The utiml Package: Multi-label Classification in R

by Adriano Rivolli and Andre C. P. L. F. de Carvalho

Abstract Learning classification tasks in which each instance is associated with one or more labels are known as multi-label learning. The implementation of multi-label algorithms, performed by different researchers, have several specificities, like input/output format, different internal functions, distinct programming language, to mention just some of them. As a result, current machine learning tools include only a small subset of multi-label decomposition strategies. The **utiml** package is a framework for the application of classification algorithms to multi-label data. Like the well known MULAN used with Weka, it provides a set of multi-label procedures such as sampling methods, transformation strategies, threshold functions, pre-processing techniques and evaluation metrics. The package was designed to allow users to easily perform complete multi-label classification experiments in the R environment. This paper describes the **utiml** API and illustrates its use in different multi-label classification scenarios.

Introduction

Multi-label classification (MLC) is a classification task where an instance can be simultaneously classified in more than one of the existing classes. Labeled data extracted from several domains, like text, web pages, multimedia (audio, image, videos), and biology are intrinsically multi-labeled. Additionally, the number of application domains with MLC data is growing fast.

Many current, real-world data science applications are MLC by nature. They are problems from very diverse domains, like labeling newspaper articles by subject and classification of proteins according to their functions. MLC algorithms have been successfully used in these and other MLC tasks (Diplaris et al., 2005). In a recent application, MLC algorithms were used to recommend food truck cuisines (Rivolli et al., 2017), assuming that a person can have more than one cuisine preference, and with the same level of preference.

Despite its growing relevance, there is a lack of comprehensive and easy to use tools for the R environment. A tool frequently used in MLC experiments is MULAN (Tsoumakas et al., 2011), which is a Java library built on top of Weka (Hall et al., 2009) to allow Weka users to deal with MLC data. Its popularity in the research community can be attributed to its ease of use, its large number and variation of its functionalities. The MLC alternative to Python users is the *scikit-multilearn* (Szymański, 2017), which provides a set of MLC algorithms and an interface for the MULAN library. Although other simpler tools, like MEKA (Read et al., 2016) and general data mining software (Gibaja and Ventura, 2015) include good functionalities to deal with MLC tasks, they address few MLC features and are not available in R.

It is important to mention that there are packages that offer some level of support for MLC in R. The most complete is the **mldr** package, an exploratory tool for the manipulation and analysis of MLC datasets (Charte and Charte, 2015). Although it does not contain MLC strategies, it supports the ARFF variation for MLC data, largely used for data mining and machine learning (ML) experiments, and has useful features, such as dataset characterization, MLC evaluation measures, and a rich user interface for the data exploration.

Some works use the **mlr** package, which was not specifically designed for MLC. As a result, it provides only a few multi-label strategies (Probst et al., 2017) and does not support the MLC ARFF format. In fact, it is a general purpose package, with an interface to more than one hundred algorithms that supports several ML tasks (Bischl et al., 2016). Another related package, **MLPUGS**, is a simple MLC package that contains only the implementation of the classifier chains (CC) strategy (Read et al., 2009).

Although the previous packages make it easier to perform some procedures related to MLC learning, their adoption in MLC experiments require more efforts from the developer/researcher than MULAN, available for Weka uses, which motivated the authors to design **utiml**, a more comprehensive, specific, easy to use, and extensible solution. The main features of the **utiml** package include:

- *Pre-processing techniques*: a set of techniques for the preparation and pre-possessing of MLC data to be used in experiments. These techniques deal with simple tasks, like removal of predictive attributes, instances and labels, replacement of nominal attribute values by numerical values, and data normalization.
- *Sampling*: a set of methods used to split MLC data through the holdout and k-fold methodologies.

Random or stratified strategies can be used for data partitioning.

- *Classification/Ranking*: the main MLC strategies. The transformation strategies support several base algorithms and the result can be seen as bipartition, probability/score, and ranking.
- *Threshold*: score-based and ranking-based threshold functions to be employed after the label prediction, so that bipartition values can be changed.
- *Evaluation*: traditional MLC evaluation measures and MLC confusion matrix for the summarization of classification result.

This paper describes the main aspects and resources of the **utiml** package. The current version is 0.1.4 and an updated list with all resources available will be maintained in the vignette document and the reference manual. The following section provides a brief review of MLC learning. Next, the package API is detailed, its resources are presented and some illustrative examples are provided. Finally, the main issues regarding the package use are highlighted in the summary section.

Multi-label classification learning

MLC tasks have attracted a growing attention in the ML community (de Carvalho and Freitas, 2009; Tsoumakas et al., 2010; Gibaja and Ventura, 2014). While in multi-class classification only a single class label is predicted, in MLC, more than one class label can be simultaneously predicted. In the same way as multi-class classification tasks can be seen as a generalization of binary classification tasks, which restricts to two the number of classes, MLC can be seen as a generalization of multi-class, which restricts to one the number of predicted classes (de Carvalho and Freitas, 2009). The main MLC approaches are *prediction of multiple labels*, *label ranking*, and *multi-label ranking* (Tsoumakas et al., 2010).

Multi-Label Classification (MLC), the most common task (Tsoumakas et al., 2010), induces a predictive model $h(x) \rightarrow Y$ from a set of training data, which later assigns one or more labels to each new example. This task can be formally defined as: let D be a set of labeled instances E, such that $D = \{E_1, E_2, ..., E_n\}$. Every labeled instance $E_i = (x_i, Y_i)$ is composed of $x_i = (x_{i1}, x_{i2}, ..., x_{id})$, which describes its position in a $\mathbb{R}^d$ input space, and $Y_i \subseteq L \mid L = \{\lambda_1, \lambda_2, ..., \lambda_q\}$, which describes a position in a $\{0,1\}^q$ output space.

The *Label Ranking* (LR) task can be characterized by a function $f(x, \lambda_i)$, which, for each class label, outputs a score value in the interval $[0.0, 1.0]$, indicating the relevance, confidence, or probability of instance x belonging to the class whose label is λ_i. The higher the score value, the better the ranking position. While MLC predicts bipartitions and LR predicts scores, *Multi-label Ranking* (MLR) generates both. Since MLC can be derived from the LR formulation (Gibaja and Ventura, 2015)[1], if a strategy can be used in the LR task, it can also be used in the two other tasks.

These models can be obtained by two approaches (Tsoumakas et al., 2010), *problem transformation* and *algorithm adaptation*. *Problem transformation* converts the original MLC task into a set of binary or multi-class classification subtasks. Afterwards, any classification algorithm, here called base algorithm, can be used to induce models for the subtasks. In the *algorithm adaptation* approach, the multi-label support is embedded into the algorithm structure. Thus, while transformation fits data to algorithms, adaptation fits algorithms to data (Zhang and Zhou, 2014).

The transformation approach can be performed in three different ways: *binary*, *pairwise*, and *powerset*. *Binary* transformation generates at least one dataset per label, as in the one-versus-all multiclass strategy. *Pairwise* transformation, instead, creates one dataset for each pair of labels, similarly to one-versus-one multiclass strategy. Finally, *powerset* is a multi-class transformation that uses labelsets as classes. The adaptation approach, on the other hand, modifies conventional ML algorithms, like Decision Tree Induction Algorithms (DT), K-Nearest Neighbors (KNN), Random Forest (RF), and Support Vector Machines (SVM).

Other steps required for the application of ML algorithms need to be adapted to deal with MLC tasks. For example, stratified sampling for MLC data must take into account multiple targets and the predictive performance evaluation must consider situations like partially correct results and ranking accuracy. A complete overview of the alternatives to deal with these issues can be seen in Zhang and Zhou (2014) and Gibaja and Ventura (2015).

[1]Where $h(x) = \{\lambda \mid f(x, \lambda) \geq \tau(x), \lambda \in L\}$, where $\tau(x)$ is a threshold function.

The utiml package

Handling multi-label classification data

The predictive performance of MLC tasks can be strongly affected by the use of data pre-processing techniques. For such, **utiml** uses the **mldr** package (Charte and Charte, 2015), which provides the support for data pre-processing. Moreover, when **utiml** is installed/loaded, the **mldr** package is automatically installed/loaded. Specially, it supports the MLC ARFF format, which has an additional XML file describing the label columns[2].

By default, the **mldr** package handles categorical data as `"character"`, instead of `"factor"`, which is not supported by the implementation of some traditional machine learning algorithms available in R, like Random Forest from the **randomForest** package. To address this limitation, the `mldata` function converts all text columns to factors of an `"mldr"` dataset. For example, the function `mldata` should be used to load the 'flags' dataset[3], contains categorical attributes, like

```
> flags <- mldata(mldr("flags"))
```

After a dataset is loaded, pre-processing techniques can be applied to it. Table 1 shows the pre-processing techniques available in the **utiml** package. All these functions receive an `"mldr"` dataset as argument and return a pre-processed version of this dataset.

Pre-processing function	Description
`fill_sparse_mldata(mdata)`	Exchanges the `NA` values present in the dataset to `0` or `""`, according to the attribute type.
`normalize_mldata(mdata)`	Re-scales all numerical attribute values to values between 0 and 1 according to the min-max transformation. The lowest value is modified to 0.0 and the highest value is converted to 1.0.
`remove_attributes(mdata, attributes)`	Removes the specified attributes from the dataset.
`remove_labels(mdata, labels)`	Removes the specified labels from the dataset.
`remove_unique_attributes(mdata)`	Removes from the dataset attributes whose values are the same for all instances.
`remove_unlabeled_instances(mdata)`	Removes from the dataset instances without class labels.
`remove_skewness_labels(mdata, t)`	Removes from the dataset highly infrequent or highly frequent labels, according to a specific threshold value. The threshold t indicates the minimum number of positive and negative instances associated with each label.
`replace_nominal_attributes(mdata)`	Replaces categorical attributes by binary attributes. An attribute with n different values will be mapped to $n - 1$ new columns containing binary values.

Table 1: Pre-processing techniques available in the **utiml** package

The **utiml** package also supports the main methodologies for data sampling, as shown in Table 2. The holdout and k-fold sampling can partition a dataset randomly and in a stratified way. They are selected by a parameter named `method`, which determines the sampling algorithm that creates the partitions. According to Sechidis et al. (2011), the accepted values are `"random"`, `"iterative"`, and `"stratified"`, where the latter two are different stratification options. The `"iterative"` process stratifies a MLC dataset considering each label independently, while `"stratified"` is based on the different combinations of labels, also known as labelset.

These techniques were designed to improve the **mldr** package and to simplify the data preparation for the learning step. Concerning the analysis of the MLC data, **utiml** does not provide additional resources in this current version. However, it is possible to use the **mldr** package, which enables the understanding and exploration of several data aspects through an interactive interface (Charte and Charte, 2015).

[2]The complete specification is available at http://mulan.sourceforge.net/format.html.
[3]This dataset, available at http://mulan.sourceforge.net/datasets-mlc.html.

Sampling function	Description
`create_holdout_partition(mdata, partitions, method)`	Splits the data into at least two distinct parts. The second parameter defines the name and size of the partitions and `method` defines the type of sampling.
`create_kfold_partition(mdata, k, method)`	Creates an object that contains the k distinct parts of the dataset using the `method` for splitting the folds. It should be used in combination with `partition_fold(object, fold)`, which provides the training and testing data to a specific `fold`. The parameter `object` is the result of `create_kfold_partition`.
`create_random_subset(mdata, instances, attributes, replacement)`	Creates a random subset of the dataset based on the proportion of `instances` and `attributes`. When `replacement=TRUE`, a same instance can appear one more time in the training data.
`create_subset(mdata, rows, cols)`	Creates a specific subset of the dataset based on the instances (`rows`) and attributes (`cols`) specified.

Table 2: Sampling functions available in the **utiml** package

The **utiml** package also provides two MLC datasets: `toyml`, a synthetic dataset generated by the Mldatagen tool (Tomás et al., 2014); and `foodtruck`, a dataset in which several food truck cuisines are mapped as labels (Rivolli et al., 2017).

Multi-label classification strategies

The classification strategies are the heart of the **utiml** package. Table 3 shows the strategies available in the current version of the package. Some of the implemented strategies, such as `brplus`, `ctrl`, `dbr`, `lift`, `prudent` and `rdbr` were not found by the authors in other tools.

Transformation strategies build the multi-label models by using a ML base algorithm. The `base.algorithm` parameter defines the base algorithm employed to create the internal models. Table 4 shows the ML algorithms currently supported by the package. Their use requires additional packages, as indicated in column *R function Called*[4]. For example, `"C5.0"` algorithm requires the **C50** package be installed. Only the `"MAJORITY"` and `"RANDOM"` algorithms require no additional packages.

The arguments of the transformation strategies follow the pattern:

1. `mdata`: an `"mldr"` dataset object.
2. `base.algorithm`: a base algorithm, as listed in Table 4.
3. *additional strategy parameters*: specific parameters for each strategy. While the BR strategy contains no additional parameters, the ensemble ECC receives 4 specific parameters, namely `m`, `subsample`, `attr.space` and `replacement`[5].
4. `...`: extra parameters used by the `base.algorithm` selected. As illustration, if `base.algorithm = "SVM"`, the extra parameters can be those defined in the `svm` function of the **e1071** package, such as `kernel`, `gamma`, `cost`, among others.
5. `cores`: number of cores used for the parallelization of the training phase. Note some classification strategies, as `lp`, ignore the parameter because the tasks can not be parallelized.
6. `seed`: a seed that ensures reproducibility. This is particularly important when the task is parallelized. In other words, if `cores = 1`, the `seed` effect is similar to that of `set.seed(seed)`. However, if the `cores` are higher than 1, the `set.seed(seed)` command will not guarantee the same result can be obtained, since the task will be performed in parallel.

[4]These packages are not installed together with **utiml**.

[4]The `J48` algorithm has no support for task parallelization.

[5]These parameters denote, respectively, number of models in the ensemble, proportion of instances, attributes, and a possible replacement of instances. The specific parameters of each strategy are reported in the reference manual.

Strategy function	Description	Approach[a]	Reference
`baseline`	Baseline	-	Metz et al. (2012)
`br`	Binary Relevance	BR	Tsoumakas et al. (2010)
`brplus`	BR+	BR, STA	Cherman et al. (2012)
`cc`	Classifier Chains	BR, CC	Read et al. (2009)
`clr`	Calibrated Label Ranking	PW	Brinker et al. (2006)
`ctrl`	ConTRolled Label correlation	BR, ENS	Li and Zhang (2014)
`dbr`	Dependent Binary Relevance	BR, STA	Montañes et al. (2014)
`ebr`	Ensemble of Binary Relevance	BR, ENS	Read et al. (2009)
`ecc`	Ensemble of Classifier Chains	BR, CC, ENS	Read et al. (2009)
`eps`	Ensemble of Pruned Set	ENS, PS	Read et al. (2008)
`homer`	Hierarchy Of Multi-label classifiER	HIE	Tsoumakas et al. (2008)
`lift`	Learning with Label specIfic FeaTures	BR, CLU	Zhang and Wu (2015)
`lp`	Label Powerset	PS	Tsoumakas and Katakis (2007)
`mbr`	Meta-BR, 2BR or stacking	BR, STA	Tsoumakas et al. (2009)
`mlknn`	Multi-label kNN	AD	Zhang and Zhou (2007)
`ns`	Nested Stacking	BR, CC	Senge et al. (2013)
`ppt`	Pruned Problem Transformation	PS	Read et al. (2008)
`prudent`	PRUned and confiDENT Stacking	BR, STA	Alali and Kubat (2015)
`ps`	Pruned Set	PS	Read (2008)
`rakel`	Random k-labelsets	ENS, PS	Tsoumakas and Vlahavas (2007)
`rdbr`	Recursive Dependent Binary Relevance	BR, ENS, STA	Rauber et al. (2014)
`rpc`	Ranking by Pairwise Comparison	PW	Hüllermeier et al. (2008)

[a] `AD` = Adaptation; `BR` = Binary transformation; `CC` = Chain of classifiers; `CLU` = Clustering based; `ENS` = Ensemble; `HIE` = Hierarchy; `PS` = Powerset transformation; `PW` = Pairwise transformation; `STA` = Stacking

Table 3: Strategies available in the **utiml** package

`base.algorithm` value	Description	R function Called
`"C5.0"`	C5.0 Decision Trees	`C50::C5.0`
`"CART"`	Classification and regression trees	`rpart::rpart`
`"KNN"`	K Nearest Neighbor	`kknn::kknn`
`"NB"`	Naive Bayes	`e1071::naiveBayes`
`"RF"`	Random Forest	`randomForest::randomForest`
`"SMO"`	Sequential Minimal Optimization	`RWeka::SMO`
`"SVM"`	Support Vector Machine	`e1071::svm`
`"XGB"`	eXtreme Gradient Boosting	`xgboost::xgboost`
`"MAJORITY"`	Majority class prediction	-
`"RANDOM"`	Random prediction	-

Table 4: Base algorithms available in the **utiml** package

After the creation of a MLC model, the model can be applied to new data through the S3 `predict` method. The arguments of `predict` are:

1. `object`: a multi-label classifier.
2. `newdata`: a `"matrix"`, `"data.frame"` or `"mldr"` object, containing the data to be classified.
3. *additional model parameters*: specific parameters for each model. For example, the vote scheme to be used in the `ecc` prediction function can be defined[6].
4. `probability`: a logical value that indicates if the prediction result should be probability/score or bipartition. If `TRUE`, a probability result is returned; otherwise, the bipartition is obtained. The result can be changed, as observed next.
5. `...`: extra parameters based on the `predict` method related to the `base.algorithm` selected.
6. `cores`: number of cores used for the parallelization of the prediction phase. Some models, like CC, ignore this parameter because the tasks cannot be parallelized.
7. `seed`: a seed that ensures reproducibility. This is particularly important when the task is parallelized and the base algorithm is not deterministic.

The prediction result is an `"mlresult"` type object and can be used directly as a matrix, where each column is a label and each row is an instance. To change the type of result to bipartition, probability/score or a ranking matrix, the functions `as.bipartition`, `as.probability`, and `as.ranking`, respectively, can be used.

[6] The specific parameters of each predict method are reported in the reference manual.

Multi-label post-processing

Threshold functions adjust the bipartition result according to the score/probabilities predicted by the predictive models. In MLC learning, these functions can be score-based or rank-based, depending on the type of data used to define the threshold values. A single threshold value for all labels is named global threshold, the use of one value per label is named label-wise, and the use of one value per instance is named instance-wise (Al-Otaibi et al., 2014).

Table 5 shows the threshold functions available in the **utiml** package. All of them receive a probability/score matrix or an `"mlresult"` object as input and return a new `"mlresult"` object with different bipartitions as output. The only exception is `scut_threshold`, which returns threshold values, instead of bipartitions, and should be combined with the `fixed_threshold` function. Additionally, the `subset_correction` can be used as a threshold function (Senge et al., 2013). It changes the bipartition based on the labelsets present in the training data and outputs only the known labelsets.

Threshold function	Description	Approach
`fixed_threshold(prediction, threshold)`	Applies a fixed global or label-wise threshold.	score-based
`lcard_threshold(prediction, cardinality)`	Applies an instance-wise threshold using the cardinality measure.	rank-based
`mcut_threshold(prediction)`	Applies an instance-wise threshold and selects the subset of labels of the highest interval between two sorted scores.	score-based
`pcut_threshold(prediction, ratio)`	Applies a global or label-wise threshold using the ratio value to define the proportion of instances that will be relevant.	score-based
`rcut_threshold(prediction, k)`	Applies an instance-wise threshold and defines the k labels with highest scores as relevant.	rank-based
`scut_threshold(prediction, expected, loss.function)`	Returns a label-wise threshold using a loss function that minimizes the difference between the value predicted and the expected prediction value.	score-based

Table 5: Threshold functions available in the **utiml** package

Finally, **utiml** also supports the evaluation of MLC models. The `multilabel_evaluate` and `multilabel_confusion_matrix` functions can be used during the evaluation. The first calculates the traditional evaluation measures also available in **mldr** and MULAN, whereas the second generates a multi-label confusion matrix (`"mlconfmat"` object) detailing labels and instances.

The `multilabel_evaluate` function receives an `"mlresult"` or an `"mlconfmat"` object and the desired evaluation measures. One or more measures, likewise one or more group of measures can be adopted. Figure 1 shows the `measures` values, currently supported. A complete review of MLC evaluation measures can be found in Zhang and Zhou (2014) and Gibaja and Ventura (2015). Moreover, if the hyperparameter `labels=TRUE`, the return will be a list that contains the multi-label and labels' results detailed.

Default options

The **utiml** package uses the `option` function to customize some default parameters. For example, the default base algorithm for all transformation strategies is `"SVM"`. The `utiml.base.algorithm` option can be used to change this parameter value. Table 6 shows the option's names, a brief description of each option parameter value, and their default value. The following code defines "Random Forest" as the default base algorithm and sets the default number of cores to 8, to illustrate the setting of the options.

```
> options(utiml.base.algorithm = "RF", utiml.cores=8)
```

The `utiml.empty.prediction` option defines whether the MLC strategies can predict no labels for one or more instances. Among the alternatives to avoid an empty prediction (Liu and Chen, 2015), **utiml** outputs the labels with the highest probability/score. It must be observed that this option may

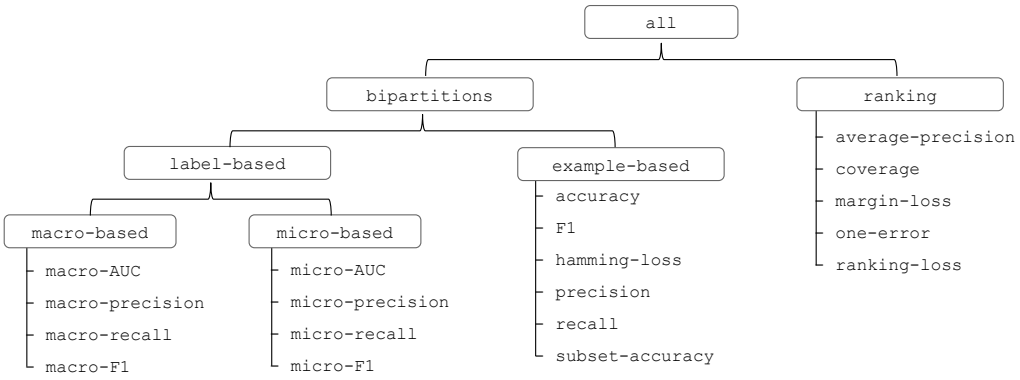

Figure 1: Groups of multi-label evaluation measures

Option parameter values	Description	Default value
`utiml.base.algorithm`	Default base algorithm used in the transformation strategies.	`"SVM"`
`utiml.cores`	Default number of cores used to parallelize the tasks.	`1`
`utiml.seed`	Default seed used by the MLC strategies.	`NA`
`utiml.use.probs`	Default type of the expected prediction results. If `TRUE`, the expected prediction is in the probability/score format, otherwise, a bipartition prediction is expected.	`TRUE`
`utiml.empty.prediction`	Default option concerning empty predictions. If `TRUE`, predictions can contain instances without labels, otherwise, the most important label is considered relevant and no empty predictions are obtained.	`FALSE`

Table 6: Options used by the **utiml** package

directly interfere with the result of the bipartition evaluation measures. Thus, it must be set according to the characteristics of the experiment being carried out.

How to use the package for multi-label classification experiments

The `toyml` dataset was used in the examples illustrated in this section. As `toyml` has two irrelevant attributes (`"iatt8"` and `"iatt9"`) and one redundant (`"ratt10"`) attribute, the pre-processing `remove_attributes` function can be applied to remove them.

```
> new.toyml <- remove_attributes(toyml, c("iatt8", "iatt9", "ratt10"))
```

```
> pre.process <- function (mdata) {
+   aux <- remove_skewness_labels(mdata, 5)  # Remove infrequent labels (less than 5)
+   aux <- remove_unlabeled_instances(aux)   # Remove instances without labels
+   aux <- remove_unique_attributes(aux)     # Remove constant attributes
+   return(mdata)
+ }
```

As `toyml` is already normalized and the dataset has a small number of instances, no other pre-processing technique is required. Thus, the `pre.process` function has no effect if applied in this case. For other datasets, the same procedure can be useful for their preparation for MLC experiments. Two

scenarios that illustrate the use of **utiml** for the development of MLC experiments are presented next. Finally, a simple experimental analysis is performed using the `foodtruck` dataset, illustrating the use of the package in a more realist scenario.

Training and test experiment example using holdout

This example shows a MLC experiment using holdout, in which 70% of the dataset instances are used for training and 30% for test. A BR model that uses "Random Forest" as a base algorithm is induced and applied to the test instances. Next, predictions are assessed using MLC evaluation measures.

```
> set.seed(123)
> ds <- create_holdout_partition(new.toyml, c(train=0.7, test=0.3))
> model <- br(ds$train, "RF")
> predictions <- predict(model, ds$test)

> results <- multilabel_evaluate(ds$test, predictions, c("example-based", "macro-F1"))
> round(results, 4)
## accuracy      F1  hamming-loss  macro-F1  precision  recall  subset-accuracy
##   0.6444  0.7411        0.1933    0.4015     0.7833  0.7722           0.3000
```

A MLC baseline can be included among the strategies being experimentally compared. In the following code, the general baseline (Metz et al., 2012) is induced. A subtle difference is observed in `"hamming-loss"` and `"F1"` measures in favor of the BR model. The small number of labels are due to very common combinations of them found in `toyml` favors the general baseline.

```
> base.preds <- predict(baseline(ds$train, "general"), ds$test)
> base.res <- multilabel_evaluate(ds$test, base.preds, c("hamming-loss", "F1"))

> round(base.res, 4)
##     F1  hamming-loss
## 0.7311        0.2000
```

In both examples, the test set predictions compose an `"mlresult"` object and, by default, show the score/probability produced by the base algorithm. For example:

```
> head(predictions)
##       y1    y2    y3    y4    y5
## 23 0.336 0.640 0.178 0.922 0.122
## 19 0.106 0.860 0.366 0.670 0.280
## 62 0.094 0.776 0.526 0.688 0.090
## 1  0.196 0.618 0.204 0.758 0.162
## 67 0.090 0.964 0.210 0.700 0.234
## 92 0.060 0.856 0.162 0.598 0.454
```

The `as.bipartition` and `as.ranking` functions can be used to change, respectively, the probability/score to a bipartition matrix or the raking values, as illustrated next. Optionally, a threshold function can be applied to change the bipartitions.

```
> head(as.bipartition(predictions))
##    y1 y2 y3 y4 y5
## 23  0  1  0  1  0
## 19  0  1  0  1  0
## 62  0  1  1  1  0
## 1   0  1  0  1  0
## 67  0  1  0  1  0
## 92  0  1  0  1  0

> head(as.ranking(predictions))
##    y1 y2 y3 y4 y5
## 23  3  2  4  1  5
## 19  5  1  3  2  4
## 62  4  1  3  2  5
## 1   4  2  3  1  5
## 67  5  1  4  2  3
## 92  5  1  4  2  3
```

```
> head(mcut_threshold(predictions))
##    y1 y2 y3 y4 y5
## 23  0  1  0  1  0
## 19  0  1  0  1  0
## 62  0  1  1  1  0
## 1   0  1  0  1  0
## 67  0  1  0  1  0
## 92  0  1  0  1  1
```

Three different ECC models are created in the following code to illustrate the use of different parameters and base algorithms. Each model uses a specific base algorithm and configuration, which, consequently, results in different models for the same data and MLC strategy.

```
> # Using KNN with k = 5 and changing ECC parameters
> model1 <- ecc(ds$train, "KNN", m=7, subsample=0.8, k=5)

> # Using C5.0 and changing ECC parameters
> model2 <- ecc(ds$train, "C5.0", subsample=0.6, attr.space=1)

> # Using SVM with cost = 10 and gamma = 0.5 and default ECC parameters
> model3 <- ecc(ds$train, "SVM", cost=10, gamma=0.5)
```

By default, the `create_holdout_partition` function creates two random partitions (train and test) with 70% and 30% of the dataset instances, respectively. The number of partitions, sizes and sampling method can be modified. The following code shows how to create three, label-stratified partitions, named `"train"`, `"test"`, and `"val"` with 70%, 20%, and 10% of the instances, respectively. The `"val"` partition can be used in a validation step for model selection or hyperparameter tuning.

```
> partitions <- c(train=0.7, test=0.2, val=0.1)
> strat <- create_holdout_partition(new.toyml, partitions, "iterative")
```

Training and test example experiment using k-fold cross validation

This section shows some examples of how to perform cross-validation MLC experiments. The `cv` method can be used to encapsulate the whole procedure, which simplifies the respective task, such that a 10-fold stratified cross-validation can be performed with few lines of code. For instance, the RAkEL strategy using the `"SVM"` base algorithm can be evaluated in the following way:

```
# Defining the evaluation measures
> measures <- c("hamming-loss", "subset-accuracy", "one-error")

# Running 10-fold cross validation
> results <- cv(new.toyml, method="rakel", base.algorith="SVM", cv.folds=10,
+               cv.sampling="stratified", cv.measures=measures, cv.seed=123)

> round(results, 4)
##   hamming-loss       one-error subset-accuracy
##          0.212           0.160           0.240
```

To obtain detailed results by folds and/or labels, the hyperparameter `cv.results=TRUE` can be set. In this case, a list is returned where the multi-label and labels' results can be obtained as illustrated in the next example.

```
> results <- cv(new.toyml, method="rakel", base.algorith="SVM", cv.results=TRUE,
+               cv.sampling="stratified", cv.measures=measures, cv.seed=123)

> t(results$multilabel)
##                 [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## hamming-loss     0.2 0.18 0.18  0.2 0.22 0.22 0.22  0.2 0.24  0.26
## one-error        0.3 0.20 0.20  0.2 0.10 0.00 0.00  0.2 0.20  0.20
## subset-accuracy  0.3 0.30 0.30  0.3 0.20 0.20 0.20  0.2 0.20  0.20

> round(sapply(results$labels, colMeans), 4)
##            y1   y2   y3     y4   y5
## accuracy 0.83 0.78 0.81 0.6900 0.83
```

```
## balacc   0.50 0.50 0.50 0.5054 0.50
## TP       0.00 7.80 0.00 6.8000 0.00
## TN       8.30 0.00 8.10 0.1000 8.30
## FP       0.00 2.20 0.00 3.0000 0.00
## FN       1.70 0.00 1.90 0.1000 1.70
```

Any MLC strategy can be used in the cv method, as well as specific hyperparameters for them. Additionally, the procedure can be parallelized, using `cv.cores`. The next example shows the ECC algorithm with specific hyperparameters being executed using 5 folds and parallelized in 4 cores.

```
> results <- cv(new.toyml, method="ecc", base.algorith="RF", subsample=0.9,
+               attr.space=0.9, cv.folds=5, cv.cores=4)
```

Finally, to perform a cross-validation procedure manually, the methods `create_kfold_partition` and `partition_fold`, can be used to create the folds and obtain the train and test dataset for each one of them, respectively. A good example, is the code used in the cv method, such that

```
...
> cvdata <- create_kfold_partition(mdata, cv.folds, cv.sampling)
> results <- parallel::mclapply(seq(cv.folds), function (k){
+    ds <- partition_fold(cvdata, k)
+    model <- do.call(method, c(list(mdata=ds$train), ...))
+    pred <- predict(model, ds$test, ...)
+    multilabel_evaluate(ds$test, pred, cv.measures, labels=TRUE)
+ }, mc.cores=cv.cores)
...
```

Experiments with the food truck dataset

In order to show how the package can be used in a real world problem, this section illustrates the use of the **utiml** to perform an exploratory analysis of the food truck dataset (Rivolli et al., 2017). First, the `br` strategy is evaluated with different ML base algorithms ("C5.0", "RF", "SVM" and "XGB") to identify the algorithm that produces the best macro and micro-F1 results.

```
> measures <- c("macro-F1", "micro-F1")
> algorithms <- c("C5.0", "RF", "SVM", "XGB")

> res <- sapply(algorithms, function(alg) {
+   cv(foodtruck, "br", base.algorithm=alg, cv.measures=measures, cv.seed=1)
+ })

> round(res, 4)
##            C5.0     RF    SVM    XGB
## macro-F1 0.1764 0.1824 0.1188 0.1827
## micro-F1 0.4856 0.5340 0.4835 0.5130
```

Regarding the macro-F1 measure, XGB presented the best result, however the difference observed between RF and C5.0 was small. For micro-F1, RF obtained the best result, followed closely by XGB. The differences observed between the macro and micro measures, independently of the base algorithm, may indicate that some infrequent labels had a poor F1 performance. To analyze this hypothesis, `br` was run again with RF, which obtained the best performance in the previous cross-validation procedure, with a new data subset. In the confusion matrix for the induced model, some patterns can be observed.

The following code shows that six labels (`mexican_food`, `chinese_food`, `japanese_food`, `arabic_food`, `healthy_food` and `fitness_food`) had no True Positive (TP) and False Positive (FP) predictions. Thus, for these labels, all instances were predicted as negative. This explains the difference observed between the macro and micro-F1 result, since the macro-F1 is the average labels' F1, which is 0 for these labels.

```
> set.seed(1)
> ds <- create_holdout_partition(foodtruck, method="iterative")

> model <- br(ds$train, "RF")
> pred <- predict(model, ds$test)

> cm <- multilabel_confusion_matrix(ds$test, pred)
```

```
> as.matrix(cm)
##                 TP  TN FP FN
## street_food     88   5 32  0
## gourmet         13  81  8 23
## italian_food     1 113  0 11
## brazilian_food   3 104  0 18
## mexican_food     0 113  0 12
## chinese_food     0 121  0  4
## japanese_food    0 115  0 10
## arabic_food      0 118  0  7
## snacks           7 103  2 13
## healthy_food     0 116  0  9
## fitness_food     0 116  0  9
## sweets_desserts 11  66 13 35
```

It must be observed that the `cm` object is a list containing several information about the prediction, like the confusion matrix values summarized by instances and labels. Any evaluation measure can be computed using only the information provided by this object. As an example, the next code summarizes the proportion of instances and the number of labels correctly predicted in the previous example. The results show that the BR model was not able to predict a correct label for almost 20% of the test instances; around 65% of the instances were correctly predicted with a single label; 12% were correctly predicted with 2 labels; and 3% were correctly predicted with 3 labels.

```
> prop.table(table(cm$TPi))
##     0     1     2     3
## 0.200 0.648 0.120 0.032
```

These results show a researcher can simulate new scenarios and explore different solutions in order to improve the predictive performance in a MLC task. The **utiml** package offers several resources that simplify the most basic and recurrent procedures adopted in the MLC domain.

Summary

Data classification is one of the main ML tasks. Although ML classification algorithms are usually designed and employed for single label classification tasks, in several application domains, an instance can have more than one class label. This paper introduced the **utiml** package, which provides several functions for MLC experiments in R. Similarly to MULAN, one of the most popular MLC tools, **utiml** offers a wide set of functionalities. The provided functions implement procedures that cover several MLC-related tasks, which include data pre-processing, data sampling model induction, optimization and evaluation of MLC models. The package **utiml** also supports the intrinsic parallelization of tasks and allows the reproducibility of MLC experiments.

To the best of the authors knowledge, some of the features present in **utiml** are not available in any other R tool, such as the implementation of MLC stratification (Sechidis et al., 2011), baselines (Metz et al., 2012), thresholds (Al-Otaibi et al., 2014) and an option that allow the users to avoid the empty prediction problem (Liu and Chen, 2015). Moreover, as in MULAN which enables users to take advantage of the resources available in the Weka environment, **utiml** users can benefit from the several libraries available in R.

The most important limitation of this package is that some common MLC procedures, like feature selection, imbalanced data, and classification strategies based on the algorithm adaptation approach are not available yet. They will be implemented in the future as a natural progression of this work and will be included in the next versions of the **utiml** package. The authors encourage other developers to integrate their own algorithms in the **utiml** package[7], so that it becomes a more robust and complete MLC package.

Bibliography

R. Al-Otaibi, P. Flach, and M. Kull. Multi-Label Classification: A Comparative Study on Threshold Selection Methods. In *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD 2014*, Nancy, France, 2014. [p29, 34]

[7]The source and project details are available on the https://github.com/rivolli/utiml page

A. Alali and M. Kubat. PruDent: A Pruned and Confident Stacking Approach for Multi-Label Classification. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2480–2493, 2015. URL https://doi.org/10.1109/tkde.2015.2416731. [p28]

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine Learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL http://jmlr.org/papers/v17/15-066.html. [p24]

K. Brinker, J. Fürnkranz, and E. Hüllermeier. A unified model for multilabel classification and ranking. In *Proceedings of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy*, pages 489–493, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press. ISBN 1-58603-642-4. URL http://dl.acm.org/citation.cfm?id=1567016.1567123. [p28]

F. Charte and F. D. Charte. Working with Multilabel Datasets in R: The mldr Package. *The R Journal*, 7 (2):149–162, 2015. [p24, 26]

E. A. Cherman, J. Metz, and M. C. Monard. Incorporating Label Dependency into the Binary Relevance Framework for Multi-Label Classification. *Expert Systems with Applications*, 39(2):1647–1655, 2012. URL https://doi.org/10.1016/j.eswa.2011.06.056. [p28]

A. C. P. L. F. de Carvalho and A. A. Freitas. *A Tutorial on Multi-Label Classification Techniques*, chapter 8, pages 177–195. Springer-Verlag, Berlin, Heidelberg, 2009. URL https://doi.org/10.1007/978-3-642-01536-6_8. [p25]

S. Diplaris, G. Tsoumakas, P. A. Mitkas, and I. Vlahavas. Protein Classification with Multiple Algorithms. In *Proceedings of the 10th Panhellenic Conference on Informatics*, pages 448–456, Volos, Greece, 2005. [p24]

E. Gibaja and 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):411–444, 2014. URL https://doi.org/10.1002/widm.1139. [p25]

E. Gibaja and S. Ventura. A Tutorial on Multilabel Learning. *ACM Computing Surveys*, 47(3):1–38, 2015. URL https://doi.org/10.1145/2716262. [p24, 25, 29]

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009. URL https://doi.org/10.1145/1656274.1656278. [p24]

E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker. Label Ranking by Learning Pairwise Preferences. *Artificial Intelligence*, 172(16-17):1897–1916, 2008. URL https://doi.org/10.1016/j.artint.2008.08.002. [p28]

Y. Li and M. Zhang. Enhancing Binary Relevance for Multi-Label Learning with Controlled Label Correlations Exploitation. In *Proceedings of the 13th Pacific Rim International Conference on Artificial Intelligence*, pages 91–103, Gold Coast, Australia, 2014. Springer-Verlag. URL https://doi.org/10.1007/978-3-319-13560-1_8. [p28]

S. M. Liu and J.-H. Chen. An Empirical Study of Empty Prediction of Multi-Label Classification. *Expert Systems with Applications*, 42(13):5567–5579, 2015. URL https://doi.org/10.1016/j.eswa.2015.01.024. [p29, 34]

J. Metz, L. F. de Abreu, E. A. Cherman, and M. C. Monard. On the Estimation of Predictive Evaluation Measure Baselines for Multi-Label Learning. In *13th Ibero-American Conference on AI*, pages 189–198, Cartagena de Indias, Colombia, 2012. URL https://doi.org/10.1007/978-3-642-34654-5_20. [p28, 31, 34]

E. Montañes, R. Senge, J. Barranquero, J. R. Quevedo, J. J. del Coz, and E. Hüllermeier. Dependent Binary Relevance Models for Multi-Label Classification. *Pattern Recognition*, 47(3):1494–1508, 2014. URL https://doi.org/10.1016/j.patcog.2013.09.029. [p28]

P. Probst, Q. Au, G. Casalicchio, C. Stachl, and B. Bischl. Multilabel Classification with R Package mlr. *The R Journal*, 9(1):352–369, 2017. [p24]

T. W. Rauber, L. H. Mello, V. F. Rocha, D. Luchi, and F. M. Varejão. Recursive Dependent Binary Relevance Model for Multi-Label Classification. In *Proceedings of the 14th Ibero-American Conference on AI*, pages 206–217, Santiago de Chile, Chile, 2014. Springer-Verlag. URL https://doi.org/10.1007/978-3-319-12027-0. [p28]

J. Read. A Pruned Problem Transformation Method for Multi-Label Classification. In *Proceedings of the New Zealand Computer Science Research Student Conference*, pages 143–150, 2008. [p28]

J. Read, B. Pfahringer, and G. Holmes. Multi-Label Classification Using Ensembles of Pruned Sets. In *Proceedings of the IEEE International Conference on Data Mining*, pages 995–1000, Pisa, Italy, 2008. IEEE. URL https://doi.org/10.1109/icdm.2008.74. [p28]

J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier Chains for Multi-Label Classification. In *Proceedings of the European Conference*, volume 5782, pages 254–269, Bled, Slovenia, 2009. URL https://doi.org/10.1007/978-3-642-04174-7_17. [p24, 28]

J. Read, P. Reutemann, B. Pfahringer, and G. Holmes. MEKA: A Multi-Label / Multi-Target Extension to WEKA. *Journal of Machine Learning Reasearch*, 17:1–5, 2016. URL http://jmlr.org/papers/v17/12-164.html. [p24]

A. Rivolli, L. C. Parker, and A. C. P. L. F. de Carvalho. Food truck recommendation using multi-label classification. In E. Oliveira, J. Gama, Z. Vale, and H. Lopes Cardoso, editors, *Progress in Artificial Intelligence*, pages 585–596. Springer-Verlag, 2017. [p24, 27, 33]

K. Sechidis, G. Tsoumakas, and I. Vlahavas. On the Stratification of Multi-Label Data. In *Lecture Notes in Computer Science*, volume 6913 LNAI, pages 145–158, 2011. URL https://doi.org/10.1007/978-3-642-23808-6_10. [p26, 34]

R. Senge, J. J. del Coz, and E. Hüllermeier. Rectifying Classifier Chains for Multi-Label Classification. In *Workshop of Lernen, Wissen & Adaptivität (LWA 2013)*, pages 162–169, Bamberg, Germany, 2013. [p28, 29]

P. Szymański. A scikit-based python environment for performing multi-label classification. *arXiv preprint arXiv:1702.01460*, 2017. [p24]

J. T. Tomás, N. Spolaôr, E. A. Cherman, and M. C. Monard. A Framework to Generate Synthetic Multi-Label Datasets. In *Proceedings of the XXXIX Latin American Computing Conference (CLEI 2013)*, volume 302, pages 155–176, Naiguata, Venezuela, 2014. URL https://doi.org/10.1016/j.entcs.2014.01.025. [p27]

G. Tsoumakas and I. Katakis. Multi-Label Classification: An Overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007. URL https://doi.org/10.4018/jdwm.2007070101. [p28]

G. Tsoumakas and I. Vlahavas. Random k -Labelsets: An Ensemble Method for Multilabel Classification. In *Proceedings of the European Conference on Machine Learning*, pages 406–417, Warsaw, Poland, 2007. URL https://doi.org/10.1007/978-3-540-74958-5_38. [p28]

G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery, Workshop on Mining Multidimensional Data*, pages 30–44, Antwerp, Belgium, 2008. [p28]

G. Tsoumakas, E. Loza Mencía, I. Katakis, S.-H. Park, and J. Fürnkranz. On the Combination of Two Decompositive Multi-Label Classification Methods. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery, Workshop on Preference Learning*, pages 114–129, Bled, Slovenia, 2009. [p28]

G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining Multi-Label Data. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, chapter 34, pages 667–685. Springer-Verlag, 2 edition, 2010. ISBN 0387244352. URL https://doi.org/10.1007/978-0-387-09823-4_34. [p25, 28]

G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas. MULAN: A Java Library for Multi-Label Learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011. [p24]

M.-L. Zhang and L. Wu. Lift: Multi-Label Learning with Label-Specific Features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(1):107–120, 2015. URL https://doi.org/10.1109/tpami.2014.2339815. [p28]

M.-L. Zhang and Z.-H. Zhou. A Review on Multi-Label Learning Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, 2014. ISSN 1041-4347. URL https://doi.org/10.1109/tkde.2013.39. [p25, 29]

M.-L. L. Zhang and Z.-H. H. Zhou. ML-KNN: A Lazy Learning Approach to Multi-Label Learning. *Pattern Recognition*, 40(7):2038–2048, 2007. URL https://doi.org/10.1016/j.patcog.2006.12.019. [p28]

Adriano Rivolli
Federal University of Technology - Parana (UTFPR)
Cornélio Procópio - PR
Brazil
rivolli@utfpr.edu.br

Andre C. P. L. F. de Carvalho
Institute of Mathematical and Computer Sciences (ICMC)
University of São Paulo (USP)
São Carlos - SP
Brazil
andre@icmc.usp.br