mldr.resampling: Efficient Reference Implementations of Multilabel Resampling Algorithms

Antonio J. Rivera[a], Miguel A. Dávila[a], D. Elizondo[b], María J. del Jesus[a], Francisco Charte[a,*]

[a] *Department of Computer Science, Universidad de Jaén, 23071 Jaén, Spain*
[b] *Department of Computer Science and Informatics, De Montfort University (UK)*

Abstract

Resampling algorithms are a useful approach to deal with imbalanced learning in multilabel scenarios. These methods have to deal with singularities in the multilabel data, such as the occurrence of frequent and infrequent labels in the same instance. Implementations of these methods are sometimes limited to the pseudocode provided by their authors in a paper. This Original Software Publication presents `mldr.resampling`, a software package that provides reference implementations for eleven multilabel resampling methods, with an emphasis on efficiency since these algorithms are usually time-consuming.

Keywords: Multilabel learning, Imbalanced learning, Resampling algorithms, R software package

1. Introduction

MultiLabel Learning (MLL) [1] is one of the most common machine learning tasks today. It is based on the idea that each data sample is associated with a certain subset of labels. The full set of labels can be large, in many cases even having more labels than input features. As a result, it is common for some labels to occur in only a few samples, while others occur much more frequently. The label imbalance [2] in MLL is almost always present, and it is a serious obstacle to training good classifiers.

Class imbalance is a very well-known problem in traditional learning tasks such as binary and multiclass classification. Hundreds of articles [3, 4, 5], conference papers [6] and books [7] have been devoted to studying it and proposing possible solutions. The most popular are data resampling, cost-sensitive learning and mixtures of these approaches [8, 9]. However, imbalanced learning in the MLL field presents some specific aspects that make it more difficult to deal with this problem.

Resampling algorithms, which can generate new samples associated with underrepresented labels or remove those that are overrepresented, are a model-independent solution to imbalance. Several such algorithms adapted to multilabel datasets (MLD) have been proposed in recent years [2, 10, 11, 12, 13, 14, 15]. However, reference implementations are not yet available for all of them.

Several of these proposals rely heavily on finding nearest neighbors between each data sample and all the others in the dataset. This is a computationally expensive task. It will be desirable to take advantage of modern hardware and speed up this work through parallelization. This will make the time needed to resample some MLDs more affordable for everyone.

The `mldr.resampling` software package is a new member of the *mldr ecosystem*, which also includes the `mldr` [16] and the `mldr.datasets` [17] packages. All of them are written in R language and are available from CRAN (*Comprehensible R Archive Network*), the official repository of R software. The new package provides implementations of a dozen of resampling algorithms for MLDs.

This document is structured as follows. Section 2 sets the context from which the package originates. The software itself, its structure, functionalities, and implementation details are described in Section 3. Several illustrative usage examples are then provided in Section 4. Finally, some conclusions are drawn in Section 5.

2. Background

Labeled data is used everyday to create models through supervised algorithms. These usually assume that each data pattern has only one label. However, this is not always the case. Beyond the scenarios of binary learning —e.g. an email is considered as spam or it is not— and multiclass learning —e.g. a flower is cataloged into one of a set of species— there are others such as MLL [1]. This is considered a non-standard learning situation [18], among others such as multiview learning [19] and multiinstance learning [20].

Let $X^1, \ldots X^f$ be the domains of the f features in a dataset and L be the set of distinct labels. The *i-th* data pattern I_i in an MLD is defined as (1).

*Corresponding author.
Email addresses: `arivera@ujaen.es` (Antonio J. Rivera), `mdavila@ujaen.es` (Miguel A. Dávila), `elizondo@dmu.ac.uk` (D. Elizondo), `mjjesus@ujaen.es` (María J. del Jesus), `fcharte@ujaen.es` (Francisco Charte)

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

In MLL Y_i can be any subset of L, instead of just one label as in binary and multiclass learning. This also includes the empty set and the full set of labels. Given an MLD D with instances x_i, the goal of MLL is to find a model such as the one in (2) that is able to predict Y_i', i.e. the subset of labels applicable to each instance.

$$Y_i' = f(x_i), Y_i' \in L. \quad (2)$$

MLL has been applied is disparate fields, including the processing of narratives related to aviation safety [21], content-based retrieval of remote sensing images [22], predicting the toxicity effects of chemicals [23], and the automatic tagging of posts in forums [24], among many others. Several reviews of all these works and other MLL techniques are available [25, 26, 27].

The data used for training a classifier is often imbalanced. As a result, the class labels that are assigned to each instance are not equally represented. For binary datasets [28] and to a lesser extent for multiclass ones [29], this is a well studied problem. In order to know the level of imbalance of the datasets, a measure called *imbalance ratio*) (*IR*) [28] is used.

$$IR = \frac{\#samples\ majority\ class}{\#samples\ minority\ class} \quad (3)$$

In a binary scenario, this ratio is computed as shown in (3). The greater the proportion of instances associated with the most common class as opposed to the minority one, the higher the *IR* will be. When there are more than two classes or labels, individual *IRs* are obtained pairwise, then an average *IR* —usually noted as *MeanIR*— is calculated.

Traditionally, techniques such as data resampling, cost-sensitive learning, and algorithm-specific adaptations have been used to deal with imbalanced classification [30]. The former have the advantage of being model independent, i.e. they do not require changes to the model to improve the learning process.

2.1. Highly imbalanced labels

Imbalance in the MLL domain presents some additional difficulties. While in the binary and multiclass scenarios having $MeanIR > 10$ is generally considered a high level of imbalance, in MLL it is quite common. In fact, only a few popular MLDs[1] have a $MeanIR \leq 10$, as can be seen in Figure 1. Many of them have $MeanIR \in [30, 100]$, a few suffer from $MeanIR \in [150, 500]$ and there are some extreme cases with $MeanIR > 1000$.

[1] *IR* of most publicly available MLDs, among other common metrics, are available in the Cometa multilabel dataset repository: `https://cometa.ujaen.es/datasets`.

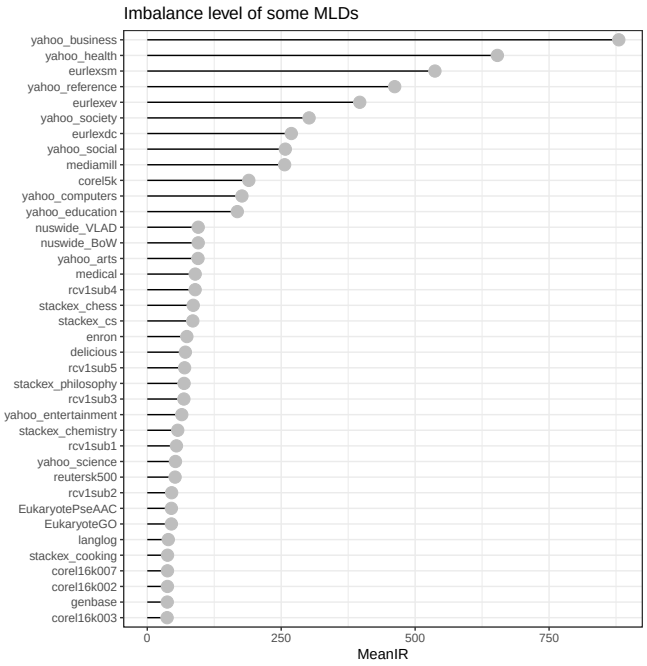

Figure 1: Mean imbalance ratio of MLDs obtained from the Cometa dataset repository. Datasets having MeanIR below 35 or above 1000 have been excluded.

This means that there are rare labels that occur once in several hundreds of the common ones. It is therefore extremely difficult to learn a model that correctly handles these low frequency labels. The problem can be exacerbated when some MLL techniques, such as binary relevance [31], are applied, since an independent model is learned for each label against all the others.

2.2. Coupling of frequent and rare labels

Most of the popular solutions for dealing with data imbalance when training models, derived from the standard scenarios (binary and multiclass classification), cannot be directly applied in the MLL case due to the unique nature of the data samples. In the first case, it is possible to collect all the samples associated with the minority class and generate some synthetic instances, so that the imbalance ratio is reduced. Similarly, some of the instances corresponding to the majority class could be removed. Since each data pattern corresponds to only one label, it is quite easy to apply data resampling methods.

In MLDs, each instance has multiple labels associated with it, so when a sample is removed or produced, all of its labels are affected, not just the most common or rarest label. In addition, it is quite normal in MLDs for the less frequent labels to appear coupled to the majority ones [32] in the same instances (see Figure 2). This casuistic is measured by means of the *SCUMBLE* metric introduced in [32]. As a consequence, an increase in the number of samples containing rare labels also implies an increase of the most common ones, and the *IR* remains unchanged.

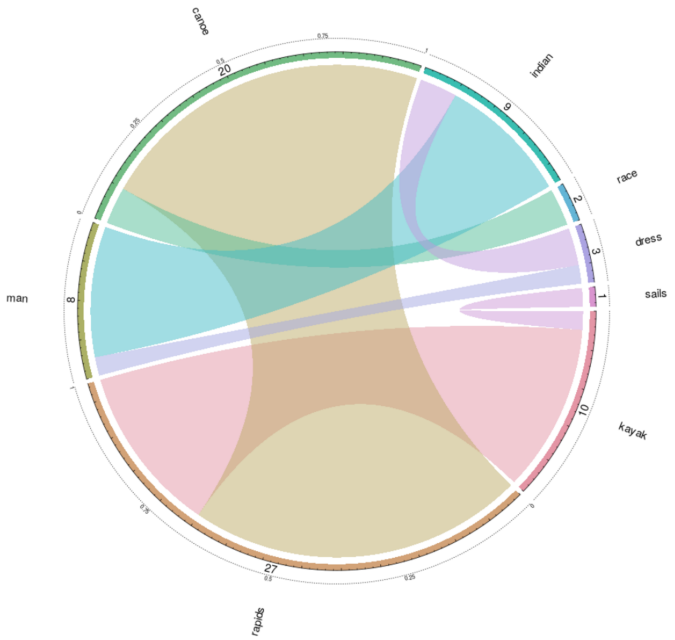

Figure 2: This concurrence diagram shows how all the instances of three minority labels, in the right side of the plot, always appear in instances having one or more majority labels.

2.3. Computational cost

Although high imbalance levels and the coupling of rare and frequent labels are the main obstacles to learning with imbalanced MLDs, there is another fact to consider: the huge computational cost imposed by some of the proposed solutions.

First of all, MLDs sometimes have hundreds if not thousands of labels. This means that there is no single majority or minority label, but sets of them. Tuning the frequency of all instances associated with these sets is more costly than dealing with just one minority class as in binary or multiclass learning.

In addition, there are proposed algorithms for resampling imbalanced MLDs that rely heavily on nearest neighbor search. Computing the distances between all the samples in a dataset is already computationally expensive. In the case of MLDs, the cost is higher because of the usually large number of features —many MLDs come from text document classification— and also the need to repeat the search for each minority or majority label to be resampled.

2.4. Resampling algorithms

The methods proposed in the literature to deal with imbalance in the MLL field are mostly based on data resampling. Many of the existing MLL methods are built as ensembles of simpler models. This is the case of [33, 34, 35, 36, 37] among many others. Introducing cost-sensitive learning or other algorithmic modifications to improve learning from imbalanced data is not an easy task. On the contrary, a data resampling algorithm can be applied in a preprocessing step, regardless of the MLL model used in the end.

Name	Description	Ref.
LPROS	Random oversampling of labelsets	[2]
MLROS	Random oversampling of one minority label	[2]
MLRkNNOS	Reverse-nearest neighborhood based oversampling	[10]
MLSMOTE	Synthetic oversampling based on neighborhood similarity	[11]
MLSOL	Oversampling based on local label imbalance	[38]
LPRUS	Random undersampling of labelsets	[2]
MLRUS	Random undersampling of one majority label	[2]
MLTL	Tomek Link undersampling	[13]
MLUL	Undersampling based on local label imbalance	[38]
MLeNN	Heuristic undersampling of majority labels	[14]
REMEDIAL	Decoupling of minority and majority labels	[32]

Table 1: Some of the most popular resampling algorithms for MLDs and reference to the papers where they were introduced.

Table 1 lists most of the MLL resampling methods proposed in the literature. There are five oversampling algorithms, five more that perform undersampling, and one that takes a different approach by decoupling instances where rare and frequent labels coexist. The rightmost column gives the reference to the paper where each algorithm was introduced. We have used the pseudo-code in these papers as a guide to implementing each of them. Most of them have been reviewed in a recent paper [39].

3. Software framework

This section describes the reasons for developing this software package, the functionality it provides, and some internals about how it works. Its dependencies, i.e. other software packages on which it depends, are also listed. Finally, it is explained how the package can be extended to include additional algorithms.

3.1. Rationale for mldr.resampling

Several MLL resampling algorithms have been introduced in recent years. Some of these proposals are just a description of the new algorithm's operation. A handful of them provide pseudocode that can be followed to implement it. Only a few also supply an open, ready-to-use implementation. In these cases, the implementation can be in any of a number of languages, including Java, R, MATLAB and Python.

Planning new experiments and comparisons with these resampling algorithms is difficult because there is no common open source implementation for all of them. In addition, the few implementations that are available to everyone are written in a variety of languages, making them difficult to run and compare. Finally, it should be noted that existing implementations rarely include optimization, so they tend to be slow in general. These are the main reasons that led us to develop the software package presented here.

We conducted a literature search to find those algorithms for which pseudocode or an existing implementation was available. Having the pseudocode is a minimum requirement, so the implementations in `mldr.resampling` followed the algorithms' authors' guidelines exactly.

When it came to choosing a programming language to implement these algorithms, we considered several aspects. First, which language had the better MLL ecosystem for working with multilabel data, as we did not want to reinvent the wheel. Second, the accessibility of the language to non-programmers, as not all data scientists are experienced programmers. Following the review in [40], it was clear that R has a larger set of software packages available for MLL. In addition, R has a gentle learning curve compared to other programming languages.

Some of the R packages related to MLL are `mldr` [16], which provides the infrastructure needed to load and analyze MLDs; `mldr.datasets` [17], with functions to import, partition and export MLDs into different file formats; `mlr` [41] offers some multilabel methods, while `utiml` [42] provides various utilities for preprocessing, sampling and analysis of multilabel data. Additional specific MLL algorithms can be found in R packages such as `NetDA` [43], MLPUGS [44] or CAMML [45], among others.

Although `mldr.resampling` is implemented in R, its functionality is also available from other languages. There are software interfaces such as `rpy2` [46], `RJava` [47] and `RInside` [48] that allow access to R software from Java, Python and C++ respectively.

3.2. Software architecture

Our `mldr.resampling` package works with multilabel datasets. The infrastructure for this is supplied by the `mldr` package [16] as an R S3 class, also called `mldr`. S3 is an object-oriented mechanism in the R language that relies on generic functions. So most of the functions exported by `mldr.resampling` take a `mldr` object as a parameter. Users would load their MLDs using the facilities offered by `mldr` and get a `mldr` object as a result. These objects are later used as inputs to the methods in `mldr.resampling`.

Each one of the resampling algorithms shown in the Table 1 is implemented in `mldr.resampling` as an independent method. This allows the users to experiment with the algorithm(s) they are interested in without any additional software layers.

As well as applying an algorithm to a MLD, the package also considers the possibility of running several methods to compare them. To achieve this goal, a function is provided that automates the procedure of calling each of the algorithms with the same MLD and the correct parameters.

3.3. Software functionalities

The main contributions of this new software package are as follows:

- Reference implementations of eleven resampling algorithms for MLD, with source available after installing the package.
- A unified interface that facilitates the application of several of these algorithms to an MLD, automating several of the steps required to compare them.
- Built-in optimizations to speed up the nearest neighbor search through task parallelization and neighbor caching.

These functionalities can be accessed in different ways depending on the user's needs. To apply one of the algorithms to an MLD, the following methods are available. For each one, the name of the method and its parameters are given, together with a brief summary of how the algorithm works.

- `LPROS(mld, pc)`: This oversampling algorithm was proposed in [2]. It applies the LP (*Label Powerset*) transformation, then randomly selects samples containing minority labelsets and clones them. The method resamples the `mld` given as the first input until a `pc`% increase in the number of samples is reached.
- `LPRUS(mld, pc)`: This undersampling algorithm was proposed in [2]. It applies the LP transformation, then randomly selects samples containing majority labelsets and removes them. The method resamples the `mld` given as the first input until a `pc`% decrease in the number of samples is reached.
- `MLeNN(mld, thr, k)`: This undersampling algorithm was proposed in [14]. It is based on the ENN (*Edited Nearest Neighbor*) rule [49], according to which if a sample has a different set of labels than its neighbors, the sample is removed. The method removes samples in the `mld` with majority labels whose differences with their `k` nearest neighbors are above the `thr` threshold.
- `MLRkNNOS(mld, k)`: This oversampling algorithm was proposed in [10]. It uses the concept of reverse nearest neighbors, in order to create new instances for each label. Several radial SVMs, one for each label, are then trained in order to predict each label of the synthetic instances. The method resamples the `mld` producing new instances based on the `k` reverse-nearest neighbors.

- `MLROS(mld, pc)`: This oversampling algorithm was proposed in [2]. It searches for samples containing minority labels and clones them by a random selection. The method resamples the `mld` by cloning random instances in which minority labels appear up to a point at which a `pc`% increase is reached.

- `MLRUS(mld, pc)`: This undersampling algorithm was proposed in [2]. It searches for samples containing majority labels and removes them by random selection. The method resamples the `mld` given as first input removing random instances in which majority labels appear up to a point at wich a `pc`% decrease is reached.

- `MLSMOTE(mld, k)`: This oversampling algorithm was proposed in [11]. It is based on the well-known SMOTE algorithm [50], which creates synthetic samples by interpolating the feature values. The labels of new samples are selected by ranking the active labels in the neighborhood. The method generates synthetic instances in which one or more minority labels appear, producing characteristics from `k` nearest neighbors.

- `MLSOL(mld, pc, k)`: This algorithm, proposed in [38], applies oversampling to difficult regions of instance space to help classifiers distinguish labels. The method resamples the `mld` creating new instances on difficult regions of the instance space, using local information of `k` neighbors.

- `MLTL(mld, thr)`: This undersampling algorithm was proposed in [13]. Its goal is to identify and remove Tomek Links [51], i.e. majority instances with a very different neighborhood. The method removes instances in the `mld` whose differences are above the `thr` threshold.

- `MLUL(mld, pc, k)`: This undersampling algorithm was proposed in [38]. It aims to identify instances that contain majority labels and remove their neighbors that are too different in terms of active labels. The method resamples the `mld` removing instances on difficult regions of the instance space, using local information of `k` neighbors.

- `REMEDIAL(mld)`: This oversampling algorithm was proposed in [15]. It decouples common and uncommon labels appearing in the same instance. It does this by aggregating new instances into the dataset and editing the labels in them. The method decouples highly imbalanced labels occurring in the same samples of the `mld` by splitting each into two instances, one with the minority labels and another with the majority labels.

It is quite common to be interested in trying different resampling approaches on the same dataset. This allows several algorithms to be compared and the best one chosen for a particular MLD. Although the aforementioned methods follow different strategies to create or remove data samples, many of them rely on retrieving information from their neighbors. Since most MLDs have hundreds or even thousands of features, as well as a similar number of instances, finding `k` nearest neighbors for every instance has a huge impact on the time needed to run them.

The `mldr.resampling` package provides an additional method —named `resample(mld, algorithms)`— capable of executing all methods specified by the `algorithms` parameter on the same `mld`. This not only automates work that would otherwise require multiple function calls, but also improves the efficiency of running these methods through a caching mechanism. Once the nearest neighbor information is obtained, it is cached and passed as input to the individual methods, so that this task does not consume any time.

Another feature built into the package, accessible to the user via the `setParallel()` function, is the ability to distribute computations across all the cores in the system, instead of using just one as usual. Limiting the number of cores used by algorithms is also possible by calling the `setNumCores()` function.

The current set of algorithms included in the package is expected to grow in the coming period as new proposals are made available with the minimum guidelines needed to implement them. The structure of the code makes this process easy.

3.4. Implementation details

All methods have been implemented in pure R, so that the software package does not need to be compiled, and the source code is available to any R practitioner interested in digging into the code. Each method has been carefully written according to the instructions given in the respective paper. Where the original source code was available, the results of the new implementation were compared with those of the original to ensure consistency.

Although having all the methods available in a language like R makes them accessible to a large group of researchers, R is certainly not the most efficient language when it comes to execution speed. For this reason, the `mldr.resampling` package has been written with efficiency in mind through the following approaches:

- The garbage collection process of most dynamic languages can have a large impact on execution times. Since most objects in R are immutable, growing certain data structures —such as vectors and matrices— means allocating new blocks of memory, copying old data, and freeing previously allocated memory. The `mldr.resampling` package takes advantage of the language's vectorization facilities whenever possible, avoiding loops to minimize the time spent on garbage collection.

- Harnessing the power of today's multi-core processors is essential to reduce execution time. However, few languages provide a standard way of doing this so that it can be used on all operating systems. The `mldr.resampling` code is designed so that some tasks can be run sequentially or in parallel, depending on whether the `parallel` R package is installed. By default, parallelization is not active, so the `parallel` package is not required. If it is available[2], the user can enable parallelism simply by calling the `setParallel()` function with the `TRUE` value as a parameter.

- Many of the resampling methods in the package are based on nearest neighbor search. In their original implementation, this is the task that consumes most of the running time. For numerical attributes, finding the k nearest neighbors for a given data sample means computing distances, almost always Euclidean, with all other instances in the MLD. If the dataset contains nominal attributes, the distances are even more expensive to obtain by VDM (*Value Difference Metric*) [52]. All methods available in the `mldr.resampling` package optionally take two additional parameters, `neighbors` and `tableVDM`, which allow reusing these calculations. This caching method is automatically used when the `resample()` function is called to run more than one algorithm on the same data.

3.5. How to extend mldr.resampling

Adding new algorithms to the `mldr.resampling` package is a straightforward process. The steps are shown below:

1. Add a new R source file named `ALGORITHM.R`, where `ALGORITHM` is the name of the algorithm to be implemented.
2. In this file, define a function called `ALGORITHM()` with the following parameters:
   - `D`: A `mldr` object with the MLD to resample.
   - `P`: The percentage to increase or decrease the size of the original MLD.
   - `k`: If the algorithm works with nearest neighbors, this parameter sets the number of them used.
   - `others`: Any other specific parameters required by the algorithm. For example, MLTL and MLeNN take an additional argument specifying the threshold to be applied.

   The function can also take two optional parameters: `neighbors` and `tableVDM`. These are only needed for algorithms that rely on computing nearest neighbors and allow the `mldr.resampling` cache system to speed up the search for distances.
3. Write the body of the `ALGORITHM()` function to implement the new method. The following guidelines should be observed:
   - The `utils.R` module provides a set of utility functions to compute euclidean and VDM functions, get the neighbors of an instance, etc.
   - Any code that can be parallelized should use the `mldr.resampling.env$.mldrApplyFun2` function instead of the usual `lapply`. This way, the users can control whether they want to use parallelism or not.
4. Document the new function in standard Roxygen format so that it can be integrated with R's help system.
5. Finally, the new function needs to be added to the body of the `executeAlgorithm()` function, located in the `utils.R` module, so that it is integrated into the package and can be accessed like any other of the existing methods.

These steps can be seen in the source code of each of the methods already implemented in the package, which is available to users so that they can use it as a guide when writing their own algorithms. In addition, the documentation included in the package details the work done by each utility function.

4. Illustrative examples

The latest stable[3] version of the `mldr.resampling` package is available on CRAN, so it can be installed like any other R package by calling `install.packages()`, as shown below. Once installed, the `library()` function will load the package and display a message to the user about how to enable parallel processing.

```
> install.packages("mldr.resampling")
> library(mldr.resampling)
Enter setParallel(TRUE) to enable parallel
    computing
```

Some MLDs are needed to run the resampling algorithms. As `mldr.resampling` has been installed, this implies that the `mldr` package has also been installed. The command `data(package="mldr")` will list the MLDs available in this package. Any other MLD can be fetched from the file system or downloaded from the Cometa repository using the functions provided by the `mldr.datasets` package.

By means of the same `data()` function the `emotions` MLD is loaded into memory and some measurements, such as the number of instances, average *IR* and *SCUMBLE*, are obtained:

[2] In latest versions of R this package is included in the base installation.

[3] The development version of the package is hosted on GitHub: `https://github.com/madr0008/mldr.resampling`.

```
> data(emotions, package="mldr")
> emotions$num.instances
[1] 593

> emotions$meanIR
[1] 1.478068

> emotions$scumble
[1] 0.01095238
```

This dataset is slightly imbalanced, with $MeanIR = 1.4781$, and the coupling of minority and majority labels is also bearable, with $SCUMBLE = 0.0109$. Nevertheless, it may benefit from the application of some resampling algorithm. A first option could be the simple LPROS oversampling method:

```
> lpemotions <- LPROS(emotions, P=25)
> lpemotions$measures[c("num.instances", "meanIR"
    , "scumble")]
$num.instances
[1] 741

$meanIR
[1] 1.355174

$scumble
[1] 0.007740464
```

The call to the `LPROS()` function returns a new `mldr` object representing the MLD after resampling. As can be seen from the measurements, this version of the dataset has a few more samples and both *MeanIR* and *SCUMBLE* have decreased, which is positive.

Since all functions corresponding to a resampling algorithm return a `mldr` object as a result, chaining several methods is quite easy. For example, LPROS creates new samples based on minority label combinations rather than individual labels. A dataset with a high degree of coupling between minority and majority labels would hardly benefit from this algorithm. However, label decoupling can be applied before LPROS by using the REMEDIAL algorithm, as shown below.

```
> lpemotions <- LPROS(REMEDIAL(emotions), P=25)
> lpemotions$measures[c("num.instances", "meanIR"
    , "scumble")]
$num.instances
[1] 999

$meanIR
[1] 1.18863

$scumble
[1] 0.00224451
```

Both the average imbalance ratio and the coupling index provided by the SCUMBLE metric are much better than those achieved by LPROS alone.

By default, the functions in `mldr.resampling` do not use parallelization. Because of this, certain algorithms — those that compute nearest neighbors — will take a considerable amount of time. A progress bar is used to tell the user how the process is going, as well as an estimate of the time remaining. Once the method is complete, as in the following example, the resulting MLD is available as with any other function.

```
> smemotions <- MLSMOTE(emotions, k=5)
|++++++++++++++++++++++++++++++++++++| 100%
    elapsed=15m 02s
> smemotions$measures[c("num.instances", "meanIR"
    , "scumble")]
$num.instances
[1] 1247

$meanIR
[1] 1.298585

$scumble
[1] 0.004986872
```

Distributing the workload across all the processing cores of the user's machine is a one-step operation, calling the `setParallel(TRUE)` function. This function checks if the required `parallel` package is installed, and notifies the user if it is not. Once parallel processing is enabled, any method execution will take advantage of this capability. In the example below, the same MLSMOTE algorithms took about a quarter of the time to run.

```
> setParallel(TRUE)
# Parallel computing enabled on all 16 available
    cores. Use function setNumCores if you wish
    to modify it
> smemotions <- MLSMOTE(emotions, k=5)  # 4m 11s
...
```

Although calling the functions associated with individual methods allows the user to test any of them, a planned experiment will usually involve running several of them. This work can be automated using the `resample()` function, as shown in the example below. In this case, four resampling algorithms are applied to the same MLD. As can be seen from the output of the function, the first step is to compute several structures that act as a nearest neighbor cache. In this way, the overall running time will be shorter than if each algorithm searched for its own set of neighbors.

```
> resample(emotions,
   c("MLROS", "MLRUS", "MLeNN", "MLSOL"),
   outputDirectory="~/datasets")
# Calculating structures for dataset musicout ,
    if necessary. Once this is done, algorithms
    will be applied faster
# Calculating VDM table for dataset musicout
# Time taken (in seconds): 0.0046391487121582
# Calculating neighbors structure for dataset
    musicout . Once this is done, algorithms will
     be applied faster
# Time taken (in seconds): 485.649948835373
# Running MLROS on musicout with P = 25
# Time taken (in seconds): 0.0200722217559814
# Running MLRUS on musicout with P = 25
# Time taken (in seconds): 0.0121262073516846
# Running MLeNN on musicout with TH = 0.5 and k =
    3
# Time taken (in seconds): 0.33689546585083
# Running MLSOL on musicout with P = 25 and k = 3
# Part 1/3: Neighbors were already calculated.
    That just saved us a lot of time!
# Part 2/3: Calculating auxiliary structures
```

```
# Part 3/3: Generating new instances
# Time taken (in seconds): 3.47672557830811
# End of execution. Generated MLDs stored under
    directory ~/datasets
# algorithm               time
# 1      MLROS 0.0200722217559814
# 2      MLRUS 0.0121262073516846
# 3      MLeNN   485.991483449936
# 4      MLSOL   489.131313562393
```

Since multiple versions of the dataset are created, one for each algorithm, the `resample()` function will store them in the file system rather than in memory. By default it will use a temporary folder, but the user can specify any path with the `outputDirectory` parameter. The name of the generated files (see Figure 3) will be a combination of the original MLD, the algorithm and the parameters to run it.

```
fcharte@Volstag:~$ ls datasets/ -l
total 1720
-rw-r--r-- 1 fcharte fcharte 477082 May 17 12:03 musicout_MLROS_P_25.arff
-rw-r--r-- 1 fcharte fcharte 329016 May 17 12:03 musicout_MLRUS_P_25.arff
-rw-r--r-- 1 fcharte fcharte 568200 May 17 12:03 musicout_MLSOL_P_25_k_3.arff
-rw-r--r-- 1 fcharte fcharte 379839 May 17 12:03 musicout_MLeNN_TH_0.5_k_3.arff
fcharte@Volstag:~$
```

Figure 3: Files produced by the `resample()` function after running four resampling algorithms over one dataset.

Once all the processed datasets are available, the functionality provided by the `mldr` package [16] allows the user to analyze how each algorithm has affected the dataset through a series of metrics and plots.

Based on these examples, we did an experiment to test all the algorithms included in the `mldr.resampling` package. The `emotions` dataset is one of the best known in the literature, so we chose it. Table 2 shows the basic imbalance and coupling metrics before and after applying each of the methods. By analyzing these results, a decision can be made as to which algorithm is more appropriate for this MLD.

Method	Instances	MeanIR	SCUMBLE
None	593	1.478068	0.010952
LPROS	741	1.355174	0.007740
MLROS	745	1.400893	0.007265
MLRkNNOS	1935	1.229133	0.075575
MLSMOTE	1247	1.298585	0.004987
MLSOL	740	1.420507	0.009091
LPRUS	450	1.408794	0.010180
MLRUS	511	1.439544	0.010126
MLTL	592	1.474701	0.010870
MLUL	445	1.503003	0.011839
MLeNN	592	1.474701	0.010870
REMEDIAL	815	1.478068	0.000562

Table 2: Imbalance and label coupling metrics for the `emotions` dataset after applying the methods provided in the `mldr.resampling` package. Default values were used for all algorithms.

5. Conclusions

This paper has presented a novel software package, `mldr.resampling` for the R language, which for the first time provides the user with reference implementations of the most popular multilabel resampling algorithms.

Imbalance is a common problem in multilabel datasets, hence the usefulness of resampling methods as a model-independent approach to overcome this obstacle. However, there are no public implementations of many of the published algorithms. Our `mldr.resampling` package fills this gap by providing consistent implementations of eleven of these methods. The package is publicly available in CRAN, the official R repository, and users get immediate access to the source code. In addition, the package is designed for efficiency, minimizing computation by caching nearest neighbor information and parallelizing data processing.

Future plans include the addition of new resampling methods as they become available and with sufficient detail to allow their implementation, extending the functionality already available in the `mldr` family of packages.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research carried out in this study is part of the project "ToSmartEADs: Towards intelligent, explainable and precise extraction of knowledge in complex problems of Data Science" financed by the Ministry of Science, Innovation and Universities with code PID2019-107793GB-I00 / AEI / 10.13039 / 501100011033.

Required Metadata

Current executable software version

See Table 3.

Current code version

See Table 4.

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	0.2.1
S2	Permanent link to executables of this version	`https://github.com/madr0008/mldr.resampling`
S3	Legal Software License	MIT ($\geq$ 2)
S4	Computing platform/Operating System	Linux, OS X, Microsoft Windows
S5	Installation requirements & dependencies	data.table, e1071, mldr, pbapply, vecsets
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	`https://cran.r-project.org/web/packages/mldr.resampling/mldr.resampling.pdf`
S7	Support email for questions	mdavila@ujaen.es

Table 3: Software metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	0.2.1
C2	Permanent link to code/repository used of this code version	`https://github.com/madr0008/mldr.resampling`
C3	Legal Code License	MIT ($\geq$ 2)
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	R ($\geq$ 3.3.0)
C6	Compilation requirements, operating environments & dependencies	data.table, e1071, mldr, pbapply, vecsets
C7	If available Link to developer documentation/manual	`https://cran.r-project.org/web/packages/mldr.resampling/mldr.resampling.pdf`
C8	Support email for questions	mdavila@ujaen.es

Table 4: Code metadata (mandatory)

References

[1] F. Herrera, F. Charte, A. J. Rivera, M. J. del Jesus, Multilabel Classification. Problem analysis, metrics and techniques, Springer, 2016. doi:10.1007/978-3-319-41111-8.

[2] F. Charte, A. J. Rivera, M. J. del Jesus, F. Herrera, Addressing imbalance in multilabel classification: Measures and random resampling algorithms, Neurocomputing 163 (0) (2015) 3–16. doi:10.1016/j.neucom.2014.08.091.

[3] A. Luque, A. Carrasco, A. Martín, A. de las Heras, The impact of class imbalance in classification performance metrics based on the binary confusion matrix, Pattern Recognition 91 (2019) 216–231. doi:10.1016/j.patcog.2019.02.023.

[4] Y. Sun, A. K. Wong, M. S. Kamel, Classification of imbalanced data: A review, International journal of pattern recognition and artificial intelligence 23 (04) (2009) 687–719. doi:10.1142/S0218001409007326.

[5] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, G. Bing, Learning from class-imbalanced data: Review of methods and applications, Expert Systems with Applications 73 (2017) 220–239. doi:10.1016/j.eswa.2016.12.035.

[6] A. Menon, H. Narasimhan, S. Agarwal, S. Chawla, On the statistical consistency of algorithms for binary classification under class imbalance, in: S. Dasgupta, D. McAllester (Eds.), Proceedings of the 30th International Conference on Machine Learning, Vol. 28 of Proceedings of Machine Learning Research, PMLR, Atlanta, Georgia, USA, 2013, pp. 603–611.

[7] H. He, Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications, Wiley-IEEE, 2013. doi:10.1002/9781118646106.

[8] S. Kotsiantis, D. Kanellopoulos, P. Pintelas, et al., Handling imbalanced datasets: A review, GESTS international transactions on computer science and engineering 30 (1) (2006) 25–36.

[9] R. Mohammed, J. Rawashdeh, M. Abdullah, Machine learning with oversampling and undersampling techniques: overview study and experimental results, in: 2020 11th international conference on information and communication systems (ICICS), IEEE, 2020, pp. 243–248. doi:10.1109/ICICS49469.2020.239556.

[10] P. Sadhukhan, S. Palit, Reverse-nearest neighborhood based oversampling for imbalanced, multi-label datasets, Pattern Recognition Letters 125 (2019) 813–820. doi:10.1016/j.patrec.2019.08.009.

[11] F. Charte, A. J. Rivera, M. J. del Jesus, F. Herrera, MLSMOTE: Approaching imbalanced multilabel learning through synthetic instance generation, Knowledge-Based Systems 89 (2015) 385–397. doi:10.1016/j.knosys.2015.07.019.

[12] B. Liu, K. Blekas, G. Tsoumakas, Multi-label sampling based on local label imbalance, Pattern Recognition 122 (2022) 108294. doi:10.1016/j.patcog.2021.108294.

[13] R. M. Pereira, Y. M. Costa, C. N. Silla Jr, MLTL: A multi-label approach for the Tomek Link undersampling algorithm, Neurocomputing 383 (2020) 95–105. doi:10.1016/j.neucom.2019.11.076.

[14] F. Charte, A. Rivera, M. del Jesus, F. Herrera, MLeNN: A First Approach to Heuristic Multilabel Undersampling, in: Proc. 15th Int. Conf. Intelligent Data Engineering and Automated Learning, Salamanca, Spain, IDEAL'14, Vol. 8669 of LNCS, 2014, pp. 1–9. doi:10.1007/978-3-319-10840-7_1.

[15] F. Charte, A. Rivera, M. J. del Jesus, F. Herrera, Resampling Multilabel Datasets by Decoupling Highly Imbalanced Labels, in: Hybrid Artificial Intelligent Systems, Vol. 9121 of Lecture Notes in Computer Science, Springer International Publishing, 2015, pp. 489–501. doi:10.1007/978-3-319-19644-2_41.

[16] F. Charte, D. Charte, Working with multilabel datasets in R: The mldr package, The R Journal 7 (2) (2015) 149–162. doi:10.32614/RJ-2015-02.

[17] F. Charte, A. J. Rivera, D. Charte, M. J. del Jesus, F. Herrera, Tips, guidelines and tools for managing multi-label datasets: The mldr.datasets r package and the cometa data repository, Neurocomputing 289 (2018) 68–85. doi:10.1016/j.neucom.2018.02.011.

[18] D. Charte, F. Charte, S. García, F. Herrera, A snapshot on nonstandard supervised learning problems: taxonomy, relationships, problem transformations and algorithm adaptations, Progress in Artificial Intelligence 8 (1) (2019) 1–14. doi:10.1007/s13748-018-00167-7.

[19] S. Sun, A survey of multi-view machine learning, Neural Computing and Applications 23 (7-8) (2013) 2031–2038. doi:10.1007/s00521-013-1362-6.

[20] Z. H. Zhou, Multi-instance learning: A survey, Department of Computer Science & Technology, Nanjing University, Tech. Rep (2004).

[21] S. D. Robinson, Multi-label classification of contributing causal factors in self-reported safety narratives, Safety 4 (2018) 30. doi:10.3390/safety4030030.

[22] O. E. Dai, B. Demir, B. Sankur, L. Bruzzone, A novel system for content-based retrieval of single and multi-label high-dimensional remote sensing images, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (99) (2018) 1–18. doi:10.1109/JSTARS.2018.2832985.

[23] T. Liu, L. Chen, X. Pan, An integrated multi-label classifier with chemical-chemical interactions for prediction of chemical toxicity effects, Combinatorial chemistry & high throughput screening 21 (6) (2018) 403–410. doi:10.2174/1386207321666180601075428.

[24] F. Charte, A. J. Rivera, M. J. del Jesus, F. Herrera, QUINTA: A question tagging assistant to improve the answering ratio in

electronic forums, in: EUROCON 2015 - International Conference on Computer as a Tool (EUROCON), IEEE, 2015, pp. 1–6. doi:10.1109/EUROCON.2015.7313677.

[25] M. Zhang, Z. Zhou, A review on multi-label learning algorithms, IEEE Transactions on Knowledge and Data Engineering 26 (8) (2014) 1819–1837. doi:10.1109/TKDE.2013.39.

[26] E. Gibaja, S. Ventura, Multi-label learning: a review of the state of the art and ongoing research, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 4 (6) (2014) 411–444. doi:10.1002/widm.1139.

[27] E. Gibaja, S. Ventura, A tutorial on multilabel learning, ACM Computing Surveys 47 (3) (2015) 52:1–52:38. doi:10.1145/2716262.

[28] N. Japkowicz, S. Stephen, The class imbalance problem: A systematic study, Intelligent Data Analysis 6 (5) (2002) 429–449. doi:10.3233/IDA-2002-6504.

[29] A. Fernández, V. López, M. Galar, M. J. del Jesus, F. Herrera, Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches, Knowl. Based Systems 42 (2013) 97 – 110. doi:10.1016/j.knosys.2013.01.018.

[30] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics, Inf. Sciences 250 (2013) 113 – 141. doi:10.1016/j.ins.2013.07.007.

[31] S. Godbole, S. Sarawagi, Discriminative Methods for Multi-Labeled Classification, in: Advances in Knowledge Discovery and Data Mining, Vol. 3056, 2004, pp. 22–30. doi:10.1007/978-3-540-24775-3_5.

[32] F. Charte, A. J. Rivera, M. J. del Jesus, F. Herrera, Dealing with difficult minority labels in imbalanced mutilabel data sets, Neurocomputing 326 (2019) 39–53. doi:10.1016/j.neucom.2016.08.158.

[33] G. Tsoumakas, I. Vlahavas, Random k-labelsets: An ensemble method for multilabel classification, in: Proc. 18th European Conf. on Machine Learning, Warsaw, Poland, ECML'07, Vol. 4701, 2007, pp. 406–417. doi:10.1007/978-3-540-74958-5_38.

[34] G. Tsoumakas, I. Katakis, I. Vlahavas, Effective and Efficient Multilabel Classification in Domains with Large Number of Labels, in: Proc. ECML/PKDD Workshop on Mining Multidimensional Data, Antwerp, Belgium, MMD'08, 2008, pp. 30–44.

[35] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, Machine Learning 85 (2011) 333–359. doi:10.1007/s10994-011-5256-5.

[36] J. Read, B. Pfahringer, G. Holmes, Multi-label classification using ensembles of pruned sets, in: 8th International Conference on Data Mining, 2008. ICDM'08, IEEE, 2008, pp. 995–1000. doi:10.1109/ICDM.2008.74.

[37] J. Read, L. Martino, P. M. Olmos, D. Luengo, Scalable multi-output label prediction: From classifier chains to classifier trellises, Pattern Recognition 48 (6) (2015) 2096 – 2109. doi:10.1016/j.patcog.2015.01.004.

[38] B. Liu, G. Tsoumakas, Synthetic oversampling of multi-label data based on local label distribution, in: Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II, Springer, 2020, pp. 180–193.

[39] A. N. Tarekegn, M. Giacobini, K. Michalak, A review of methods for imbalanced multi-label classification, Pattern Recognition 118 (2021) 107965. doi:https://doi.org/10.1016/j.patcog.2021.107965.

[40] F. Charte, A comprehensive and didactic review on multilabel learning software tools, IEEE Access 8 (2020) 50330–50354. doi:10.1109/ACCESS.2020.2979787.

[41] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, Z. M. Jones, mlr: Machine learning in r, Journal of Machine Learning Research 17 (170) (2016) 1–5.

[42] A. Rivolli, A. C. P. L. F. de Carvalho, The utiml Package: Multi-label Classification in R, The R Journal 10 (2) (2018) 24–37. doi:10.32614/RJ-2018-041.
URL `https://doi.org/10.32614/RJ-2018-041`

[43] L.-P. Chen, et al., Netda: An r package for network-based discriminant analysis subject to multilabel classes, Journal of Probability and Statistics 2022 (2022).

[44] M. Popov, Multi-label Classification with MLPUGS, Comprehensive R Network Archive (2016).
URL `https://github.com/bearloga/MLPUGS`

[45] C. Schiebout, H. R. Frost, CAMML: multi-label immune cell-typing and stemness analysis for single-cell RNA-sequencing, in: PACIFIC SYMPOSIUM ON BIOCOMPUTING 2022, World Scientific, 2021, pp. 199–210.

[46] L. Gautier, rpy2 3.5.13 - R in Python (2023).
URL `https://rpy2.github.io`

[47] S. Urbanek, rJava: Low-level R to Java interface (2021).
URL `https://cran.r-project.org/web/packages/rJava/index.html`

[48] D. Eddelbuettel, Rinside, in: Seamless R and C++ Integration with Rcpp, Springer New York, New York, NY, 2013, pp. 127–137. doi:10.1007/978-1-4614-6868-4_9.

[49] D. L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, IEEE Transactions on Systems, Man, and Cybernetics (3) (1972) 408–421. doi:10.1109/TSMC.1972.4309137.

[50] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, J. Artificial Intelligence Res. 16 (2002) 321–357. doi:10.1613/jair.953.

[51] I. Tomek, Two modifications of CNN, IEEE Transactions On Systems, Man and Cybernetics 6 (11) (1976) 769–772. doi:10.1109/TSMC.1976.4309452.

[52] C. Stanfill, D. Waltz, Toward memory-based reasoning, Communications of the ACM 29 (12) (1986) 1213–1228. doi:10.1145/7902.7906.