluation.measure` could be used to accomplish this task with a bit of programming work. The `Evaluator` class expects a trained classifier and a set of test instances as arguments, instead of predictions. The `evaluate()` method takes care of doing all the work internally, obtaining the predictions for each test sample from the classifier and updating the chosen metrics. The `Evaluator` class also has a `crossValidate()` method able to perform cross validation, using a specific number of folds and returning average performance metrics.

Both MEKA tools, the Explorer and the Experimenter, return a full list of performance metrics for each completed experiment. The values are shown in the GUI and they can be exported to a file as well. Internally, MEKA relies on the methods in the `Evaluation()` class to assess the

[4]Aside from the usual MLL performance metrics, MULAN also considers a few specific ones aimed to evaluate multiregression outputs. These are not shown in Table 5.

TABLE 5. Summary of performance metrics provided by each tool.

Metric	meka	mldr	mlr	mulan	scikitml	utiml
Instance-based						
Accuracy	✓	✓	✓	✓	✓	✓
Example AUC	✗	✓	✗	✗	✓	✗
F-measure	✓	✓	✓	✓	✓	✓
Hamming loss	✓	✓	✓	✓	✓	✓
Hierarchical loss	✗	✗	✗	✓	✗	✗
PPV	✗	✗	✓	✗	✗	✗
Precision	✓	✓	✗	✓	✓	✓
Recall	✓	✓	✗	✓	✓	✓
Subset 0/1	✓	✗	✓	✗	✓	✗
Subset accuracy	✓	✓	✗	✓	✓	✓
TNR/Specificity	✗	✗	✗	✓	✗	✗
TPR/Sensitivity	✗	✗	✓	✗	✗	✗
Label-based						
Macro AUC	✓	✓	✗	✓	✓	✓
Macro AUPRC	✓	✗	✗	✗	✗	✗
Macro F-measure	✓	✓	✗	✓	✓	✓
Macro Precision	✓	✓	✗	✓	✓	✓
Macro Recall	✓	✓	✗	✓	✓	✓
Macro Specificity	✗	✗	✗	✓	✗	✗
Micro AUC	✗	✓	✗	✓	✓	✓
Micro F-measure	✓	✓	✗	✓	✓	✓
Micro Precision	✓	✓	✗	✓	✓	✓
Micro Recall	✓	✓	✗	✓	✓	✓
Micro Specificity	✗	✗	✗	✓	✗	✗
Ranking-based						
Average precision	✓	✓	✗	✓	✓	✓
Coverage	✗	✓	✗	✓	✓	✓
Log loss	✓	✗	✗	✗	✗	✗
Margin loss	✗	✓	✗	✓	✗	✓
One Error	✓	✗	✗	✓	✗	✓
Ranking loss	✓	✓	✗	✓	✓	✓

AUC=Area Under ROC Curve, AUPRC=Area Under Precision Recall Curve, PPV=Positive Predictive Value, TNR=True Negative Rate, TPR=True Positive Rate.

performance of a classifier. The metrics are calculated through the static methods offered by the `Metrics` class in the `meka.core` package. Aside from the ones shown in Table 5, MEKA also considers a few metrics that are not so usual in MLL, such as the Jaccard index or the Levenshtein distance.

B. EVALUATING PERFORMANCE FROM R

The procedure for evaluating predictive performance in R will mainly depend on the format in which the predicted and true labels are stored. If they are in a `Prediction Multilabel` object, returned by the `predict()` function in the `mlr` package [47], the obvious way would be passing it to `mlr`'s `performance()` function. It usually needs two parameters: the object having the predictions and a list with the measures to be computed. The model that generated the predictions could be also needed for some measurements. A list of all supported multilabel metrics can be retrieved through the `listMeasures()` functions. As can be seen in the following example, only a small subset of existing measures are currently implemented:

The `predict()` function in `utiml` package [46] returns an `mlresult` object holding the predictions. It has to

```
>
> measures <- mget(listMeasures("multilabel"), envir = as.environment("package:
    mlr"))
> performance(preds, measures, model = rfmodel)
          featperc      multilabel.tpr  multilabel.hamloss multilabel.subset01
         1.0000000           0.6394875           0.4580336           0.9928058
          timeboth           timetrain         timepredict      multilabel.ppv
         0.0900000           0.0800000           0.0100000           0.3725792
     multilabel.f1      multilabel.acc
         0.4489768           0.3215382
>
```

Example 4. Evaluating predictive performance with mlr.

be fed to functions `multilabel_evaluate()` and `multilabel_confusion_matrix()`, responsible of computing performance metrics and produce a confusion matrix, respectively. A total of 19 multilabel metrics are considered. A vector with their names can be retrieved by calling the `multilabel_measures()` in this package.

Although the `mldr` [40] package does not include any MLL algorithm, it provides a evaluation function able to return a comprehensive set of performance metrics. The original goal was to facilitate a reference implementation of as many metrics as possible. This implementation has been used by other package designers. The function in charge of computing the metrics is `mldr_evaluate()`. It takes two parameters as input, the ground truth labels for every instance and the set of predictions. The returned value is a list of 20 metrics including the data needed to plot a ROC curve, as explained in Appendix C.

C. EVALUATING PERFORMANCE FROM PYTHON

The well-known Python's `scikit-learn` library has a module named `metrics`. A large collection of functions, aimed to compute binary, multiclass and multilabel performance metrics, can be found inside it. Usually, they take at least two parameters, the set of ground truth labels and the set of predicted ones. Many of them also accept an additional `average` parameter. It can take the `"samples"`, `"macro"` and `"micro"` values, so that any metric function, such as `precision_score()` or `f1_score()`, can compute both sample-based and label-based MLL metrics.

Apart from obtaining individual performance indicators, the `metrics` module also has a function able to produce a text report including the main evaluation measures. In the multilabel case this function, named `classification_report()`, displays individual precision, recall and F1-score values for each label in the MLD.

VIII. CONCLUDING REMARKS

Multilabel learning techniques have experienced an important growth in late years. However, they are still barely supported by the most popular machine learning software tools. This paper provides an up-to-date and comprehensive revision of MLL software, including data repositories, exploratory data analysis tools, and general purpose packages with MLL algorithms implementations. Firstly, a broad overview of these tools' capabilities has been portrayed, digging deep into the functionality they offer. Then, a didactic

TABLE 6. Most usual classes from the MULAN library.

Class/Package	Description
`MultiLabelInstances`	Loads a dataset from an ARFF file getting label information from an XML file.
`Statistics`	Computes several MLL data traits, including label correlation coefficients.
`mulan.transformations`	The classes in this package are in charge of performing data transformations such as BR (`BinaryRelevanceTransformation`) and LP (`LabelPowersetTransformation`).
`mulan.classifier`	This package is the root of several sub-packages, such as `mulan.classifier.lazy`, `mulan.classifier.meta` or `mulan.classifier.transformation`, that hold the classes where each MLL algorithm is implemented.
`Evaluator`	The `evaluate()` and `crossValidate()` methods in this class take care of assessing the classifiers performance.
`mulan.evaluation.measures`	All the performance metrics are defined as classes in this package.

description on how to install and use the most important ones has been contributed. This way, the present paper complements previously published tutorials focused on MLL methods [15] or the approach for performing MLL experiments [34], allowing researchers and practitioners to choose the tools that best suit their needs.

APPENDIXES
APPENDIX A
MULAN

MULAN [36] was the first general purpose library for conducting MLL experiments. It is written in Java and relies on the functionality of WEKA [3]. The MULAN library, along with its source code, documentation, the corresponding WEKA version and some sample datasets, can be downloaded from http://mulan.sourceforge.net. The installation is straightforward, since the user only has to unpack a ZIP file and set the proper path in an environment variable. A recent Java Development Kit (JDK) is assumed to be installed in the computer.

Using MULAN implies writing a Java application that imports the MULAN and WEKA packages and uses their classes. So, a certain knowledge of the Java language and basic programming techniques is required. Since Java 9 there exists a Java REPL named `jshell`. It can be used to work interactively in a similar fashion to Python or R. Once the `JAR` files containing the WEKA and MULAN libraries have been loaded, the classes summarized in Table 6 can be used to accomplish the most usual MLL tasks.

Once the program is written and compiled, the following command would run it from the command line: `java -cp mulan.jar;weka.jar myprogram`. Alternatively, `jshell` can be launched as shown in Fig. 11, with the `-c classpath` option, to work interactively. Assuming this latter configuration, the following is a sample work session with MULAN. It loads the full `emotions` dataset and obtains some data traits. Then, training and testing partitions are loaded and two different classifiers are trained and evaluated.

```
jshell> import mulan.classifier.*;
jshell> import mulan.data.*;
jshell> import mulan.classifier.transformation.ClassifierChain;
jshell> import mulan.classifier.lazy.MLkNN;
jshell> import mulan.evaluation.*;
jshell> import mulan.evaluation.measure.*;
jshell> import weka.classifiers.trees.J48;
jshell> import weka.core.Utils;
jshell> var emotions = new MultiLabelInstances("emotions.arff", "emotions.xml");
jshell> var statistics = new Statistics();
jshell> statistics.calculateStats(emotions);
jshell> statistics
Examples: 593
Predictors: 72
--Nominal: 0
--Numeric: 72
Labels: 6

Cardinality: 1.8684654300168635
Density: 0.3114109050028106
Distinct Labelsets: 27

Percentage of examples with label 1: 0.2917369308600337
Percentage of examples with label 2: 0.2799325463743676
...
// Load the train and test partitions from ARFF files
jshell> var train = new MultiLabelInstances("emotions-train.arff", "emotions.xml
        ");
jshell> var test = new MultiLabelInstances("emotions-test.arff", "emotions.xml")
        ;

// Some data exploration
jshell> System.out.println("Instances: " + train.getNumInstances() + ", Labels:
        " + train.getNumLabels() +
", Card: " + train.getCardinality())
Instances: 391, Labels: 6, Card: 1.813299232736573

// A transformation-based classifier and an adapted classifier
jshell> var class1 = new ClassifierChain(new J48());
jshell> var class2 = new MLkNN(5, 1.0);

// Train both classifiers
jshell> class1.build(train);
jshell> class2.build(train)

// Define the evaluation metrics to obtain
jshell> var evaluator = new Evaluator();
jshell> List<Measure> metrics = Arrays.asList(new HammingLoss(), new
        ExampleBasedAccuracy(), new ExampleBasedFMeasure());

// Evaluate the ClassifierChain classifier
jshell> evaluator.evaluate(class1, test, metrics)
Hamming Loss: 0,2896
Example-Based Accuracy: 0,4277
Example-Based F Measure: 0,5145

// Evaluate the MLkNN classifier
jshell> evaluator.evaluate(class2, test, metrics)
Hamming Loss: 0,2120
Example-Based Accuracy: 0,5165
Example-Based F Measure: 0,5959
```

Example 5. Sample MULAN work session.

APPENDIX B
MEKA

MEKA [37] differs from most other tools in two ways:[5] on the one hand, its objective is to facilitate the execution of

[5]Aside from multilabel learning, MEKA also provides tools to accomplish multitarget learning and hierarchical learning. Those are not considered here.

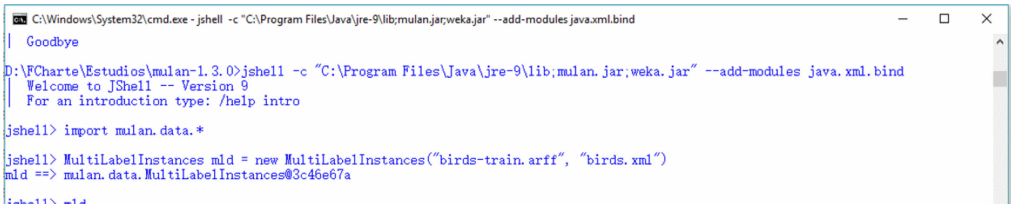


FIGURE 11. Interactive MULAN session using the Java 9 REPL.

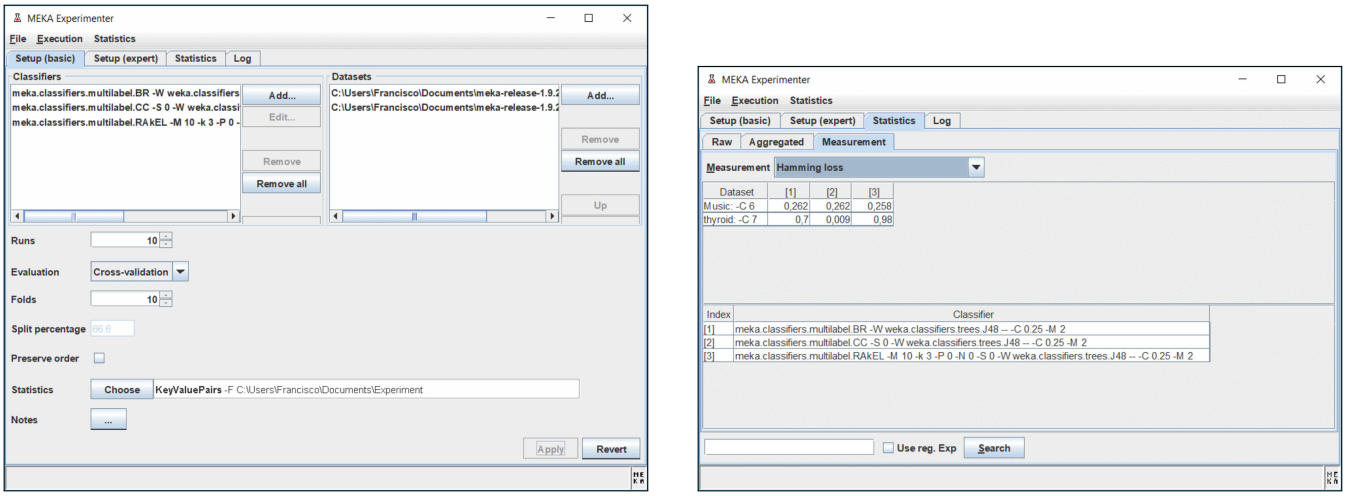


FIGURE 12. Configuring an MLL experiment in the MEKA experimenter and analyzing the results.

```
$ CLASSPATH=./lib/*
$ java meka.classifiers.multilabel.BR -t data/Music.arff
== Evaluation Info
Classifier                  meka.classifiers.multilabel.BR
Options                     [-W, weka.classifiers.trees.J48,--,-C,0.25,-M,2]
Additional Info
Dataset                     Music
Number of labels (L)        6
Type                        ML
Threshold                   0.9333333333333333
Verbosity                   1

== Predictive Performance
Number of test instances (N)
Accuracy                    0.443
Jaccard index               0.443
Hamming score               0.741
Exact match                 0.173

== Additional Measurements
Number of training instances  355
Number of test instances      237
Label cardinality (train set)  1.789
Label cardinality (test set)  1.992
Build Time                  3.05
```

Example 6. Sample MEKA work session.

MLL algorithms and obtain results rather than to perform EDA; on the other hand, it offers a GUI instead of a command line or API as MULAN, `mlr` and `scikit-multilearn` do. The MEKA GUI is inspired by WEKA's [3], so the procedure to follow to configure and run MLL experiments is quite similar to that used in standard learning tasks with the latter tool. MEKA can be downloaded from http://waikato.github.io/meka/ in a ZIP file. Once it has been unpacked, the user can simply double click the `run.sh` (GNU/Linux and MacOS) or `run.bat` (Windows) file to launch the application.

Running MLL experiments by means of the MEKA GUI is a straightforward process. The MEKA Experimenter is the best choice for comparing several algorithms over a set of MLDs. Firstly, the experiment has to be configured choosing the MLL methods and their parameters, as well as the list of datasets as shown in Fig. 12 (left). The number of runs and partitioning/evaluation scheme are also set in this page, as well as the destination file where the statistics will be stored. Once the Apply button sets the configuration, the experiment is run with the Execution>Start option. After finishing, the Statistics page allows the user to choose any metric, including running time, and compare the performance amongst classifiers (see right image in Fig. 12).

By means of the MEKA Explorer, the user can test different classifier configurations, see all the performance metrics, plot some of their precision-recall curves and ROC curves, save them for further use, etc. Fig. 13 shows several runs of different MLL methods (left) and the ROC curve corresponding to the last one (right).

Like WEKA, MEKA also allows the user to run these tasks from the command line. Assuming that the `CLASSPATH` environment variable is correctly set, any MLL algorithm can be called with the command `java meka.classifiers.multilabel.METHOD`, indicating the MLD to be processed with the `-t` option, as shown below. Additional parameters can be passed to set the underlying binary/multiclass classifier and other options.

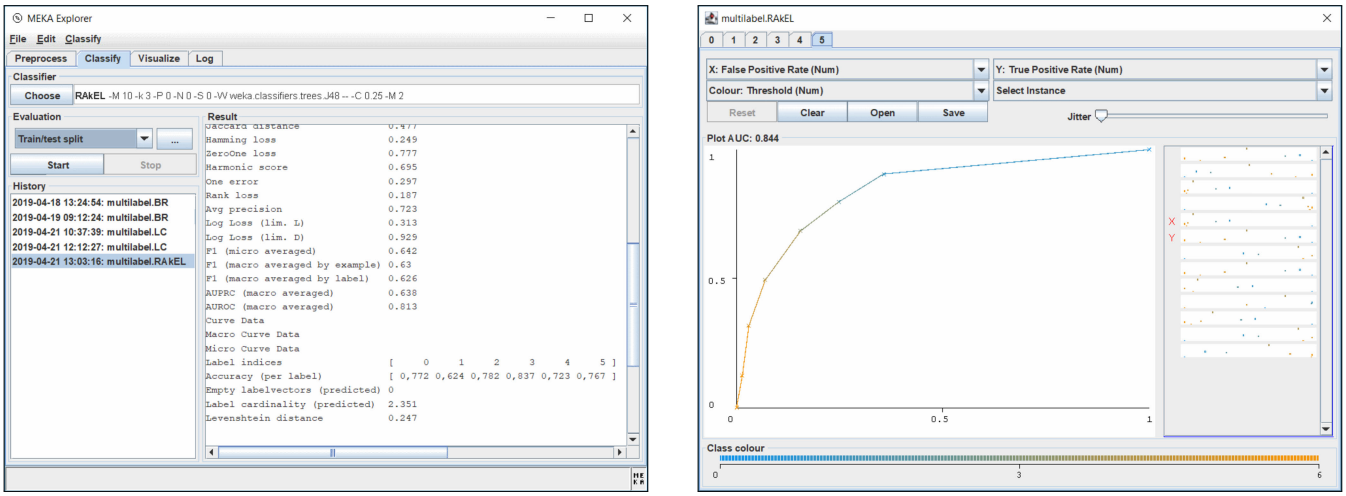


FIGURE 13. The MEKA explorer allows testing disparate configurations and visualizing the results.

FIGURE 14. Loading the mldr package into the R session.

APPENDIX C
MLDR

Aside from Java and Python, R is among the most used languages for data science. As a consequence, several R packages related to multilabel learning are available. The `mldr` package, thoroughly introduced in [40], was the first of them to be published in the CRAN[6] (Comprehensive R Archive Network), the official network for R packages. Therefore, it can be installed simply by entering the usual `install.packages("mldr")` command into the R console. The latest version of this software can be also obtained from github.com/fcharte/mldr.

Although it provides functions to perform some other tasks, the main goal of this R package is to ease the exploratory data analysis (EDA) of multilabel datasets. This work can be done either from the command line or through a graphical user interface (GUI). The command to open the GUI is shown in the console as soon as the package is loaded onto memory, as can be seen in Fig. 14.

One of the key aspects of this package is that it incorporates into R a new data structure, the "`mldr`" S3 class. This is a special kind of list holding all the information about any loaded MLD. Before the introduction of the `mldr` package there was no structure in R for working with multilabel data. Interestingly, other R packages which have included MLL capabilities lately, such as `mlr` [49], `mldr.datasets` [34] or `utiml` [46], support this same data structure.

Once a dataset has been loaded or generated, by calling `mldr()` or `mldr_from_dataframe()`, the obtained "`mldr`" object carries both the data and a summary of its traits. This includes basic characterization metrics, label and labelsets distribution, imbalance and label concurrence data, etc. Since the package overloads basic R functions such as `print()`, `summary()` and `plot()`,[7] the "`mldr`" object contents can be retrieved and plotted through these standard methods. A summary of the functions exported by the package is provided in Table 7.

As can be observed, aside from EDA functions this package also includes implementations of some transformations methods, such as BR, LP and REMEDIAL, as well as the tools needed to evaluate a set of predictions. With the exception of these, all the functionality of the package is also accessible through the GUI shown in Fig. 15.

Assuming the package is already installed, the following script demonstrates how to use some of its capabilities.

The script starts by loading a dataset, then obtaining a summary of its traits. Next, a list of labels and their basic characteristics is retrieved. A report of concurrence among labels is printed after that. Lastly, and assuming that a set of predictions have been obtained through some procedure,

[6]https://cran.r-project.org/

[7]Seven types of plots are considered, including multilabel specific designs of histograms, bar plots and circular plots.

TABLE 7. Most usual commands from the mldr package.

Command	Description
`mldr`	Loads a dataset from an ARFF file and optionally an XML file containing label information. This way MULAN, MEKA and other ARFF-based datasets can be loaded as `"mldr"` objects.
`mldr_from_dataframe`	Creates an `"mldr"` object from the data contained in an R `data.frame`, easing the generation of synthetic datasets.
`mldrGUI`	Launches a web GUI aimed to simplify the process of loading and exploring any multilabel dataset. The GUI relies on the functions that the user can also call from the command line.
`concurrenceReport`	Produces a textual report on label concurrence, including metrics, a list of label interactions and also a circular plot showings these interactions.
`mldr_transform`	Transforms a multilabel dataset into a set of binary classifiers (BR) or a multiclass dataset (LP).
`write_arff`	Exports an `"mldr"` object to an ARFF-format file, also creating the XML file containing label information.
`==`, `+` and `[]`	Operators to compare two `"mldr"` objects, concatenate two datasets and filter the samples in a multilabel dataset, respectively.
`mldr_evaluate`	Taking as input an `"mldr"` object and a set of predictions, this function computes and returns a set of 20 evaluation metrics. These can be also individually obtained through their respective functions.
`remedial`	Applies the REMEDIAL algorithm described in [48] to the dataset, thus decoupling highly imbalanced labels that concur in the same instances.

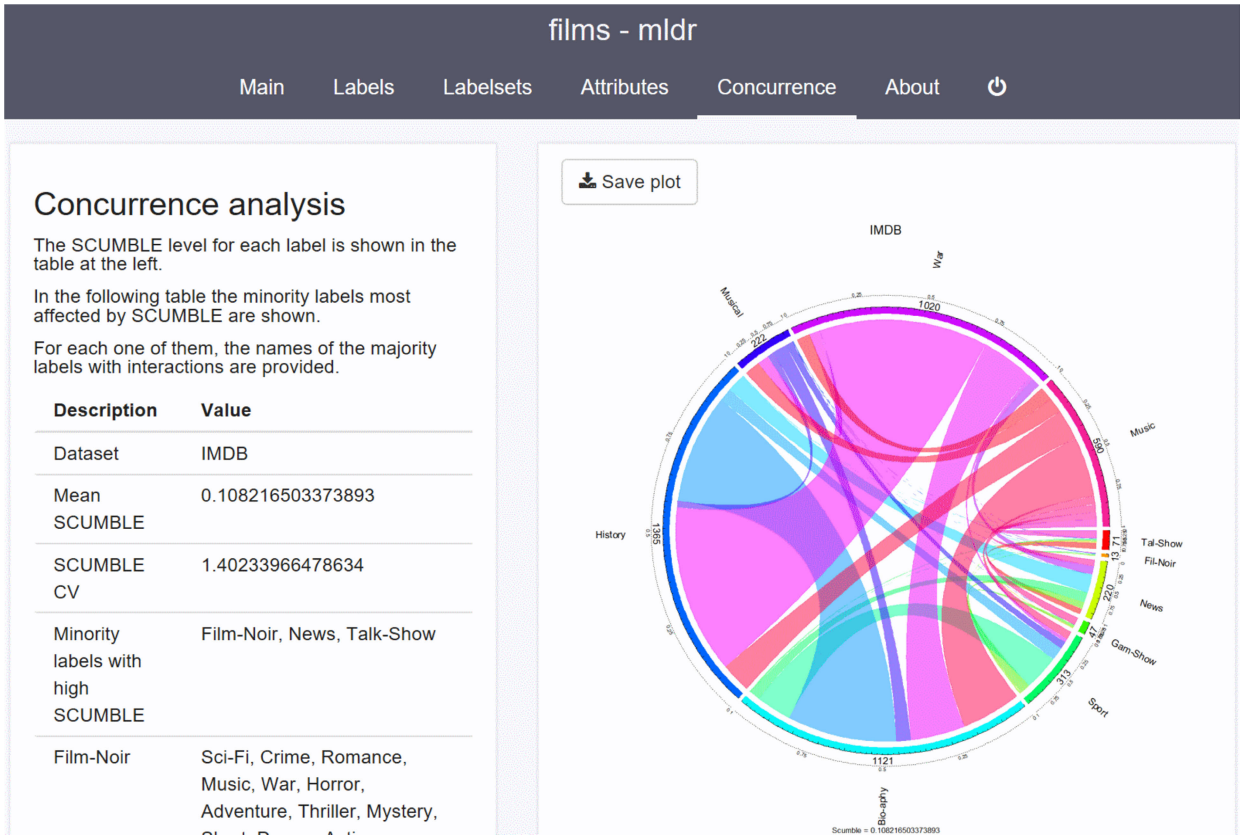


FIGURE 15. The GUI of the mldr package has several pages, providing summaries and plots for different aspects of the loaded dataset.

a evaluation is performed and the set of performance measurements is shown in the console. The last member of this list, named `roc`, can be provided to the `plot()` function to graph the usual ROC curve.

APPENDIX D
MLDR.DATASETS

Like the previous one, this is an R package available on CRAN. Therefore, it can be installed by issuing the command

TABLE 8. Most usual commands from the mldr.datasets package.

Command	Description
`available.mldrs`	Returns an R `data.frame` with name, URL and other basic data about the available datasets in the Cometa repository.
`get.mldr`	Takes the name of a dataset as parameter and loads it, if it is locally available, or downloads it from Cometa. Alternatively, the package provides a function for each dataset, e.g. `emotions()`, `slashdot()`, `etc.`, that loads the requested dataset like `get.mldr("name")` would do.
`density/sparsity`	These functions compute and return the level of density/sparsity of the dataset. The larger is the amount of attributes with zero value among the dataset instances the higher its sparsity and lower its density.
`XXX.holdout`	Where XXX can be `iterative.stratification`, `stratified` or `random`, depending on the desired partitioning strategy. This function divides the dataset into one training and one test partitions.
`XXX.kfolds`	Divides the dataset into the number of desired folds, each one made of a pair of training/test partitions. The same three partitioning strategies of `XXX.holdout` are available.
`toBibtex`	Returns a string with the citation data of the dataset formatted as a BibTeX entry.
`write.mldr`	Exports either a full or partitioned dataset to several file formats, creating the proper files.

```
> library("mldr")  # Loads the package into the current R session
>
> # Load the "emotions.arff" file with labels in "emotions.xml"
> emotions <- mldr("emotions")
> summary(emotions)
num.attributes num.instances num.inputs num.labels num.labelsets num.single.
       labelsets
1              78          593         72          6            27
                          4
max.frequency cardinality   density   meanIR    scumble scumble.cv      tcs
1           81    1.868465 0.3114109 1.478068 0.01095238    1.26456 9.364262
> 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
>
> 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:
# ...
>
> res <- mldr_evaluate(emotions, predictions)
> str(res)
List of 20
$ accuracy         : num 0.974
$ example_auc      : num 0.97
$ average_precision: num 0.965
...
$ roc              :List of 15
..$ percent          : logi FALSE
..$ sensitivities    : num [1:3] 1 0.957 0
..$ specificities    : num [1:3] 0 0.981 1
....
```

Example 7. Sample mldr work session.

```
> library("mldr.datasets")  # Load the package into the current R session
>
> # Meta-data from three of the available datasets
> available.mldrs()[c(1,16,20),]
Name                                  Description Instances Attributes Labels
                                                                    URL
1    bibtex                    Dataset with BibTeX entries      7395       1836
          159   https://cometa.ml/public/full/bibtex.rds
16 eurlexdc               EUR-Lex directory codes dataset     19348       5000
          412 https://cometa.ml/public/full/eurlexdc.rds
20     imdb Dataset generated from the IMDB film database    120919       1001
           28     https://cometa.ml/public/full/imdb.rds
>
> # Load the bibtex dataset, downloading it if it is not locally available
> databib <- bibtex()
Looking for dataset bibtex in the download directory
Looking for dataset bibtex online...
Downloading dataset bibtex
trying URL 'https://cometa.ml/public/full/bibtex.rds'
Content type 'text/plain' length 959692 bytes (937 KB)
downloaded 937 KB
>
> summary(databib) # Get a summary of traits of the data
num.attributes num.instances num.inputs num.labels num.labelsets
1           1995          7395       1836        159          2856
num.single.labelsets max.frequency cardinality    density   meanIR    scumble
1                 2199           471    2.401893 0.01510625 12.49826 0.09378705
scumble.cv      tcs
1   1.365618 20.54143
>
# Retrieves citation information of a previously loaded dataset: emotions
> cat(toBibtex(emotions))
@incollection{,
title = "multiLabel Classification of Emotions in Music",
author = "Wieczorkowska, A. and Synak, P. and Ra'{s}, Z.",
booktitle = "Intelligent Information Processing and Web Mining",
...
}
>
> # Divide the emotions dataset into 10 folds and write them using MULAN and CSV
> # file formats, taking into account the sparsity level
> write.mldr(random.kfolds(emotions, k=10), format=c("MULAN", "CSV"),
+ sparse=sparsity(emotions) > 0.5, basename="emotions")
Wrote file emotions-1x10-tra.arff
Wrote file emotions-1x10-tra.xml
Wrote file emotions-1x10-tra.csv
Wrote file emotions-1x10-tra_labels.csv
Wrote file emotions-1x10-test.arff
Wrote file emotions-1x10-test.xml
Wrote file emotions-1x10-test.csv
Wrote file emotions-1x10-test_labels.csv
Wrote file emotions-2x10-tra.arff
...
```

Example 8. Sample mldr.datasets work session.

`install.packages("mldr.datasets")` in the R console. The source code of the latest version of this software, comprehensively described in [34], is always available in the GitHub repository at github.com/fcharte/mldr.datasets.

Based on the data structure defined in `mldr`, the `mldr.datasets` package provides the functionality needed to automatically download datasets from the Cometa repository, partitioning them according to different strategies, and export them to several other file formats. A full set of informative methods, able to extract data traits and citation data, are provided as well.

Once installed, the `library("mldr.datasets")` command will load it into the current R session. Afterward, the most common steps would be the ones summarized in Table 8. The following example shows how to use some of them along the output they produced.

Once the package has been loaded, the meta-data from three available datasets is retrieved and printed. By running the `available.mldrs()` function, without the `[]` operator, a full list would be obtained. Then, one of these datasets is

automatically downloaded from Cometa. If the dataset were available in the current working directory, it would be loaded into memory without the previous step. After obtaining some traits of the data, such as the number of labels, features, label cardinality, etc., the BibTeX entry to cite a dataset is printed into the console. Lastly, the dataset is partitioned and exported to two different file formats.[8]

APPENDIX E
MLR

The `mlr` [47] package is among the best known by R users when it comes to conducting machine learning experiments. It was later extended to also face MLL tasks. These new abilities are detailed in [49]. As most R packages, `mlr` is available in the CRAN, so it can be installed as usual by issuing the corresponding `install.packages()` command. Source code and the latest version of this package is available at github.com/mlr-org/mlr.

Although `mldr` provided methods to transform MLDs, so that existing binary and multiclass learners could be used with them, `mlr` was arguably the first package to fully incorporate MLL methods into R. This package provides a common working procedure, no matter the kind of duty to perform, binary/multiclass/multilabel classification, regression, clustering, etc. Firstly a task from the original data has to be created. Second, a learner is configured from the task. Then, the learner is trained and used to obtain predictions. Lastly, performance indicators are computed from these predictions.

Table 9 summarizes the `mlr` commands that one user will need to carry on an MLL experiment. This package does not provide any method to load MLDs stored in MEKA, MULAN or other file formats. Data have to be already loaded into an R `data.frame`. Labels are expected to be stored as logical vectors. Attributes have to be of `numerical` or `factor` data types. The `character` data type (string of characters) is not supported.

Assuming the MLD is loaded into a `data.frame`, what the `mlr` package offers is essentially a set of multilabel classifiers. Specifically, it provides two adaptation-based methods and five transformation-based ones. The former group is made up of the multivariate random forest [50] (`multilabel.randomForestSRC`) and random ferns [51] (`multilabel.cforest`) methods. The latter one provides the basic binary relevant (BR) approach plus four additional binary transformations, classifier chains (CC), dependent binary relevance (DBR), BR stacking and nested BR stacking.

Regarding the evaluation of predictive performance, `mlr` implements the computation of a small subset of common MLL metrics. These include Accuracy, Hamming loss, F-measure and Subset 0/1. The full list can be obtained by issuing at the R command line the `listMeasures("multilabel")` statement.

[8]The ARFF file format considers two sub-formats called *dense* and *sparse*. The former enumerates all the attribute values in each sample, whereas the latter provides the index and value of non-zero attributes. Based on the level returned by the `density/sparsity` functions the user can decide which one of these formats to use.

```
> library("mlr")
>
> # We will create an mlr task from an mldr.datasets MLD,
> # applying the needed changes
>
> mld <- mldr.datasets::flags
> mldmlr <- data.frame( # Builds a data.frame changing the type of labels to logical
mld$dataset[,mld$attributesIndexes],
sapply(mld$dataset[,mld$labels$index], as.logical))
>
> mldmlr[sapply(mldmlr, is.character)] <-  # Change "character" columns to "factor"
lapply(mldmlr[sapply(mldmlr, is.character)], as.factor)
> mldtask <- makeMultilabelTask(id = "MLLTest",
data = mldmlr, target = row.names(mld$labels))
>
> mldtask # Explore the mlr task object
Supervised task: MLLTest
Type: multilabel
Target: red,green,blue,yellow,white,black,orange
Observations: 194
Features:
numerics     factors     ordered functionals
10           9           0           0
Missings: FALSE
Has weights: FALSE
Has blocking: FALSE
Has coordinates: FALSE
Classes: 7
red  green   blue yellow  white  black orange
153     91     99     91    146     52     26

> # Test one adaptation-based algorithm and one transformation-based
> class1 <- makeLearner("multilabel.randomForestSRC")
> class2 <- makeMultilabelBinaryRelevanceWrapper(
    makeLearner("classif.rpart", predict.type = "prob"))
> model1 <- train(class1, mldtask, subset = 1:150)
> model2 <- train(class2, mldtask, subset = 1:150)
> predict1 <- predict(model1, task = mldtask, subset = 151:194)
> predict2 <- predict(model2, task = mldtask, subset = 151:194)
> measures <- list(multilabel.hamloss, multilabel.f1, multilabel.acc, multilabel.subset01)
> performance(predict1, measures = measures)
multilabel.hamloss       multilabel.f1      multilabel.acc multilabel.subset01
0.2305195           0.7493930           0.6505952           0.7500000
> performance(predict2, measures = measures)
multilabel.hamloss       multilabel.f1      multilabel.acc multilabel.subset01
0.3051948           0.6511424           0.5324134           0.9090909
>
```

Example 9. Sample mlr work session.

The following example demonstrates how to use some of `mlr`'s capabilities. Two MLL classifiers are used to process an MLD retrieved from the `mldr.datasets` package. The first lines show how to apply the necessary changes to be used for an `mlr` task.

APPENDIX F
UTIML

The `utiml` package [46] is the most recent addition to MLL capabilities in R. Its goal is to facilitate implementations of several multilabel classifiers. Unlike `mlr`, `utiml` has been designed for working with this kind of data from the beginning. It is available at CRAN, so it can be installed through the `install.packages()` function. Their authors maintain the latest version of `utiml` in a code repository at github.com/rivolli/utiml.

Just like the `mlr` package, `utiml` also lacks the functions needed to read MLDs regardless of their format. It relies on the methods provided by `mldr` to do so. In fact, loading `utiml` into R also loads `mldr`. Once the data is available in an `mldr` object, the user can conduct preprocessing, partitioning, learning and evaluation tasks. These are the four groups of functions provided by `utiml`. The most relevant ones are summarized in Table 10.

`utiml` provides the user with a comprehensive set of transformation-based MLC methods. Some of them are not

TABLE 9. Most usual commands from the mlr package.

Command	Description
`makeMultilabelTask`	Builds a `MultilabelTask` object from the multilabel data stored in a `data.frame`. Besides the data itself, this method takes as input a `target` argument stating the names of the labels.
`makeLearner`	Instantiates any of the learning methods supported by `mlr`, either algorithms adapted to work with multilabel data or standard ones to be used as binary classifiers. Takes the name of the algorithm and the type of prediction, which usually is `"prob"` for MLL tasks, as inputs.
`makeMultilabelXXXWrapper`	Taking a base binary classifier as input, this set of methods build a multilabel learner by applying some kind of transformation. `XXX` can be `BinaryRelevance`, `ClassifierChains`, `DBR`, `Stacking` or `NestedStacking`, depending on the transformation method to follow.
`train`	Trains the selected learner using the data associated to a task. All the instances in the data are used by default, but a subset can be specified as additional parameter. This method returns the trained model as result.
`predict`	Uses the model returned by `train()` to predict the relevant labels for a set of instances. These are taken from a task, usually selecting a subset of them.
`performance`	Evaluates the predictive performance of a model by computing one or more metrics. A list with the available ones can be retrieved through the `listMeasures()` function.

TABLE 10. Most usual commands from the utiml package.

Command	Description
`fill_sparse_mldata`	Replaces NA values to 0 or "" depending on the attribute type.
`remove_XXX`	Where XXX can be `attributes`, `unique_attributes`, `labels`, `unlabeled_instances` or `skewness_labels`. These functions delete the chosen instances, attributes or labels from the dataset.
`replace_nominal_attributes`	Transforms a multivalued attribute to a set of binary ones.
`create_XXX_partition`	Where XXX can be `holdout` or `kfold`. Partitions the MLD into two parts or a set of k folds for cross-validation.
`br`, `brplus`, `cc`, `ctrl`, `dbr`, `ebr`, `ecc`, `lift`, `mbr`, `ns`, `prudent`, `rdbr`	Produce classification models based on several variations of the BR transformation approach.
`clr`, `rpc`	Produce classification models based on label-pairwise transformation.
`eps`, `lp`, `ppt`, `ps`, `rakel`	Produce classification models based on the LP transformation approach.
`ML-kNN`	Implements the ML-kNN method
`predict`	Uses the model returned by any of the previous functions to predict relevant labels for a set of instances given as parameter.
`XXX_threshold`	Where XXX can be `fixed`, `lcard`, `mcut`, `pcut`, `rcut` or `scut`. Applies a specific threshold to the predictions in order to generate the corresponding bipartition.
`multilabel_evaluate`	Evaluates the predictive performance of a model, returning the chosen metrics.

available in other software packages. By contrast, only one adaptation-based algorithm is included, ML-kNN. To train any of these models, the corresponding function has to be called, providing the training data partition as argument. The underlying classifier to be used with each binary set resulting from the transformation can also be specified. A set of eight algorithms, including C5.0, KNN, SVM and Random Forest, can be used. They need the installation of additional R packages, since they are not included in `utiml`.

Once the model has been trained, the usual `predict()` function has to be called in order to obtain label predictions for new sets of instances. These would be evaluated through the `multilabel_evalute()` method. It usually takes three parameters, the set of ground-truth labels, the predicted ones, and a vector with the performance metrics to be computed. The following code snippet shows how to perform the outlined steps.

A remarkable characteristic of `utiml` is its capability to run certain tasks in parallel. It relies on the `parallel` R package to do so.

APPENDIX G
SCIKIT-MULTILEARN

The support for MLL in Python comes from the `scikit-multilearn` library. It follows the path established by `scikit-learn`, the set of machine learning tools for data mining with Python, as it provides a comprehensive collection of classifiers.

TABLE 11. Most usual commands from the scikit-multilearn library.

Command	Description
`load_dataset`	Loads an MLD downloading it from the library's own repository if needed.
`load_from_arff`	Loads an MLD from a standard ARFF file.
`BinaryRelevance`, `ClassifierChain`, `LabelPowerset`, `RakelD`	Implementation of classifiers using the BR, CC, LP or RAkEL transformation-based algorithms.
`BRkNNaClassifier`, `BRkNNbClassifier`, `ML-kNN`, `MLARAM`, `MLTSVM`	Implementation for different adaptation-based MLL classifiers.
`fit`	This method, provided by the classifiers produced by previous functions, trains the classifier using the data given as parameter.
`predict`	Each classifier has a `predict` method to return label predictions.
`sklearn.metrics.*`	The functions in the standard `sklearn.metrics` module are in charge of computing evaluation metrics.

```
> library(utiml)
## Loading required package: mldr
...
## Loading required package: gplots
> # Get the emotions dataset from the mldr package and partition it
> partitions <- create_holdout_partition(mldr::emotions, c(train = 0.7, test =
      0.3))
> # install.packages("e1071")  # Run this to install the needed package to run
      SVM as base classifier for ECC
> # Test one adaptation-based algorithm and one transformation-based
> class1 <- ecc(partitions$train)
> class2 <- ML-kNN(partitions$train)
>
> # Predict test data and evaluate the classifiers
> predict1 <- predict(class1, partitions$test)
> predict2 <- predict(class2, partitions$test)
>
> measures <- c("hamming-loss", "F1", "accuracy")
>
> multilabel_evaluate(partitions$test, predict1, measures)
##     accuracy           F1 hamming-loss
##    0.5936330    0.6657303    0.1872659
>
> multilabel_evaluate(partitions$test, predict2, measures)
##     accuracy           F1 hamming-loss
##    0.3960674    0.4825843    0.2593633
```

Example 10. Sample utiml work session.

```
>>> from skmultilearn.dataset import available_data_sets
>>> from skmultilearn.dataset import load_dataset
>>> from skmultilearn.problem_transform import BinaryRelevance
'''Other imports'''

'''Obtain integrated MLDs'''
>>> for x in available_data_sets().keys():
>>>     print(x)

('bibtex', 'undivided')
('bibtex', 'test')
('bibtex', 'train')
('birds', 'undivided')
('birds', 'test')
('birds', 'train')
('Corel5k', 'undivided')
...
'''Load training and testing partitions'''
>>> emotions_X_train, emotions_Y_train, _, _ = load_dataset('emotions', 'train')
>>> emotions_X_test,  emotions_Y_test, _, _  = load_dataset('emotions', 'test')

'''And use them to train two classifiers'''
>>> class1 = BinaryRelevance(classifier=SVC(gamma="auto"))
>>> class1.fit(emotions_X_train, emotions_Y_train)
>>> class2 = MLkNN()
>>> class2.fit(emotions_X_train, emotions_Y_train)

'''Predict test labels and evaluate the performance'''
>>> prediction = class1.predict(emotions_X_test)
>>> print('Hamming loss: ', metrics.hamming_loss(emotions_Y_test, prediction))
>>> print('Accuracy: ', metrics.accuracy_score(emotions_Y_test, prediction))

Hamming loss:  0.26485148514851486
Accuracy:  0.14356435643564355

>>> prediction = class2.predict(emotions_X_test)
>>> print('Hamming loss: ', metrics.hamming_loss(emotions_Y_test, prediction))
>>> print('Accuracy: ', metrics.accuracy_score(emotions_Y_test, prediction))

Hamming loss:  0.30363036303630364
Accuracy:  0.13366336633663367
```

Example 11. Sample scikit-multilearn work session.

Assuming that Python and `scikit-learn` are already installed in the system,[9] adding `scikit-multilearn` is as simple as issuing the command `pip install scikit-multilearn arff` in our operating system terminal window.

[9]Anaconda (https://www.anaconda.com) is the preferred way of installing Python along the main libraries, including the Jupyter Lab work environment.

Table 11 summarizes the `scikit-multilearn` functions and objects usually needed to load an MLD, create and train a classifier, obtain and evaluate predictions. This library does not provide as many MLL algorithms as other packages, as can be seen in Table 4. However, it has other interesting and exclusive capabilities. For instance, it offers a couple of label embeddings algorithms able to find label space manifolds, so that the original MLL problem can be tackled with regression methods. It also considers generating ensembles of classifiers from arbitrary label space divisions. Lastly, a wrapper around MEKA opens the door to all the existing functionality in this Java-based package (see the corresponding appendix).

The code shown in Example 11 is part of a Jupyter notebook which queries the `scikit-multilearn` data repository, loads an MLD, for which training and testing partitions are provided, and performs several operations over it, including training and evaluating two classifiers, ML-kNN and BR.

REFERENCES

[1] M. Kuhn, "A short introduction to the caret package," *R Found. Stat. Comput.*, vol. 1, pp. 1–10, Jan. 2015.

[2] Mathworks. (2019). *Statistics and Machine Learning Toolbox*. [Online]. Available: https://www.mathworks.com/products/statistics.html

[3] G. Holmes, A. Donkin, and I. H. Witten, "WEKA: A machine learning workbench," in *Proc. 2nd Austral. New Zealand Conf. Intell. Inf. Syst. (ANZIIS)*, 2002, pp. 357–361.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[5] D. Charte, F. Charte, S. García, and F. Herrera, "A snapshot on nonstandard supervised learning problems: Taxonomy, relationships, problem transformations and algorithm adaptations," *Prog. Artif. Intell.*, vol. 8, no. 1, pp. 1–14, Apr. 2019.

[6] F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesus, *Multilabel Classification: Problem Analysis, Metrics and Techniques*. Cham, Switzerland: Springer, 2016.

[7] Z. H. Zhou, "Multi-instance learning: A survey," Dept. Comput. Sci. Technol., Nanjing Univ., Nanjing, China, Tech. Rep. 2, 2004.

[8] S. Sun, "A survey of multi-view machine learning," *Neural Comput. Appl.*, vol. 23, nos. 7–8, pp. 2031–2038, Dec. 2013.

[9] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "QUINTA: A question tagging assistant to improve the answering ratio in electronic forums," in *Proc. Int. Conf. Comput. Tool (EUROCON)*, Sep. 2015, pp. 1–6.

[10] S. D. Robinson, "Multi-label classification of contributing causal factors in self-reported safety narratives," *Safety*, vol. 4, no. 3, p. 30, 2018.

[11] O. E. Dai, B. Demir, B. Sankur, and L. Bruzzone, "A novel system for content-based retrieval of single and multi-label high-dimensional remote sensing images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 7, pp. 2473–2490, Jul. 2018.
[12] T. Liu, L. Chen, and X. Pan, "An integrated multi-label classifier with chemical-chemical interactions for prediction of chemical toxicity effects," *Combinat. Chem. High Throughput Screening*, vol. 21, no. 6, pp. 403–410, Aug. 2018.
[13] M.-L. Zhang and Z.-H. Zhou, "A review on multi-label learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1819–1837, Aug. 2014.
[14] E. Gibaja and S. Ventura, "Multi-label learning: A review of the state of the art and ongoing research," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 4, no. 6, pp. 411–444, Nov. 2014.
[15] E. Gibaja and S. Ventura, "A tutorial on multilabel learning," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 1–38, Apr. 2015.
[16] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Addressing imbalance in multilabel classification: Measures and random resampling algorithms," *Neurocomputing*, vol. 163, pp. 3–16, Sep. 2015.
[17] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "Dealing with difficult minority labels in imbalanced mutilabel data sets," *Neurocomputing*, vols. 326–327, pp. 39–53, Jan. 2019.
[18] F. Charte, A. Rivera, M. J. del Jesus, and F. Herrera, "On the impact of dataset complexity and sampling strategy in multilabel classifiers performance," in *Proc. 11th Int. Conf. Hybrid Artif. Intell. Syst. (HAIS)*, in Lecture Notes in Computer Science, vol. 9648, Apr. 2016, pp. 500–511.
[19] P. Barot and M. Panchal, "Review on various problem transformation methods for classifying multi-label data," *Int. J. Data Mining Emerg. Technol.*, vol. 4, no. 2, pp. 45–52, 2014.
[20] S. Godbole and S. Sarawagi, "Discriminative methods for multi-labeled classification," in *Advances in Knowledge Discovery and Data Mining*, vol. 3056. Berlin, Germany: Springer, 2004, pp. 22–30.
[21] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern Recognit.*, vol. 37, no. 9, pp. 1757–1771, Sep. 2004.
[22] M.-L. Zhang and Z.-H. Zhou, "Multilabel neural networks with applications to functional genomics and text categorization," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 10, pp. 1338–1351, Oct. 2006.
[23] M.-L. Zhang, "ML-RBF: RBF neural networks for multi-label learning," *Neural Process. Lett.*, vol. 29, no. 2, pp. 61–74, Apr. 2009.
[24] P. Zhang, X. Zhou, P. Pelliccione, and H. Leung, "RBF-MLMR: A multi-label metamorphic relation prediction approach using RBF neural network," *IEEE Access*, vol. 5, pp. 21791–21805, 2017.
[25] F. Markatopoulou, V. Mezaris, and I. Patras, "Implicit and explicit concept relations in deep neural networks for multi-label video/image annotation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 29, no. 6, pp. 1631–1644, Jun. 2019.
[26] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *Proc. 5th Eur. Conf. Princ. Data Mining Knowl. Discovery (PKDD)*, vol. 2168. Berlin, Germany: Springer, 2001, pp. 42–53.
[27] M.-L. Zhang and Z.-H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," *Pattern Recognit.*, vol. 40, no. 7, pp. 2038–2048, Jul. 2007.
[28] G. Tsoumakas and I. Vlahavas, "Random k-labelsets: An ensemble method for multilabel classification," in *Proc. 18th Eur. Conf. Mach. Learn. (ECML)*, in Lecture Notes in Computer Science, vol. 4701. Berlin, Germany: Springer, 2007, pp. 406–417.
[29] J. Read, B. Pfahringer, and G. Holmes, "Multi-label classification using ensembles of pruned sets," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 995–1000.
[30] G. Tsoumakas, I. Katakis, and I. Vlahavas, "Effective and efficient multilabel classification in domains with large number of labels," in *Proc. ECML/PKDD Workshop Mining Multidimensional Data (MMD)*, 2008, pp. 30–44.
[31] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Mach. Learn.*, vol. 85, no. 3, pp. 333–359, Dec. 2011.
[32] M. A. Tahir, J. Kittler, and A. Bouridane, "Multilabel classification using heterogeneous ensemble of multi-label classifiers," *Pattern Recognit. Lett.*, vol. 33, no. 5, pp. 513–523, Apr. 2012.
[33] X. Z. Wu and Z. H. Zhou, "A unified view of multi-label performance measures," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3780–3788.
[34] F. Charte, A. J. Rivera, D. Charte, M. J. del Jesus, and F. Herrera, "Tips, guidelines and tools for managing multi-label datasets: The mldr.datasets R package and the Cometa data repository," *Neurocomputing*, vol. 289, pp. 68–85, May 2018.
[35] D. Dheeru and E. K. Taniskidou. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml
[36] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, "MULAN: A Java library for multi-label learning," *J. Mach. Learn. Res.*, vol. 12, pp. 2411–2414, Jun. 2011.
[37] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes, "MEKA: A multi-label/multi-target extension to WEKA," *J. Mach. Learn. Res.*, vol. 17, pp. 21:1–21:5, Jan. 2016.
[38] C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel, "Decision trees for hierarchical multi-label classification," *Mach. Learn.*, vol. 73, no. 2, pp. 185–214, Nov. 2008.
[39] J. Xu, J. Liu, J. Yin, and C. Sun, "A multi-label feature extraction algorithm via maximizing feature variance and feature-label dependence simultaneously," *Knowl.-Based Syst.*, vol. 98, pp. 172–184, Apr. 2016.
[40] F. Charte and D. Charte, "Working with multilabel datasets in R: The mldr package," *R J.*, vol. 7, no. 2, pp. 149–162, 2015.
[41] P. Szymański and T. Kajdanowicz, "Scikit-multilearn: A Python library for multi-label classification," *J. Mach. Learn. Res.*, vol. 20, no. 6, pp. 1–22, 2019.
[42] J. T. Tomás, N. Spolaôr, E. A. Cherman, and M. C. Monard, "A framework to generate synthetic multi-label datasets," *Electron. Notes Theor. Comput. Sci.*, vol. 302, pp. 155–176, Feb. 2014.
[43] O. Luaces, J. Dıez, J. J. del Coz, J. Barranquero, and A. Bahamonde, "Synthetic datasets for sound experimental evaluation of multilabel classifiers," Universidad de Oviedo, Oviedo, Spain, Tech. Rep., 2012, pp. 1–16. [Online]. Available: https://www.aic.uniovi.es/mlgroup/repository/ml_generator/
[44] J. M. Moyano, E. L. Gibaja, and S. Ventura, "MLDA: A tool for analyzing multi-label datasets," *Knowl.-Based Syst.*, vol. 121, pp. 1–3, Apr. 2017.
[45] K. Sechidis, G. Tsoumakas, and I. Vlahavas, "On the stratification of multi-label data," in *Machine Learning and Knowledge Discovery in Databases*. Berlin, Germany: Springer, 2011, pp. 145–158.
[46] A. Rivolli and A. C. P. L. F. de Carvalho, "The utiml package: Multi-label classification in R," *R J.*, vol. 10, no. 2, pp. 24–37, 2018.
[47] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones, "MLR: Machine learning in R," *J. Mach. Learn. Res.*, vol. 17, p. 170:1–170:5, 2016.
[48] F. Charte, A. J. Rivera, M. J. del Jesus, and F. Herrera, "REMEDIAL-HwR: Tackling multilabel imbalance through label decoupling and data resampling hybridization," *Neurocomputing*, vols. 326–327, pp. 110–122, Jan. 2019.
[49] P. Probst, Q. Au, G. Casalicchio, C. Stachl, and B. Bischl, "Multilabel classification with R package MLR," *R J.*, vol. 9, no. 1, pp. 352–369, 2017.
[50] M. Segal and Y. Xiao, "Multivariate random forests," *WIREs Data Mining Knowl. Discovery*, vol. 1, no. 1, pp. 80–87, Jan. 2011.
[51] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua, "Fast keypoint recognition using random ferns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 3, pp. 448–461, Mar. 2010.

FRANCISCO CHARTE (Member, IEEE) received the T.Eng. and B.Eng. degrees (Hons.) in computer science from the Universidad de Jaén, in 2008 and 2010, respectively, and the M.Sc. degree in soft computing and computational intelligence and the Ph.D. degree from the Universidad de Granada, in 2011 and 2015, respectively.

He is currently an Assistant Professor with the Computer Science Department, Universidad de Jaén, Spain. He is the author of more than 120 books, including the book *Multilabel Classification. Problem Analysis, Metrics and Techniques* (Springer), and authored more than 20 JCR research articles and 25 contributions to international conferences. His main research interests include multilabel learning, imbalanced and high-dimensionality problems, and representation learning through deep learning techniques. He was a recipient of the Extraordinary Award for the T.Eng. and B.Eng. degrees and the 1st National Award for Excellence in Academic Performance from the MECD, in 2010.