Library for Multi-instance Multi-label learning (MIML)

User Manual
MIML version 1.0
June 20, 2020

Álvaro Andrés Belmonte Pérez
Amelia Zafra Gómez
Eva Lucrecia Gibaja Galindo

Contents

Acronyms

BR: Binary Relevance

DD: Diverse Density

ECC: Ensemble of Classifier Chains

EPS: Ensemble of Pruned Sets

kNN: k Nearest Neighbors

LP: Label Powerset

LR: Label Ranking

MD: Multi-Dimensional

MI: Multi-Instance

MIML: Multi-Instance Multi-Label

MILR: Multiple-Instance Logistic Regression

MLC: Multi-Label Classification

MLR: Multi-Label Ranking

ML: Multi-Label

MOR: Multi-Output Regression

MT: Multi-Task

RA*k*EL: Random k-lAbELset

SVM: Support Vector Machine

CHAPTER 1

Introduction

In recent years, machine learning and data mining community has had to face more complex classification problems, being hard to find a proper representation of information. Experience has shown that finding an accurate representation, capable of representing all relationships and interactions in the data, has a direct effect on a more effective solution to the problem.

This fact has led to new learning paradigms that have emerged with the aim of representing objects in a more flexible way and solving problems that were not adequately solved with traditional approaches. In exchange of this flexibility, more complexity in data representation is introduced. In this context, Multi-Instance (MI) learning is presented as a more flexible learning paradigm to represent the input space. In MI, each object is represented by a pattern, a.k.a. *bag*, containing a variable number of instances, all of them with the same number of attributes [1]. This representation associates an object with multiple observations or configurations that allow a more flexible representation of the input space as alternative descriptions [1], components [2], or showing an evolution in time [3].

On the other hand, Multi-Task (MT) [4] learning represents the output space in a more flexible way than the traditional paradigms, since each object can belong to several classes. Among these approaches, one of the most popular is the so-called ML learning in which patterns in the training set can belong simultaneously to a set of binary classes (*labels*) [5]. Other MT paradigms are Multi-Dimensional (MD) learning, in which outputs are nominal [6], and Multi-Output Regression (MOR) in which outputs are continuous and numeric [7].

In this context, Multi-Instance Multi-Label (MIML) has emerged as a promising option that allows a more flexible representation of the input space and the output space. On one hand, MI representation introduces a more flexible representation of input space associating a pattern with multiple instances (*bag*). On the other hand, ML representation introduces a more flexible representation of the output space associating a pattern with a set of classes (*labels*). For instance, in image classification, an image could be represented by multiple instances being each one a region in the image and each image could have several labels (e.g. *cloud*, *lion*, *landscape*). MIML allows to carry out a natural formulation of complex objects in real problems such as texts and images categorization [8, 2, 9], audio and video detection [10] or bioinformatics [11, 12].

Currently, there are available several libraries to work in MI and ML learning such as Weka [13] for MI learning and Mulan [14] or Meka [15] for ML learning. Nevertheless, MIML can not be addressed with the former libraries. To the best knowledge of the authors, the only publicly available algorithms to solve MIML problems have been developed by research group LAMDA [16]. The main limitations of these implementations are the fact of being in MATLAB so a software license is needed to execute them and that they are not integrated in a library so each algorithm has a specific configuration to be able to run and a specific input and output format. This fact complicates seriously the development of experimental studies and new proposals.

The main motivation of this work is the development of a Java MIML library. MIML library is a modular library which makes easier to run and develop MIML classification algorithms to solve MIML problems. The library considers both methods which attempt to solve the problem directly and methods which transform previously the problem to a MI or to a ML, and then solve the problem using one of those learning frameworks.

This library is based on the Weka and MULAN libraries, so researchers on MIML who use any of these libraries will be familiar with its structure and format. Among its most relevant characteristics we can highlight:

- It uses a data format designed specifically for MIML learning, it has a set of developed algorithms that work directly with this format.
- It allows to transform the problem and use MI classifiers implemented on Weka framework and ML classifiers implemented on Mulan framework in a MIML context.
- It facilitates the design and development of new models that solve classification problems with a MIML representation.
- It allows to carry out an experimental study using crossed validation and holdout validation methods generating output reports with a personalized set of measures.
- Its use is simple by means of the configuration of *xml* files.

The rest of the document is organized as follows: Chapter 2 reviews the literature and current status of ML, MI, and MIML learning; Chapter 3 details the steps required to download and use the library; Chapter 4 shows the description of the library, considering the data format, its functionality, its architecture and its main packages and elements, as well as examples to configure the algorithms included in the library and a guide to develop a classifier step-by-step using the available features.

CHAPTER

2 Preliminary

This section carries out a background of relevant concepts in MIML environment. First, a brief definition of the most important concepts of MI and ML learning are addressed. Then, MIML learning is introduced. Finally, information relevant about metrics about dataset and evaluation are defined.

2.1 Multi-label learning

In traditional supervised learning (i.e. single-label learning), a pattern corresponds to a single instance consisting of a feature vector and an associated class label. Formally, let $\mathcal{X} = X_1 \times \ldots \times X_d$ be a d-dimensional input space and $\mathcal{Y} = \{\lambda_1, \lambda_2, ..., \lambda_q\}$ a set of q class labels. A pattern is a tuple $(\mathbf{x}, y)$ where $\mathbf{x} = (x_1, \ldots, x_d) \in \mathcal{X}$ and $y \in \mathcal{Y}$. Given $D = \{(\mathbf{x}_i, y_i) | 1 \leq i \leq m\}$ a dataset of m patterns, a multi-class classifier can be seen as a function $h_{\text{MC}} : \mathcal{X} \rightarrow \mathcal{Y}$. Note that a binary classifier is a particular case where $h_{\text{B}} : \mathcal{X} \rightarrow \{0, 1\}$.

Unlike traditional learning, ML learning is characterized by allowing an object (pattern) having more than one class (*label*), not being satisfied the restriction of *only-one-label-per-pattern* of traditional learning (a.k.a. single-label). In order to represent this fact, labels are binary variables that denote the belonging to each of the classes similarly to multi-class learning, but with the difference that a pattern may have more than one binary value activated [17]. Figure 2.1 and Figure 2.2 show the difference between traditional and multi-label learning. In the case of multi-label, the image can have simultaneously associated a set of classes or labels (e.g. *bridge*, *forest* and *river*), while in traditional single-label learning this is not allowed. In general terms, ML learning has undergone major developments in domains such as text and multimedia classification [18] [19], prediction of functions of genes and proteins [20], social networks data mining [21], or direct marketing [22].

A ML dataset can be defined as $D = \{(\mathbf{x}_i, Y_i) | 1 \leq i \leq m\}$, where $\mathbf{x_i} \in \mathcal{X}$ and $Y_i \subseteq \mathcal{Y}$ is a set of labels so-called *labelset*. Label associations can be also represented as a q-dimensional binary vector $\mathbf{y} = (y_1, y_2, \ldots, y_q) = \{0, 1\}^q$ where each element is 1 if the label is relevant and 0 otherwise.

According to [23], in ML learning two main tasks can be differentiated: Multi-Label Classification (MLC) and Label Ranking (LR). On the one hand, MLC consists of defining a function $h_{\text{MLC}} : \mathcal{X} \rightarrow 2^{\mathcal{Y}}$. Therefore, given an input instance, a multi-label classifier will return a set of

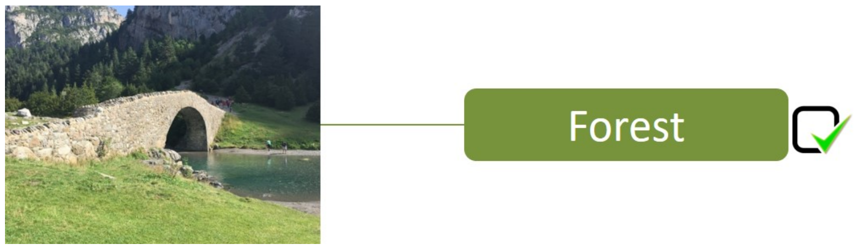


Figure 2.1: Single-label (SL) learning

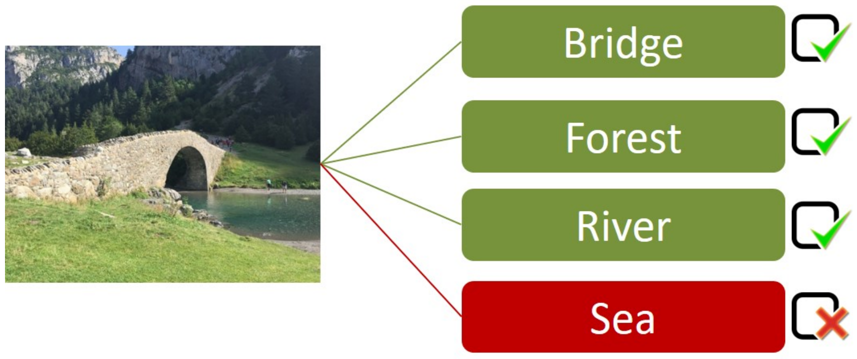


Figure 2.2: Multi-label (ML) learning

relevant labels, Y, being the complement of this set, $\overline{Y}$, the set of irrelevant labels. So, a bipartition of the set of labels into relevant and irrelevant labels is obtained.

On the other hand, Label Ranking (LR) defines a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbf{R}$ that returns an ordering of all the possible labels according to their relevance to a given instance $\mathbf{x}$. Thus label λ_1 is considered to be ranked higher than λ_2 if $f(\mathbf{x}, \lambda_1) > f(\mathbf{x}, \lambda_2)$. A rank function, $\tau_{\mathbf{x}}$, maps the output real value of the classifier to the position of the label in the ranking, $\{1, 2, ..., q\}$. Therefore, if $f(\mathbf{x}, \lambda_1) > f(\mathbf{x}, \lambda_2)$ then $\tau_{\mathbf{x}}(\lambda_1) < \tau_{\mathbf{x}}(\lambda_2)$. The lower the position, the better the position in the ranking is.

Finally, a third task, called Multi-Label Ranking (MLR), that can be seen as a generalization of MLC and LR can be defined. It produces at the same time both a bipartition and a consistent ranking. In other words, if Y is the set of labels associated with an instance, $\mathbf{x}$, and $\lambda_1 \in Y$ and $\lambda_2 \in \overline{Y}$ then a consistent ranking will rank labels in Y higher than labels in $\overline{Y}$, $\tau_{\mathbf{x}}(\lambda_1) < \tau_{\mathbf{x}}(\lambda_2)$. The definition of multi-label classifier from a multi-label ranking model can be derived from the function $f(\mathbf{x}, \lambda) : h(\mathbf{x}) = \{\lambda | f(\mathbf{x}, \lambda) > t(\mathbf{x}), \lambda \in \mathcal{Y}\}$, where $t(\mathbf{x})$ is a threshold function.

MLC algorithms can be categorized into *transformation algorithms* and *adaptation algorithms*. Algorithms in the former group transform a multi-label dataset into one or several (depending on the transformation used) datasets and then a well known single-label algorithm is applied. Some transformation methods, as Binary Relevance (BR), consider labels are independent. Other alternatives, as Label Powerset (LP), consider all label combinations, which involves a high computational complexity. More recent proposals have been focused on consider label relationships but with a reasonable computational cost [24]. The second group is composed by algorithms that adapt traditional algorithms to directly cope with ML data. Almost all classification paradigms have been adapted to the ML framework. It is worth highlighting some instance-based algorithms such as MLkNN [25] or IBLR [26]. Finally, other authors consider a third category of methods so-called multi-label ensembles in which base classifier are also multi-label classifiers [24] [17]. Many of these methods have yield high predictive performance. We can cite Random k-lAbELset (RAkEL), which builds a ensemble of LP classifiers by means of random label projections [27]. Ensemble of Pruned

Sets (EPS) [28] builds an ensemble of LP classifiers by applying a previous pruning of the less frequent labels. Finally, Ensemble of Classifier Chains (ECC) generates binary classifiers but chained in such a way that each classifier in the chain includes as inputs labels predicted by the previous classifiers in the chain [24].

The more challenging issues with ML learning are related with the need of deal with label relationships, the presence of imbalanced data and the high dimensionality of data both in the input (features, instances) and in the output space (labels). The latter is considered the main challenge of ML learning [4]. As noted ML framework is a field with significant progress mainly focused on the development of more scalable and precise models.

2.2 Multi-instance learning

MI is a learning paradigm proposed by Dietterich in 1997 with the aim of solving a problem of modelling the relationship between structure and the activity of drugs [1]. In this framework, each pattern, called *bag*, contains a variable number of instances. Each instance has the same number of attributes [29]. This representation allows to represent a pattern by means of several observations, usually corresponding to several perspectives or configurations of the same object. The great flexibility of this representation has promotes its use in applications such as document classification [30], web-index recommendation [31], scene classification [32] and image recovery [33]. Figure 2.3 shows an example of multiple-instance representation of an image. Each image is a bag represented by a sets of regions (instances) and with a class label associated.

In MI learning the aim is learning a function $h_{\text{MI}} : 2^{\mathcal{X}} \to \mathcal{Y}$ from a dataset $D\left\{(X_i, y_i)|1 \leq i \leq m\right\}$ where $X_i \subseteq \mathcal{X}$ is a set of instances $\left\{\mathbf{x}_{i1}, \mathbf{x}_{i2}, \ldots, \mathbf{x}_{in_i}\right\}, \mathbf{x}_{ij} \in \mathcal{X}, (j = 1, 2, \ldots, n_i)$, and $y_i \in \mathcal{Y}$ is the label of X_i. Each pattern i, a.k.a. bag, is a set of n_i instances.

There are many algorithmic proposals for MI learning. On the one hand, algorithms specially designed for MI, and, on the other hand there are algorithms which adapt the traditional learning hypothesis to the MI framework [34].

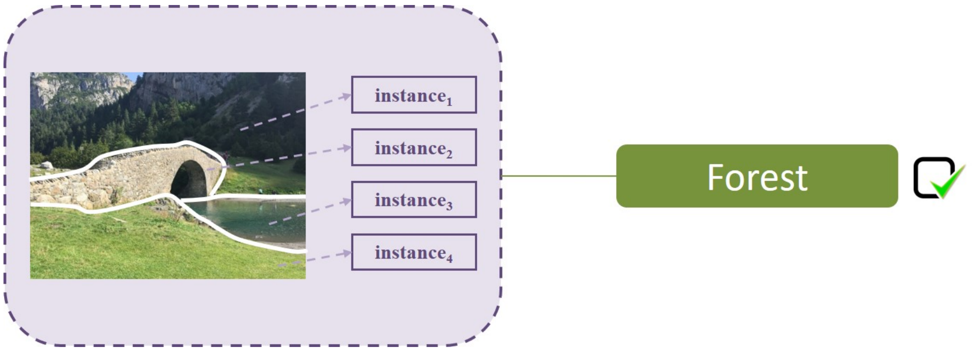


Figure 2.3: Multi-instance (MI) learning

APR [1] and Diverse Density (DD) [35] have been specially designed for MI. DD [36] is one of the most well-known algorithms. It is based on learning a concept whose feature space is close enough at least to an instance of each positive example and significantly far from all instances to negative objects. To this end, the concept of diverse density, a measure to determine the proximity or distance of instances in positive and negative objects to the estimated point. The key of the algorithm is selecting a point which maximizes diverse density by applying a standard bayesian classifier by considering bags with a set of instances instead single instances.

Multiple-Instance Logistic Regression (MILR) [37] adapts logistic regression to MI learning. For that, it assumes a logistic model of simple instances and uses the probability of their classes to calculate the class probabilities at bag level by using a noise model applied in DD. As labels at instance level are not known, MILR learns the parameters of this logistic regression model by maximizing the probability at bag level.

It is also worth citing the great amount of approaches based on Support Vector Machine (SVM) [38] [39] [40] whose results show great performance in many application domains. It can be noted MISMO, which replaces the kernel function of traditional learning by a multi-instance kernel (an instance-based similarity function). MISMO uses SMO algorithm [41] for SVM learning together with a multi-instance kernel [42].

The k Nearest Neighbors (kNN) approach was first used in a MI framework by Zucker [43]. The main difference between the different kNN based approaches is the metric used for distance between bags. The Hausdorff and Kullback–Leibler have been widely used. CitationKNN [43] is a kNN based approach in which distance between bags is measured with the minimal Hausdorff distance. In contrast with the traditional approach, that just considers nearest neighbours to classify an example, CitationKNN considers those examples in the train set in which the pattern to classify is the nearest in both references and citations. MIOptimalBall is based on the *optimal ball* method [44] and applies classification based on the distance to a reference point. Particularly, this method tries to find a sphere in the instance space where all instances of all negative bags are out of the sphere, and at least one positive instance of each bag is inside the sphere.

Finally, MIBoost [45] inspired in AdaBoost [45] is a boosting algorithm that builds a set of weak classifiers using a single-instance learner in which single instances receive the labels of their corresponding bag. Different hypothesis are considered to obtain the bag-level labels from the labels of single instances assigned by the classifiers (i.e. geometric mean, arithmetic mean and maximum and minimum values).

Methods adapting traditional learning algorithms to the MI framework have been also developed. For instance, MISimple computes a series of summary statistics to obtain a single instance from a whole bag. Depending on the option, it computes the geometric mean, the arithmetic mean or the minimum and the maximum values.

2.3 Multi-instance multi-label learning

ML and MI have rapidly evolved and, in recent years, some researchers have applied an hybrid approach to work simultaneously with complex data representation both in input and in output space [34]. In MIML paradigm, each pattern consists of a variable number of instances, having all instances the same number of attributes, and each pattern may have associated a set of class labels. Figure 2.4 represents an example of image for MIML framework. An image (bag) could be represented as a set of regions (instances) and have simultaneously associated several categories (labels).

Therefore, in MIML learning the aim is to learn a function $h_{\text{MIML}} : 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ from a dataset $D\{(X_i, Y_i) | 1 \leq i \leq m\}$ where $X_i \subseteq \mathcal{X}$ is a set of instances $\{\mathbf{x}_{i1}, \mathbf{x}_{i2}, \ldots, \mathbf{x}_{in_i}\}, \mathbf{x}_{ij} \in \mathcal{X}(j = 1, 2, \ldots, n_i)$, and $Y_i \subseteq \mathcal{Y}$ is a set of labels associated with X_i where $\mathcal{Y} = (y_1, y_2, \ldots, y_q) = \{0,1\}^q$.

Classification algorithms for MIML may be categorized into two approaches [46]. On the one hand, algorithms which transform previously the MIML problem. On the other hand, algorithms which address the MIML problem directly.

As MIML learing is based on both MI learning and ML learning, two types of transformations can be applied to solve a MIML by means of transformation problem [47]. In the first group, the problem is transformed to MI problem and then the resulting problem is solved by MI algorithms. The second transformation approach consists on transforming the problem to ML problem and

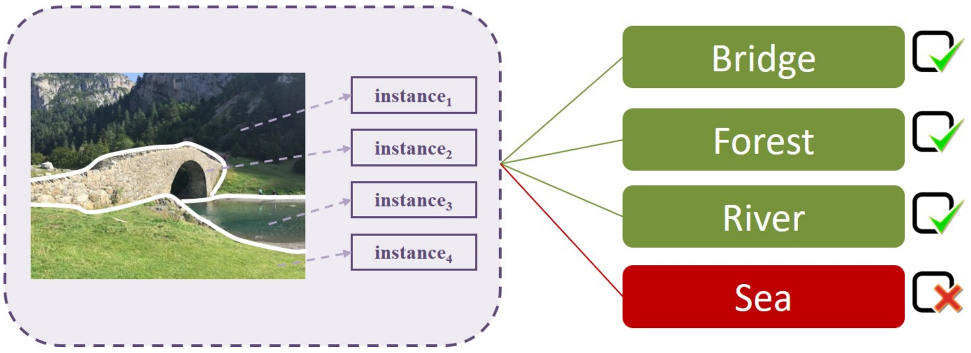
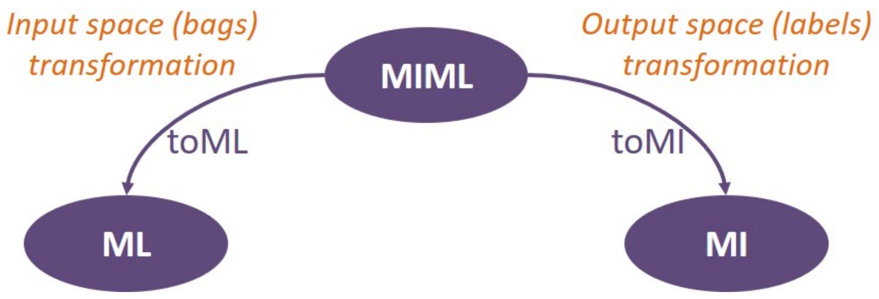


Figure 2.4: Multi-Instance multi-label (MIML) learning

then, the resulting problem is solved by ML algorithms. As it can be noted, the first approach is applied to the output space (labels) whereas the second one is applied to the input space (bags). Figure 2.5 shows both approaches.



Input space (bags) transformation
Output space (labels) transformation
MIML
toML
toMI
ML
MI

Figure 2.5: MIML transforming the problem

In the literature, algorithms that perform a transformation of the problem can be found. It can be highlighted ensemble methods [47], [10], SVMs [11] and neural network based methods [8]. The performance of these algorithms can be affected by the loose of information produced by the simplification/transformation. Connections among instances and labels as well as label correlations should be considered. Due to this reason, algorithms to deal with MIML directly also have been proposed. These proposals are mainly based on neural networks [48], ensembles [12], SVMs [9] and kNN [2]. In [29] an exhaustive description of proposals for MIML can be found.

2.4 Metrics about datasets

MIML learning combines MI learning and ML learning, two kind of metrics about datasets can be differentiated: metrics for MI data and metrics ML data. According to the notation given in previous sections, $D\{(X_i, Y_i)|1 \le i \le m\}$ represents a MIML dataset of m instances.

2.4.1 ML data metrics

The *label cardinality* (see Equation 2.1) and *label density* (see Equation 2.2) are two well-known metrics to measure how multi-labelled a dataset is. Cardinality is the average number of labels per pattern. Density is the cardinality divided by the total number of labels and it is used to compare datasets with different numbers of labels.

$$LCard(D) = \frac{1}{m}\sum_{i=1}^{m}|Y_i| \tag{2.1}$$

$$LDen(D) = \frac{LCard(D)}{q} \tag{2.2}$$

The *Distinct LabelSets* (see Equation 2.3) is described as the number of different label combinations in the dataset. *Diversity* (see Equation 2.4) is defined as the percentage of the bound of label sets (maximum number of labelsets that may exist in the dataset) that the distinct represents (that is actually in the dataset).

$$DL(D) = |Y \subseteq \mathcal{Y} | \exists (X, Y) \in D| \tag{2.3}$$

$$Diversity(D) = \frac{DL(D)}{2^q} \tag{2.4}$$

In [49] a complete description and taxonomy about metrics of ML datasets can be found.

2.4.2 MI data metrics

There are also some interesting metrics for Multi-Instance datasets, such as: number of attributes per bag, maximum, minimum and average number of instances per bag.

2.5 Evaluation metrics

When the performance of a MIML classifier is evaluated, a multi-label prediction could be completely right (all the labels are well predicted), partially right (just a set of the labels are well predicted), or completely wrong (any label is well predicted). Therefore specific evaluation metrics for ML learning that consider this fact must be used. ML performance evaluation metrics are usually categorized into two groups: *label-based metrics* and *example-based metrics.*

2.5.1 Label-based metrics

Any binary classification metrics can be computed with a label-based approach (e.g. precision, recall, sensibility, specificity, etc.). To this end, for each label, a contingency table with the number of *true positives* (tp), *true negatives* (tn), *false positives* (fp) and *false negatives* (fn) can be obtained (see Table 2.1).

Actual / Predicted	True	False
True	tp	fp
False	fn	tn

Table 2.1: Contingency table for a single label

Having a contingency table per label, values can be aggregated by following *macro* or *micro* [5] approach. It is supposed a dataset with q labels. The macro approach first computes a binary metric for each label, and then, averaged value is obtained (see equation 2.5). This approach considers the same weight for all labels being independent of their frequency so that it is recommended when the frequency of labels is not relevant for the classifier performance.

$$B_{macro} = \frac{1}{q} \sum_{i=1}^{q} B(tp_i, fp_i, tn_i, fn_i) \tag{2.5}$$

The micro approach first aggregates the values of all the contingency tables into a single table and then the value of the metric is computed (see equation 2.6). As it can be seen, this approach is more influenced by the most frequent labels.

$$B_{micro} = B(\sum_{i=1}^{q} tp_i, \sum_{i=1}^{q} fp_i, \sum_{i=1}^{q} tn_i, \sum_{i=1}^{q} fn_i) \tag{2.6}$$

Label-based metrics are easy to compute, but they ignore label relationships. A summary of the most used label-based metrics can be found in Table 2.2.

	Macro	Micro
Precision	$\frac{1}{q}\sum_{i=1}^{q}\frac{tp_i}{tp_i+fp_i}$	$\frac{\sum_{i=1}^{q} tp_i}{\sum_{i=1}^{q} tp_i + \sum_{i=1}^{q} fp_i}$
Recall (sensitivity, tp rate)	$\frac{1}{q}\sum_{i=1}^{q}\frac{tp_i}{tp_i+fn_i}$	$\frac{\sum_{i=1}^{q} tp_i}{\sum_{i=1}^{q} tp_i + \sum_{i=1}^{q} fn_i}$
Specificity (tn rate)	$\frac{1}{q}\sum_{i=1}^{q}\frac{tn_i}{tn_i+fp_i}$	$\frac{\sum_{i=1}^{q} tn_i}{\sum_{i=1}^{q} tn_i + \sum_{i=1}^{q} fp_i}$
Accuracy	$\frac{1}{q}\sum_{i=1}^{q}\frac{tp_i+tn_i}{tp_i+tn_i+fp_i+fn_i}$	$\frac{\sum_{i=1}^{q} tp_i + tn_i}{\sum_{i=1}^{q} tp_i + \sum_{i=1}^{q} tn_i + \sum_{i=1}^{q} fp_i + \sum_{i=1}^{q} fn_i}$
F-Measure	$2\frac{precision_{macro} \cdot recall_{macro}}{precision_{macro} + recall_{macro}}$	$2\frac{precision_{micro} \cdot recall_{micro}}{precision_{micro} + recall_{micro}}$

Table 2.2: Label-based metrics

2.5.2 **Example-based metrics**

Example-based metrics compute a metric value for each pattern, and then, an averaged value is obtained. These metrics can be categorized into metrics to evaluate bipartitions, rankings or confidences.

Let T be a MIML dataset with $\mid T \mid$ bags, each one with a set of associated labels, Y. A classifier predicts a set of labels Z for each bag. For any predicate, π, $I(\pi)$ returns 1 if the predicate is true and 0 in otherwise. Let Δ be the symmetric difference between the current, Y, and predicted sets of labels, Z (corresponding to the XOR operator of boolean logic). Let $\tau*$ be the current *ranking.*

- *Bipartitions*: these measures are based on evaluating differences between true (*ground truth*) and predicted label vectors. Table 2.3 shows the definition of these metrics that they are the next:
  - *Subset accuracy*: it computes the percentage of patterns in which predicted labels completely match the expected labels. It is a very strict metric as it requires an exact match.
  - *Hamming loss*: it considers both prediction errors (a wrong label is predicted) and omission errors (a label is not predicted). Its value is normalized by q and by the number of patterns in order to obtain a value in [0,1].
  - *Accuracy*: it is the proportion of label values correctly classified of the total number (predicted and actual) of labels.
  - *Precision*: it is the proportion of labels correctly classified of the predicted labels.
  - *Recall*: it is the proportion of predicted correct labels of the actual labels.
  - F-*Measure*: it combines precision and recall.

$$
\begin{aligned}
&0/1 Subset\ accuracy = \frac{1}{\mid T \mid} \sum_{i=1}^{|T|} I(Z_i = Y_i) \\
&Hamming \quad loss = \frac{1}{\mid T \mid} \sum_{i=1}^{|t|} \frac{\mid Y_i \Delta\ Z_i \mid}{q} \\
&Accuracy = \frac{1}{\mid T \mid} \sum_{i=1}^{|T|} \frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \cup Z_i \mid} \\
&Precision = \frac{1}{\mid T \mid} \sum_{i=1}^{|T|} \frac{\mid Y_i \cap Z_i \mid}{\mid Z_i \mid} \\
&Recall = \frac{1}{\mid T \mid} \sum_{i=1}^{|T|} \frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \mid} \\
&F - Measure = 2 \frac{precision \cdot recall}{precision + recall}
\end{aligned}
$$

Table 2.3: Example-based metrics - bipartitions

- *Rankings*: there are also a set of metrics to evaluate rankings of labels. If the classifer's output consists of a *ranking*, it is common to evaluate its performance with the metrics showed in Table 2.4 and describe below:

  - *One-error*: it evaluates how many times the label with best *ranking* was not in the set of possible labels, so the lower this value is, the better it is. The expression of this metric is shown in the equation of Table 2.4 where the function *arg* returns a label $\lambda \in \mathcal{Y}$.
  - *Coverage*: it measures the average depth in the *ranking* to cover all labels associated with an instance. The lower the value of this measure, the better the performance.
  - *IsError*: it measures whether the *ranking* predicted is perfect or not. Returns 0 if the *ranking* is perfect, and 1 otherwise, regardless of how bad the *ranking* is. This measurement has the same meaning as the *subset accuracy* described above, but applied to *rankings*.
  - *Ranking loss*: it evaluates, on average, the fraction of pairs of labels that are disordered in one instance. The lower this value, the better its performance.
  - *Average precision*: *coverage* and *one-error* are not complete metrics for multi-label classification, since you can have good values for *coverage* and a high value for *one-error*. Therefore, this metric is used, which evaluates the average fraction of labels classified above a specific label, $\lambda \in \mathcal{Y}$. Efficiency is perfect when the value of this metric is 1, the higher the value, the better.

$$
\begin{aligned}
&one - error = \frac{1}{t} \sum_{i=1}^{t} [\![arg \min_{\lambda \in Y} \ \tau_i(\lambda) \notin Y_i]\!] \\
&coverage = \frac{1}{t} \sum_{i=1}^{t} \max_{\lambda \in Y_i} \ \tau_i(\lambda) - 1 \\
&ranking\ loss = \frac{1}{t} \sum_{i=1}^{t} \frac{1}{|Y_i| \left|\overline{Y_i}\right|} |E| \ where \\
&\mathrm{E} = \{\ (\lambda, \lambda')|\tau_i(\lambda) > \tau_i(\lambda'), (\lambda, \lambda') \in Y_i \times \overline{Y_i}\} \\
&is\ error = \frac{1}{t} \sum_{i=1}^{t} [\![\sum_{\lambda \in \mathcal{L}} |\tau_i^*(\lambda) - \tau_i(\lambda)| \neq 0]\!] \\
&avg.\ precision = \frac{1}{t} \sum_{i=1}^{t} \frac{1}{|Y_i|} \sum_{\lambda \in Y_i} \frac{|\{\lambda' \in Y_i | \tau_i(\lambda') \leq \tau_i(\lambda)\}|}{\tau_i(\lambda)}
\end{aligned}
$$

Table 2.4: Example-based metrics - rankings

- *Confidences*: Finally, metrics to evaluate confidences can be defined.
  - *Logarithmic Loss*: it punishes larger errors more when the output of a multi-label classifier is a vector of confidence values for each label (Table 2.5). The error is graded based on the confidence with which it is predicted: predicting false positives with low confidence induces a lower logarithmic error than doing it with high confidence.

$LogarithmicLoss = \frac{1}{tq}\sum_{i=1}^{t}\sum_{\lambda\in\gamma} min(-\text{LogLoss}(\lambda, \mathbf{w_i}), \ln(t))$
where $\text{LogLoss}(\lambda, \mathbf{w_i}) = \ln(w_\lambda) if \lambda \in \overline{Y}$

Table 2.5: Example-based metrics - confidences

CHAPTER

3

Getting and running the library

This section describes all the necessary steps to download, install and configure everything you need to use the library and start developing your own code. There are three different ways to work with the library: through the project developed using Maven tool, through the Java project from an IDE or directly from a *jar* file through a terminal.

Before downloading the library, it is necessary to have Java Development Kit version 8 or higher installed. In order to download and install this JDK, you must to go to `https://www.oracle.com/technetwork/java/javase/downloads/index.html`, where you have access to the last supported versions and download the version which corresponds to your operative system and install it.

From here, it is specified the different steps according to preference to run the library: using maven tool, java project or from *jar* file.

3.1 Using Maven tool

1 To work with the library in this way, it is necessary to download Maven tool from `https://maven.apache.org/download.cgi`.

2 The library is released via GitHub, available at `https://github.com/i32bepea/MIML_Maven/releases/tag/v1.0`. Here you will find both the source code and the final *jar* file to execute experiments directly with the algorithms and functionalities of the library, the source code also has configuration files and datasets that you can use in the experimentation. Verify that the following directories exist within the main directory:

- *src*: it contains the library source code.
- *data*: it contains the data sets to be used during experimentation.
- *results*: it contains output files generated by example configurations included in the library.
- *configurations*: it contains configuration files used as examples by this manual.
- *apidoc*: it contains all the necessary API documentation for the use of MIML library.

- *dist*: it contains a *rar* file with the library distribution (its dependencies, configuration and data files, and the documentation generated) and the *jar* files.

3 To work with the source code, it is recommended to do it from an IDE such as Eclipse. In this way, it is easier to work with Maven, although you can also do everything from the command console. To work with the library through an IDE such as Eclipse go to step 4, to work with a terminal go to step 5 and 6.

4 Once the source code is downloaded, it is possible to import the project directly as a Maven Project from Eclipse. The project itself doesn't have all the dependencies it needs, in order to install them it is necessary to make use of Maven's functionalities. For this, it is necessary to create in the IDE a new *Run Configuration*, more specifically a *Maven Build*. Here, it will be necessary to configure two essential fields: *Base directory*, where the path of the imported workspace will be indicated; and the Maven Build's *goals*, here it is necessary to indicate "**clean install**" without quotes. Once the changes have been applied, the configuration can be run and, if there has not been any error, all the dependencies must have been correctly downloaded and installed and the message "BUILD SUCCESS" must appear in the IDE's console. When this is done, you can start to run experiments creating configurations as Java Applications; the main class of the library is *miml.run.RunAlgorithm* and it is necessary to specify the configuration file route used in the experiment though the option *-c*, for example:

```
-c configurations/MIMLclassifier/MIMLkNN.config
```

5 If an IDE isn't available or the one that is being used doesn't support Maven, the whole process detailed above could be done through a terminal. The only necessary step would be to get to the project directory and execute the command:

```
$ mvn clean install
```

6 Whether the Maven tool has been installed correctly there should be no problem and all dependencies have had to be downloaded. In addition, when Maven is run with the "**install**" goal, a folder called *target* will also be created in the workspace, which will contain, among other things, two *jar* files: *miml-1.0.jar* and *miml-1.0-jar-with-dependencies.jar*. The first *jar* file created doesn't have the dependencies of the library inside and serves to use it, for example, as a dependency of other Maven projects; the second *jar* file contains all the dependencies and can be used as an executable of the library. Below, it is shown an example of how running the library from the terminal assuming it is located at the root of the project:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -
    c configurations/MIMLclassifier/MIMLkNN.config
```

3.2 Using Java project

1 The library is released via GitHub, available at `https://github.com/i32bepea/MIML/releases/tag/v1.0`. Here you will find both the source code and the final *jar* file to execute experiments directly with the algorithms and functionalities of the library, the source code also has configuration files and datasets that you can use in the experimentation. Verify that he following directories exist within the main directory:

- *src*: it contains the library source code.
- *data*: it contains the data sets to be used during experimentation.
- *results*: it contains output files generated by example configurations included in the library.
- *configurations*: it contains configuration files used as examples by this manual.

- *apidoc*: it contains all the necessary API documentation for the use of MIML library.
- *lib*: it contains the dependencies used in the library.
- *lib-src*: it contains source *jar* of the dependencies used in the library.
- *dist*: it contains a *rar* file with the library distribution (its dependencies, configuration and data files, and the documentation generated) and the library *jar* and source *jar*.

2 Once the source code is downloaded, it is possible to import the project directly as a Java Project from an IDE such as Eclipse. The project itself have all the dependencies it needs so it is not necessary download any additional library. When this is done, you can start to run experiments creating configurations as Java Applications; the main class of the library is *miml.run.RunAlgorithm* and it is necessary to specify the configuration file path used in the experiment though the option *-c*, for example:

```
-c configurations/MIMLclassifier/MIMLkNN.config
```

3.3 Using *jar* file

1 Both projects (Java project and Maven project described in the previous sections) contain in the *dist* folder a *jar* file to run the library through a terminal. This *jar* file is also available in both projects GitHub releases.

2 This *jar* file contains all the dependencies and can be used as an executable of the library. Below, it is shown an example of how running the library from the terminal:

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/
    MIMLclassifier/MIMLkNN.config
```

CHAPTER
4 MIML library

MIML library presents a framework to work with MIML learning based on two well-known libraries. On one hand, Weka library that is able to deal with the MI representation and on the other hand, MULAN library [14] that is able to deal with ML representation. Thus, researchers get used to work with these libraries can rapidly become familiar with the structure and data format used in MIML library. These are the main remarkable features:

1. It uses data format which has been specifically designed for MIML learning. However, it is based on data format used in Weka and MULAN to make easier and intuitive the use of MIML library.

2. It includes a set of MIML algorithms. Concretely, 32 algorithms are included in the library. As it is based on Weka and MULAN, it allows a wide set of their MI and ML algorithms to be used in a MIML context using the appropriate transformation method.

3. Algorithms can be easily used and executed by means of *xml* configuration files.

4. Experimental study using holdout and cross validation methods can be developed.

5. The framework includes also a wide set of performance evaluation metrics for MIML learning.

6. The structure of the library provides an easy way to develop and test new algorithms to solve MIML classification problems.

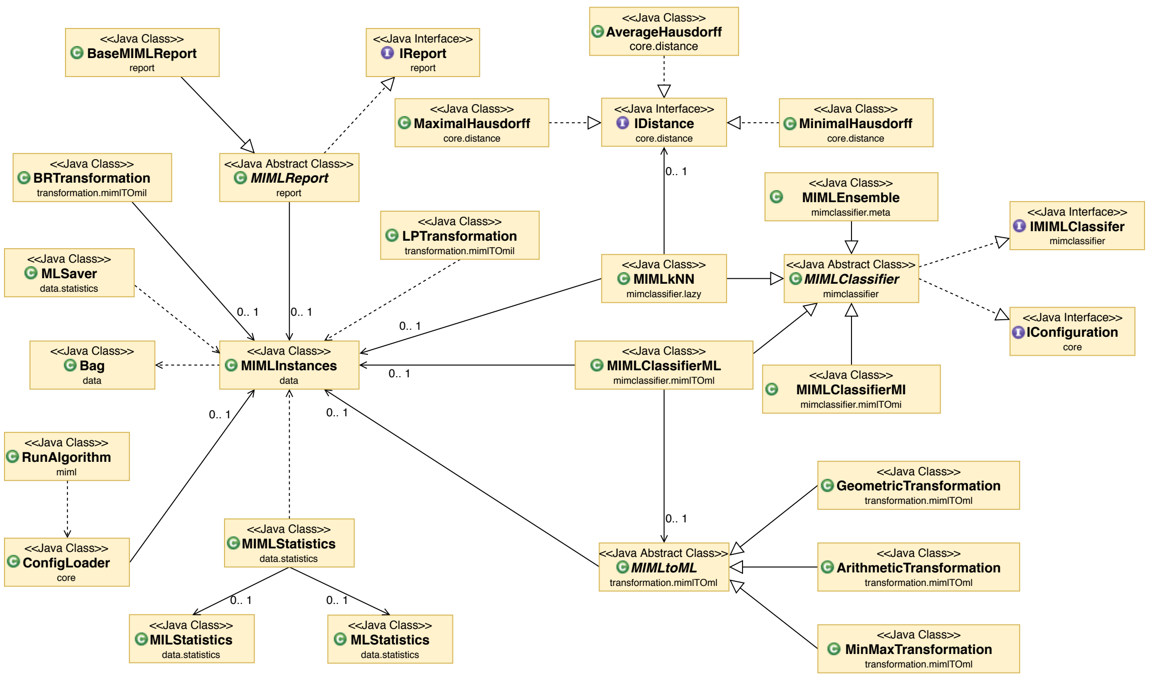
In this section, the library architecture, the data format and the main functionalities are explained. Moreover, it is specified the configuration of an experiment and the development of new algorithms thanks to the features that the library provides.

4.1 MIML library architecture

The library has been developed in open source Java. It is based on MULAN and Weka libraries and it is organized in packages. All the packages contain the interfaces and the classes required to extend the functionality. Therefore, it is easy to develop new transformation methods or classification algorithms. Following, it is specified the main functionality of each package:

- *core.* It contains classes related with the execution of algorithms by means of *xml* configuration files. *Iconfiguration* interface must be implemented by any algorithm to be configured by *xml* files. The *ConfigLoader* class allows to read the *xml* file and to configure an experiment.
- *core.distance.* It includes several variants of the *Hausdorff* distance to compute distance between bags.
- *data.* It includes classes to deal with the data format described in Section 4.2. Therefore, classes in this package allow to load a MIML dataset and know properties about data such as the number of attributes, the number of bags, the number of labels, etc. Besides, it is possible to access to a bag as well as to its instances and labels.
- *data.statistics.* It contains classes to provide descriptive information about a MIML dataset. It considers both MI information (e.g. number of attributes per bag, maximum, minimum and average number of instances per bag etc.) and ML descriptive information (e.g. cardinality, density, frequency of labelsets, label co-occurrences, etc.)
- *transformation.mimlTOml.* It includes methods to transform a MIML dataset into a ML one.
- *transformation.mimlTOmi.* It includes methods to transform a MIML dataset into a MI one.
- *classifiers.miml.* It includes interfaces and abstract classes required to develop MIML classification algorithms.
- *classifiers.miml.mimlTOmi.* It includes classification algorithms that solve the MIML problem by transforming it to a MI problem. Currently, the library contains 15 algorithms.
- *classifiers.miml.mimlTOml.* It contains classification algorithms that solve a MIML problem by transforming it into a ML problem. Currently, the library contains 15 algorithms.
- *classifiers.miml.lazy.* It includes different kinds of lazy algorithms. Currently it includes the MIMLkNN algorithm [2].
- *classifiers.miml.meta.* It contains the bagging [50] algorithm scheme.
- *report.* It contains classes to generate result reports about the experiments carried out. The library has a general report where the main evaluation metrics are considered. More specific output reports can be extended using these classes.
- *tutorial.* It includes a set of usage examples: running a MIML classification algorithm, transforming a MIML dataset to MI and to ML, etc.
- *run.* It contains the class to execute any classifier of the library configured by means of a *xml* file.

Figure 4.1 shows a class diagram with the main classes of the library and their relationships.



<<Java Class>> BaseMIMLReport report
<<Java Interface>> IReport report
<<Java Class>> AverageHausdorff core.distance
<<Java Class>> MaximalHausdorff core.distance
<<Java Interface>> IDistance core.distance
<<Java Class>> MinimalHausdorff core.distance
<<Java Class>> BRTransformation transformation.mimlTOmil
<<Java Abstract Class>> MIMLReport report
<<Java Class>> MIMLEnsemble mimclassifier.meta
<<Java Interface>> IMIMLClassifer mimclassifier
<<Java Class>> LPTransformation transformation.mimlTOmil
<<Java Class>> MLSaver data.statistics
<<Java Class>> MIMLkNN mimclassifier.lazy
<<Java Abstract Class>> MIMLClassifier mimclassifier
<<Java Interface>> IConfiguration core
<<Java Class>> Bag data
<<Java Class>> MIMLInstances data
<<Java Class>> MIMLClassifierML mimclassifier.mimlTOml
<<Java Class>> MIMLClassifierMI mimclassifier.mimlTOmi
<<Java Class>> RunAlgorithm miml
<<Java Class>> GeometricTransformation transformation.mimlTOml
<<Java Class>> ConfigLoader core
<<Java Class>> MIMLStatistics data.statistics
<<Java Abstract Class>> MIMLtoML transformation.mimlTOml
<<Java Class>> ArithmeticTransformation transformation.mimlTOml
<<Java Class>> MILStatistics data.statistics
<<Java Class>> MLStatistics data.statistics
<<Java Class>> MinMaxTransformation transformation.mimlTOml
0.. 1

Figure 4.1: Class diagram

4.2 Managing MIML data

4.2.1 MIML data set format

The format of data is based on the Weka's format for MI learning and on the Mulan's format for ML learning. Concretely, each data set is represented by two files:

- An *xml* file based on Mulan's format containing the description of labels. Its aim is to identify those attributes in the *arff* file representing labels. Note that the class attributes do not need to be the last attributes in the *arff* file and also their order in both at the *arff* and the *xml* file does not matter. A hierarchy of labels can be represented by nesting the label tags. The following is an example of *xml* file with 4 labels:

```xml
<?xml version="1.0" encoding="utf-8"?>
<labels xmlns="http://mulan.sourceforge.net/labels">
  <label name="label1"></label>
  <label name="label2"></label>
  <label name="label3"></label>
  <label name="label4"></label>
</labels>
```

  The following is an example of *xml* file with a hierarchy of labels:

```xml
<?xml version="1.0" encoding="utf-8"?>
<labels xmlns="http://mulan.sourceforge.net/labels">
    <label name="sports">
        <label name="football"></label>
        <label name="basketball"></label>
    </label>
    <label name="arts">
        <label name="sculpture"></label>
        <label name="photography"></label>
    </label>
</labels>
```

- An *arff* (*Attribute-Relation File Format*) file based on Weka's multi-instance format containing the data. Comment lines begin with %. This file is organized in two parts: header and data.
  - *Header:* it contains the name of the relation and a list with the attributes and their data types.
    - The first line of the file contains the *@relation <relation-name>* sentence, which defines the name of the dataset. This is a string and it must be quoted if the relation-name includes spaces.
    - Next, on the first level, there are only two attributes and the attributes corresponding to the labels.
      - *<bag-id>*. Nominal attribute. Unique bag identifier for each bag.
      - *<bag>*. Relational attribute. Contains instances attributes.
      - *<labels>*. One binary attribute for each label (nominal with 0 or 1 value).

    Attributes are defined with *@attribute <attribute-name><data-type>* sentences. There is a line per attribute.
    - Numeric attributes are specified by *numeric*.
    - In case of nominal attributes, the list of values must be specified with curly brackets and separated by commas: *{value$_1$, value$_2$, ..., value$_N$}*.
  - *Data:* it begins with *@data* and describes each example (*bag*) in a line. The order of attributes in each line must be the same in which they were defined in the previous header. Each attribute value is separated by comma (,) and all lines must have the same number of attributes. Decimal position is marked with a dot (.). The data of the relational attribute is surrounded by single (') or double (") quotes, Weka recognizes both formats, and the single instances inside the bag are separated by line-feeds (*\n*).

    Next, an example of *arff* file is showed. In the example, each bag contains instances described by 3 numeric attributes and there are 4 labels. The dataset has two bags, the first one with 3 instances and the second one with 2 instances.

```
@relation toy
@attribute id {bag1,bag2}
@attribute bag relational
  @attribute f1 numeric
  @attribute f2 numeric
  @attribute f3 numeric
@end bag
@attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}
@data
bag1,"42,-198,-109\n42.9,-191,-142\n3,4,6",1,0,0,1
bag2,"12,-98,10\n42.5,-19,-12",0,1,1,0
```

The distribution of the library includes the *birds* dataset [51]. It is a dataset to predict the set of birds species that are present, given a ten-second audio clip. The full dataset consisted of 645 ten-second audio recordings in uncompressed WAV format (16kHz sampling frequency, 16 bits per sample, mono). Being a competition, just 282 patterns were available (1/3 of the original)[1]. This dataset has been formatted to MIML format specified in this section and Table 4.1 contains a summary of the main features of this dataset (they are described in section 2.4).

[1]More information can be found in https://www.kaggle.com/c/mlsp-2013-birds

dataset	domain	bags	avg. inst/bags	min inst/bag	max inst/bag	attr	labels	card	dens	dist
Birds	audio	282	7.400	1	36	38	19	2.010	0.105	125

Table 4.1: Features of the *birds* dataset

4.2.2 **Obtaining information of MIML data set**

The library offers in the *data.statistics* package a series of metrics for data exploration and analysis of MIML datasets that could be taken into account to develop and study new proposals (*MIML-Statistics* class) - See section 2.4. These metrics include dimensionality metrics (number of bags, attributes, labels, etc.). Moreover, it allows to perform an analysis of imbalance and relationships among labels.

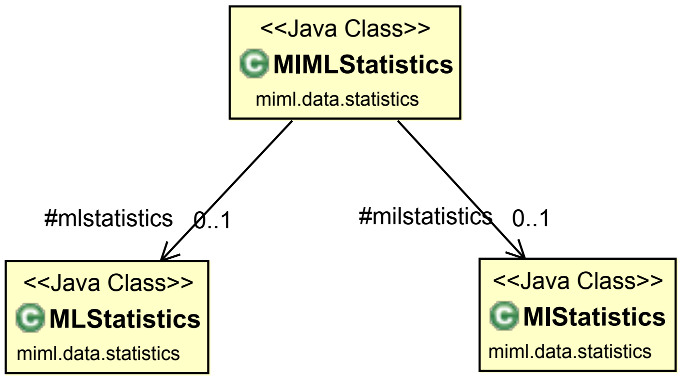


Figure 4.2: Statistics class diagram

The Figure 4.2 shows the library classes that provide the functionality for obtaining data descriptions from MIML, MI and ML datasets.

Next, it is detailed what attributes and methods make up these classes. It is important to note that, in addition to the methods explained, all classes have all the necessary getters and setters to obtain the desired information, as well as various methods that allow obtaining information both in plain text and in csv format.

4.2.2.1 **Information for multi-label data**

The MLStatistics class belongs to the *miml.data.statistics* package, and will be responsible for obtaining information about a ML data. It is based on the *mulan.data.Statistics.java* class and it has been included methods to evaluate the imbalance of the labels and a bug in the printPhiDiagram method has been corrected.

- Attributes:
  - *numLabels*: Number of labels.
  - *numExamples*: Number of examples.
  - *numAttributes*: Number of attributes.
  - *numNominal*: Number of nominal predictive attributes.
  - *numNumeric*: Number of numeric predictive attributes.
  - *positiveExamplesPerLabel*: number of positive examples per label.
  - *distributionLabelsPerExample*: distribution of examples having 0, 1, 2,..., n labels.

  - *peak*: number of occurrences of the highest frequent label combination.
  - *base*: number of occurrences of the lowest frequent label combination.
  - *nUnique*: number of label sets with only one pattern.
  - *maxCount*: number of label sets with the peak value.
  - *coocurrenceMatrix*: matrix with the coocurrence of pairs of labels.
  - *phi*: matrix with Phi correlation among pairs of labels.
  - *chi2*: matrix with Chi-square correlation among pairs of labels.

- Methods:
  - *calculateStats()*: it calculates various Multi-Label statistics, the most of the remaining methods require call this one previously.
  - *cardinality()*: it computes the cardinality as the average number of labels per pattern.
  - *density()*: it computes the density as the cardinality/number of labels.
  - *priors()*: it returns the prior probabilities of the labels.
  - *calculateCoocurrence()*: it calculates a matrix with the coocurrences of pairs of labels.
  - *calculatePhiChi2()*: it calculates Phi and Chi-square correlation matrix.
  - *getPhiHistogram()*: it calculates a histogram of Phi correlations.
  - *uncorrelatedLabels()*: it returns the indices of the labels whose Phi coefficient values lie between -*bound* <= phi >= *bound*, *bound* value is given as a parameter.
  - *topPhiCorrelatedLabels()*: it returns the indices of the labels that have the strongest Phi correlation with the label which is given as a parameter.
  - *innerClassIR()*: it calculates the degree of imbalance for each of the labels binary as the number of negative patterns divided by the number of positive patterns for each binary label.
  - *interClassIR()*: it calculates the degree of imbalance of each binary label with respect to the majority binary label as the number of positive patterns of the majority label divided by the number of positive patterns of each label.
  - *averageIR()*: it computes the average value of a vector with the degree of imbalance for each binary label.
  - *varianceIR()*: it computes the variance value of a vector with the degree of imbalance for each binary label.
  - *pUnique()*: it returns proportion of unique label combinations value defined as the proportion of label sets which are unique across the total number of examples.
  - *pMax()*: it returns the proportion of associated examples with the most frequently occurring label set.
  - *labelSkew()*: it calculates the degree of imbalance of each combination of labels as the number of patterns of the most frequent label set divided by the number of patterns of the label set in question.
  - *averageSkew()*: it computes the average labelSkew.
  - *skewRatio()*: it computes the skewRatio as peak/base.

4.2.2.2 **Information for multi-instance data**

This class is located in *miml.data.statistics* and allows to obtain information about MI data such as number of attributes per bag, average number of instances per bag, distribution of number of instances per bag, etc.

- Attributes:
  - *attributesPerBag*: number of attributes per bag.
  - *avgInstancesPerBag*: average number of instances per bag.
  - *distributionBags*: distribution of number of instances per bag.
  - *maxInstancesPerBag*: maximum number of instances per bag.
  - *minInstancesPerBag*: minimum number of instances per bag.
  - *numBags*: number of bags.
  - *totalInstances*: total number of instances.
- Methods:
  - *calculateStats()*: it calculates all multi-instance statistics defined previously.

4.2.2.3 Information for multi-instance multi-label data

This class is contained in *miml.data.statistics* package too. It has methods for obtaining MIML dataset statistics. This class allows to perform with MIML data and obtain statics both multi-instance and multi-label using the previous classes.

4.2.3 Transforming MIML data sets

The library contains methods to transform a MIML data set to a MI data set using the Weka library format or an ML data set in MULAN library format. These data sets can be used respectively by Weka's MI classification algorithms and MULAN's ML classification algorithms. It contains the classes *MIMLInstance* and *Bag* whose purpose is to represent the structure of a MIML dataset. It also contains the class *MLSave*, which allows to save in a file ML and MIML datasets.

The Figure 4.3 shows the library classes that provide the functionality for transforming datasets.

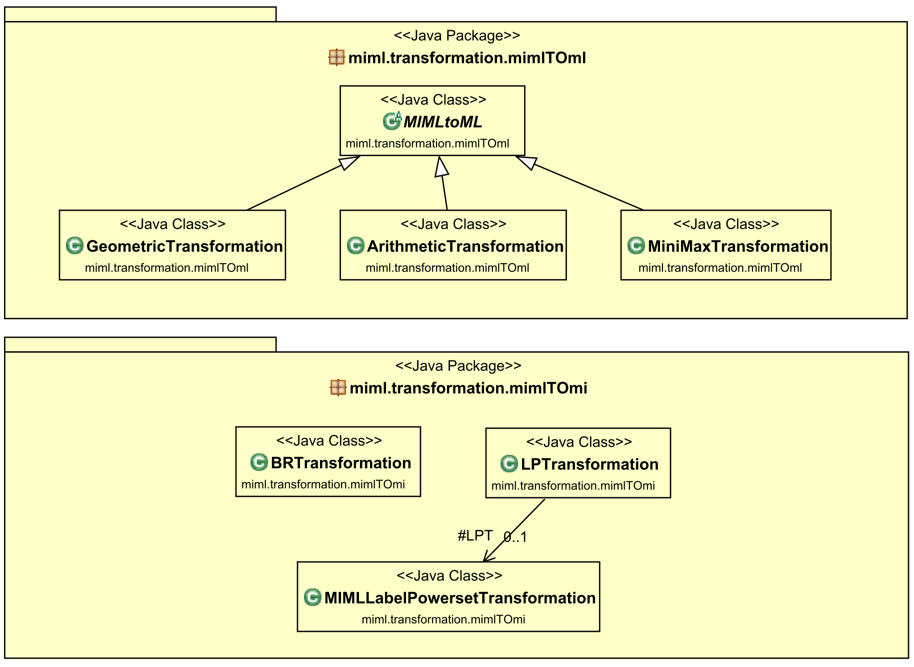

<<Java Package>>
miml.transformation.mimlTOml
<<Java Class>>
MIMLtoML
miml.transformation.mimlTOml
<<Java Class>>
GeometricTransformation
miml.transformation.mimlTOml
<<Java Class>>
ArithmeticTransformation
miml.transformation.mimlTOml
<<Java Class>>
MiniMaxTransformation
miml.transformation.mimlTOml
<<Java Package>>
miml.transformation.mimlTOmi
<<Java Class>>
BRTransformation
miml.transformation.mimlTOmi
<<Java Class>>
LPTransformation
miml.transformation.mimlTOmi
#LPT 0..1
<<Java Class>>
MIMLLabelPowersetTransformation
miml.transformation.mimlTOmi

Figure 4.3: Class diagram for transforming the problem

4.2.3.1 Methods to transform MIML data to MI data

The library includes two different methods to transform MIML dataset to MI dataset with the format used in Weka [5]:

- *Binary Relevance Transformation*: it transforms a MIML data set into as many binary MI data sets as labels the problem has.
- *Label Powerset Transformation*: it transforms a MIML dataset into a multiclass in which each possible combination of tags from the original dataset is considered a different class.

Table 4.2 shows the Weka MI algorithms than can be used for each transformation. Note that if MDD, MIDD, MIBoost, MILR, MIOptimalBall, MIRI, MISMO, MISVM or MITI are run with an LP transformation the following execution error is raised *Cannot handle multi-valued nominal class!*. This is due to the philosophy of the LP method which obtains one multi-class dataset and these algorithms are only able to deal with binary class data. Due to this fact, these methods have not been included in Table 4.2 for LP transformation.

Label transformation	**MI classifiers (Weka)**
BR [53]	CitationKNN [43]
	MDD [52]
	MIDD [52]
	MIBoost [45]
	MILR [37]
	MIOptimalBall [44]
	MIRI [54]
	MISMO [42]
	MISVM [55]
	MITI [56]
	MIWrapper[57]
	SimpleMI[58]
LP [53]	CitationKNN [43]
	MIWrapper [57]
	SimpleMI [58]

Table 4.2: Classifiers that can be used to solve the problem transformed to MI problem

4.2.3.2 Methods to transform MIML data to ML data

The library includes three different methods described in [58] to transform MIML dataset to ML dataset with the format used in Mulan:

- *Arithmetic Transformation*: transforms each bag into a single instance where the value for each attribute is its average value within the bag.
- *Geometric Transformation*: transforms each bag into a single instance where the value for each attribute is the geometric center of its maximum and minimum values within the bag.
- *Min-Max Transformation*: transforms each bag into a single instance that contains, for each attribute, its minimum and maximum values within that bag. Each instance is defined by twice as many attributes as it previously had.

Table 4.3 shows the Mulan ML algorithms that can be used for each transformation.

Bag transformation	ML classifiers (Mulan)
Arithmetic Geometric Min-Max [58]	BR [23]
	LP [23]
	RPC [59]
	CLR [60]
	BRkNN [61]
	DMLkNN [62]
	IBLR [26]
	MLkNN [25]
	HOMER [63]
	RAkEL [27]
	PS [28]
	EPS [28]
	CC [24]
	ECC [24]
	MLStacking [53]

Table 4.3: Classifiers that can be used with to solve the problem transformed to ML problem

4.3 Running a classification MIML algorithm included in the library

All algorithms included in the library are executed by means of the *RunAlgorithm* class (located in the package *miml.run*) and using a configuration file to specify the algorithm and parameters that are going to be used in the experiment. The specific format of configuration file is specified in section 4.3.1 and examples are shown in the following sections.

Concretely, **32** proposals can be executed in this library considering 15 MI classifiers when the problem is transformed to MI problem, 15 ML classifiers when the problem is transformed to ML problem and 2 specific algorithms for MIML learning. Moreover, many more combinations can be run considering all possible combinations between algorithms and transformation methods available in the library:

- *MIMLClassifierMI*: it includes algorithms that perform a transformation of the MIML problem to get an MI problem, and then it is solved the MI problem. The library considers two transformations widely used in the multi-label learning environment, transformation based on LP and transformation based on BR. Once the transformation has been performed, it gets a result by solving the problem with a specified MI algorithm. Being compatible with the Weka library, Table 4.2 shows an example of 12 Weka algorithms that could be used directly with BR and the 3 algorithms that could be used with LP. In the section 4.3.2 it is shown examples of the execution of each algorithm, showing their configuration file and the results obtained.

- *MIMLClassifierML*: it includes algorithms that perform a transformation of the MIML problem to get an ML problem, and then it is solved the ML problem. The library considers three transformations widely used in the multi-instance learning environment: arithmetic, geometric or min-max transformation. Once the transformation has been performed, it gets a result by solving the problem with a specified ML algorithm. As it is compatible with the MULAN library, Table 4.3 shows an example of 15 MULAN algorithms that could be used directly. The section 4.3.3 shows examples of the execution of each one of these algorithms, showing its configuration file and results obtained.

- *MIML-kNN* [2]: it is an algorithm that directly solves the problem working with the MIML data, without making any previous transformation of the problem. This algorithm uses the nearest cites and references to a bag to estimate the possible classes to which it belongs. In the section 4.3.4.1 an example of the execution of this algorithm is shown, explaining its configuration file and the results obtained.

- *MIML Bagging*: it is an adaptation of the traditional bagging strategy of the machine learning [50] which does not need any previous transformation of the problem. Consists of generating m different classifiers, each of which will work with a different dataset formed from the original, by means of a uniform sampling and with replacement (or not). In the section 4.3.4.2 an example of the execution of this algorithm is shown, explaining its configuration file and the results obtained.

It is necessary to specify the config file path though command line with the option *-c*. The class *RunAlgorithm* is responsible for making use of *ConfigLoader* to load the three different parts that compose a configuration file: classifier, evaluator and report. An example of execution it would be:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/MIMLkNN.config
```

4.3.1 Configuration file format

This section explains the format of configuration files and in the next sections, the specific configuration files that should be used to execute each algorithm considered in the library are shown.

These files use an *xml* format, with the next structure:

```
<configuration>
  <classifier>  </classifier>
  <evaluator>  </evaluator>
  <report> </report>
</configuration>
```

All files start at root element *configuration* and contain three branches: *<classifier>* element, *<evaluator>* element and *<report>* element. In the *<classifier>* element, it is specified the classification algorithm of the library to be used. The specific attribute of this element is *name* to describe the classification algorithm to use. Moreover, it contains several child elements that specify the parameters of the algorithm. In this manual, for each algorithm is given its specific parameters to correctly execute it. In the example, it is shown the MIMLkNN algorithm specification which needs three parameters *nReferences*, *nCiters* and *metric*,:

```
<classifier name="miml.classifiers.miml.lazy.MIMLkNN">
  <nReferences>4</nReferences>
  <nCiters>6</nCiters>
  <metric name="miml.core.distance.AverageHausdorff"></metric>
</classifier>
```

The next element that is a branch direct of root element is *<evaluator>*. This element describes the dataset used and different validation methods that can be used that is specified in *<data>* element and the *seed* used specified in *<seed>* element. With respect to validation methods, it is included both holdout and cross-validation, both located in the package *miml.evaluation*. However, it is possible to design an own evaluator implementing the interface IEvaluator. It is important to know that depending on the chosen method, the parameters that configure it can change. For the holdout evaluator, it is necessary to indicate: the path of train dataset file in *arff* format, the path of test dataset file in *arff* format and the path of *xml* file that contains the description of the labels, as it was seen in the section 4.2. An example configuration file for holdout would be:

```xml
<evaluator name="miml.evaluation.EvaluatorHoldout">
    <seed>72361</seed>
  <data>
    <trainFile>data/miml_birds_random_80train.arff</trainFile>
    <testFile>data/miml_birds_random_20test.arff</testFile>
    <xmlFile>data/miml_birds.xml</xmlFile>
  </data>
</evaluator>
```

For the cross-validation evaluator, it is necessary to specify three elements: number of folds, the path of dataset in *arff* format and the *xml* file path corresponding to the dataset.

```xml
<evaluator name="miml.evaluation.EvaluatorCV">
    <seed>712637</seed>
  <numFolds>5</numFolds>
  <data>
    <file>data/miml_birds.arff</file>
    <xmlFile>data/miml_birds.xml</xmlFile>
  </data>
</evaluator>
```

Another point to keep in mind is that all parameters related with the dataset used during the run of a experiment (*<file>*, *<trainFile>*, *<testFile>* and *<xmlFile>*) must be included in the element *<data></data>*.

Finally, the element *<report>* indicates the report specification that the output file generates. This class can be easily extended to obtain the most convenient output format. This element contains the attribute *name* to specify the report to use. Then, the element *<fileName>* is defined to specify the path where the result output file will be stored. Optionally, it can be defined the *<measure>* element describing the measures that will be shown in the output report.

In the example, it is specified the measures: *hamming loss*, *subset accuracy*, *macro-averaged precision*, *macro-averaged f-measure* and *geometric mean average interpolated precision*. If no measure is specified, all measures allowed by the specified classifier are considered. Table 4.4 shows the metrics included in the library.

```xml
<report name="miml.report.BaseMIMLReport">
  <fileName>results/mimlknnn.csv</fileName>
  <header>true</header>
  <standardDeviation>true</standardDeviation>
  <measures perLabel='true'>
    <measure>Hamming Loss</measure>
    <measure>Subset Accuracy</measure>
    <measure>Macro-averaged Precision</measure>
    <measure>Macro-averaged F-Measure</measure>
    <measure>Geometric Mean Average Interpolated Precision</measure>
  </measures>
</report>
```

In addition, it is possible to configure the report with three more elements: *<header>*, *<standardDeviation>* and the attribute *perLabel* on *<measures>* element. These characteristics indicate to the library if it has to include in the beginning of the document a header with the description of each value included in the report, if it has to add the standard deviation of the measures (just for crossed validation) and if it has to include the values of each measure for each of the labels that form the dataset used in the experiment respectively.

Label-based	Macro	Macro-averaged Precision Macro-averaged Recall Macro-averaged F-Measure Macro-averaged Specificity
	Micro	Micro-averaged Precision Micro-averaged Recall Micro-averaged F-Measure Micro-averaged Specificity
Example-based	Bipartitions	Hamming Loss Subset Accuracy Example-Based Precision Example-Based Recall Example-Based F Measure Example-Based Accuracy Example-Based Specificity
	Ranking	Average Precision Coverage OneError IsError ErrorSetSize Ranking Loss
	Confidences	Mean Average Precision Geometric Mean Average Precision Mean Average Interpolated Precision Geometric Mean Average Interpolated Precision Micro-averaged AUC Macro-averaged AUC Logarithmic Loss

Table 4.4: Evaluation measures included in MIML

The library contains a set of configuration files for each algorithm included. These files, located in *configurations* folder, can be used as template for create your own configurations. Generally, all configuration files keep the structure specified in this section. Nevertheless, in the following sections are given specific examples for each algorithm.

4.3.2 MIML algorithms transforming MIML problem to MI problem

This section shows a set of examples with the different algorithms that transform the MIML problem into an MI problem and then, it is used a MI algorithm to solve the problem.

Table 4.2 shows the Weka MI algorithms that can be used for each transformation. Note that if MDD, MIDD, MIBoost, MILR, MIOptimalBall, MIRI, MISMO, MISVM or MITI are run with an LP transformation the following execution error is raised *Cannot handle multi-valued nominal class!*. This is due to the philosophy of the LP method which obtains one multi-class dataset and these algorithms are only able to deal with binary class data. Due to this fact these methods have not been included in Table 4.2 for LP transformation.

In general, algorithms which transform the problem to MI need to specify in the configuration file: the transformation algorithm that transforms the MIML problem to MI one, and the MI classifier that you want to apply. Although it is possible to develop your own transformation method and the library has the necessary interfaces to facilitate its implementation, the library contains transformation methods (they are detailed in the section 4.2.3.1 and 4.2.3.2). In addition, Table 4.2 contains the MI classifiers from the Weka library that work correctly for this type of problem.

Both MI algorithm and transformation method must be specified in the configuration file in the <*classifier*> element using the <*multiInstanceClassifier*> and <*transformationMethodr*> elements. Here is an example:

```xml
<classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierMI">
  <multiInstanceClassifier name="xxxxxxxxxxx" />
  <transformationMethod name="xxxxxxxxxx"/>
</classifier>
```

Next, it is shown each available classification algorithm commenting on a brief description and the main configuration parameters. In addition, a complete configuration file is displayed (also available in the library) along with the necessary steps to execute it.

4.3.2.1 **CitationKNN classifier**

CitationKNN [43] is an adaptation of K-Nearest Neighbor to the MI problem. This classifier can be configured with both transformation methods available in the library and explained in section 4.3.1. In this example, BR method is used.

The classifier can be easily configurable using *<listOptions>* element. The specific parameters of algorithm are: the number of references (option -*R*) which is assigned the value 2 in the example, the number of citers (option -*C*) which is assigned the value 2 in the example and the rank of the Hausdorff Distance which is 1 (option -*H*). In the Weka documentation, it is possible to check the different configuration options that each classifier accepts.

```xml
<classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
  <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
  <multiInstanceClassifier name="weka.classifiers.mi.CitationKNN">
    <listOptions>
      -R 2 -C 2 -H 1
    </listOptions>
  </multiInstanceClassifier>
</classifier>
```

The configuration file to execute this algorithm is located in *configurations/toMI/MIML-toMI_BR_CitationKNN.config*. It must indicate the Weka classification citationKNN, along with the BR transformation that the MIML library has.

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.CitationKNN">
      <listOptions>
        -R 2 -C 2 -H 1
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_CitationKNN.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

In the case of the `<evaluator>` element, method *EvaluatorHoldout* is being used and the training and test *arff* files have been indicated, as well as the *xml* file of the dataset. For the `<report>` element, the generated output will be saved in the path *results/toMi/BR_CitationKNN.csv*. Standard deviation of the metrics will not be included (indicated by the `<standardDeviation>` element), a previous informative header will be included in the generated file (`<header>` element) and the following metrics will be included: Hamming Loss, SubsetAccuracy, Macro-averaged Precision and Macro-averaged F-Measure; in addition, with the `perLabel` attribute sets to "false" it is being indicated that the metrics for each label should not be shown in the case of macro-averaged measures.

Then, it is necessary to run *RunAlgorithm* class using the previous configuration file, with the commands:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI\_BR\_CitationKNN.config
```

The output obtained after execution is stored at *results/toMI/BR_CitationKNN.csv* as specified in configuration file, it would be the one in Table 4.5.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.CitationKNN
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_CitationKNN.config
Train_time_ms	14610
Test_time_ms	7390
Hamming Loss	0.17669172932330826
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.06753056884635832
Macro-averaged F-Measure	0.06421120114292674

Table 4.5: Output generated by the CitationKNN report

4.3.2.2 **MDD classifier**

MDD classifier [36] (Modified Diverse Density algorithm, with collective assumption) can be easily configurable using `<listOptions>` element. It is configurable through option *N* to indicate if the dataset has to be normalized (value 0), standardized (value 1) or neither (value 2). This classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_MDD.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MDD">
      <listOptions>
        -N 0
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorCV">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_birds.arff</file>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MIDD.csv</fileName>
    <standardDeviation>true</standardDeviation>
    <header>true</header>
    <measures perLabel='true'>
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

In this case, another option implemented in the library, cross-validation, has been used as evaluation method. Files related to the dataset used are indicated by *<file>* and *<xmlFile>* elements and with the *<numFolds>* element it is possible to configure the number of folds that the evaluator will use.

With respect to the specification of the output report (*<report>* element), it is specified that measures are shown *perLabel* = *true*. In this manner, in the report each measure will be shown for each label considered (it can be seen in the generated output that is shown).

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MDD.config
```

In addition, the generated output (*results/toMI/BR_MDD.csv*) will include the standard deviation of the chosen metrics along with the values obtained for each class in the case of macro-averaged (see Table 4.6).

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MDD
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds.arff
Configuration file	MIMLtoMI_BR_MDD.config
Train_time_ms(avg)	119149.2
Train_time_ms(std)	13103.088176456724
Test_time_ms(avg)	57.4
Test_time_ms(std)	49.313689782858475
Hamming Loss	0.09980543463922967
Hamming Loss Std	0.006870682833568517
Subset Accuracy	0.1031954887218045
Subset Accuracy Std	0.0487884058027673
Macro-averaged Precision	0.32254917597022864
Macro-averaged Precision Std	0.04652512101728586
Macro-averaged Precision-BRCR	0.5545454545454545
Macro-averaged Precision-BRCR Std	0.13453810835629015
Macro-averaged Precision-PAWR	0.1
Macro-averaged Precision-PAWR Std	0.10800000000000001
Macro-averaged Precision-PSFL	0.6
Macro-averaged Precision-PSFL Std	168
...	...

Table 4.6: Output generated by the MDD report

4.3.2.3 MIBoost classifier

This classifier [64] considers the geometric mean of posterior of instances inside a bag (arithmetic mean of log-posterior) and the expectation for a bag is taken inside the loss function. It can be easily configurable using *<listOptions>* element. It accepts the following parameters:

- B: the number of bins in discretization (0 to indicate no discretization).
- R: maximum number of boost iteration.
- W: full name of classifier to boost.

This classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_MIBoost.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MIBoost">
      <listOptions>
        -B 1 -R 8 -W weka.classifiers.bayes.NaiveBayes
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorCV">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_birds.arff</file>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MIBoost.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="true">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration file specifies a cross validation method with 5 folds and the output report is configured with four measures and they must be shown per label.

This configuration can be run with the command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MIBoost.config
```

The output generated, showed in Table 4.7, is saved in *results/toMI/BR_MIBoost.csv*.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MIBoost
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds.arff
Configuration file	MIMLtoMI_BR_MIBoost.config
Train_time_ms(avg)	736.6
Test_time_ms(avg)	185.2
Hamming Loss	0.1067174515235457
Subset Accuracy	0.0
Macro-averaged Precision	0.07368421052631578
Macro-averaged Precision-BRCR	0.0
Macro-averaged Precision-PAWR	0.0
Macro-averaged Precision-PSFL	0.6
...	...

Table 4.7: Output generated by the MIBoost report

4.3.2.4 MIDD classifier

It is a re-implementing of MDD [36] changing the testing procedure. It can be easily configurable using *<listOptions>* element. Concretely, this classifier is configurable with option N to indicate if the dataset must be normalized (value 0), standardized (value 1) or neither (value 2). MIDD classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIML-toMI_BR_MIDD.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MIDD">
      <listOptions>
        -N 2
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
```

```
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MIDD.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

It can be seen that the experiment is configured with holdout as validation method and four different measures are specified in the output report. In this case, they are not shown per label. If the method of validation used is holdout, it has not sense that standard deviation is shown.

This configuration can be run with the command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MIDD.config
```

The output generated and showed in Table 4.8 is saved in *results/toMI/BR_MIDD.csv*.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MIDD
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_MIDD.config
Train_time_ms	1033
Test_time_ms	22
Hamming Loss	0.11278195488721807
Subset Accuracy	0.0
Macro-averaged Precision	0.0
Macro-averaged F-Measure	0.0

Table 4.8: Output generated by the MIDD report

4.3.2.5 **MILR classifier**

MILR classifier is an adaptation of standard single-instance logistic regression to the multi-instance setting. It can be easily configurable using <*listOptions*> element. Concretely, it accepts to configure the next options:

- *R*: double value to set the ridge in the log-likelihood.
- *A*: defines the type of algorithm:
  - *0*: standard MI assumption.
  - *1*: collective MI assumption, arithmetic mean for posteriors.
  - *2*: collective MI assumption, geometric mean for posteriors.

This classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_MILR.config*:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MILR">
      <listOptions>
        -A 2
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MILR.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible run this configuration with the command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MILR.config
```

The output generated and showed in Table 4.9 is saved in *results/toMI/BR_MILR.csv*.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MILR
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_MILR.config
Train_time_ms	4213
Test_time_ms	35
Hamming Loss	0.16917293233082709
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.05037593984962406
Macro-averaged F-Measure	0.04449760765550239

Table 4.9: Output generated by the MILR report

4.3.2.6 MIOptimalBall classifier

MIOptimalBall classifier [44] tries to find a suitable ball in the multiple-instance space, with a certain data point in the instance space as a ball center. The possible ball center is a certain instance in a positive bag. The possible radiuses are those which can achieve the highest classification accuracy. The model selects the maximum radius as the radius of the optimal ball. It can be easily configurable using <*listOptions*> element. Its configuration option is the same as for MDD or MIDD classifiers. This classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_MIOptimalBall.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MIOptimalBall">
      <listOptions>
        -N 0
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MIOptimalBall.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible run this configuration with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MIOptimalBall.config
```

The output generated and showed in Table 4.10 is saved in *results/toMI/BR_MIOptimalBall.csv*.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MIOptimalBall
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_MIOptimalBall.config
Train_time_ms	1501
Test_time_ms	66
Hamming Loss	0.15319548872180455
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.10960925039872409
Macro-averaged F-Measure	0.09739558051915032

Table 4.10: Output generated by the MIOptimalBall report

4.3.2.7 **MIRI classifier**

This classifier (Multi Instance Rule Inducer) [56] utilizes partial MITI trees with a single positive leaf to learn and represent rules. It can be easily configurable using `<listOptions>` element. It accepts various parameters, such as:

- *M*: the method used to determine best split:
  - *1*: Gini.
  - *2*: MaxBEPP.
  - *3*: SSBEPP.
- *K*: the constant used in the tozero() heuristic.
- *L*: it scales the value of K to the size of the bags.
- *U*: it indicates the use of unbiased estimate rather than BEPP.
- *B*: it uses the instances present for the bag counts at each node when splitting, weighted according to 1 - Ba n̂, where n is the number of instances present which belong to the bag, and Ba is another parameter.
- *Ba*: it defines the type of algorithm: multiplier for count influence of a bag based on the number of its instances.
- *A*: the number of randomly selected attributes to split:
  - *1*: all attributes.
  - *2*: square root of the total number of attributes.
- *An*: the number of top scoring attribute splits to randomly pick from:
  - *1*: all splits (completely random selection).
  - *2*: square root of the number of splits.
- *S*: random number seed.

MIRI classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_MIRI.config*:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MIRI">
      <listOptions>
        -M 2 -U -A 1 -S 123
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MIRI.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>

</configuration>
```

The configuration of experiment determines that it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible run this configuration with the command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MIRI.config
```

The output generated and showed in Table 4.11 is saved in *results/toMI/BR_MIRI.csv*.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MIRI
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_MIRI.config
Train_time_ms	13414
Test_time_ms	2645
Hamming Loss	0.1851503759398496
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.08025030525030526
Macro-averaged F-Measure	0.07922564657027246

Table 4.11: Output generated by the MIRI report

4.3.2.8 **MISMO classifier**

MISMO classifier implements John Platt's sequential minimal optimization algorithm [41] for training a support vector classifier. It can be easily configurable using `<listOptions>` element. Concretely, these are the options that can be configured in the classifier:

- *C*: the complexity constant C.
- *N*: it indicates if the dataset must be normalized (value 0), standardized (value 1) or neither (value 2).
- *I*: it indicates using MIminimax feature space.
- *L*: the tolerance parameter.
- *P*: the epsilon for round-off error.
- *M*: it fits logistic models to SVM outputs.
- *V*: number of folds for the internal cross-validation.
- *W*: random number seed.
- *K*: full name of the kernel to use. It is important to set one which be able to handle Multi-Instance data.

For this classifier, the library has a own implementation that resolves a problem at the moment of managing dataset before prediction occurs. This wrapper, called *MISMOWrapper*, can be found in the package *miml.classifiers.mi*.

MISMO classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_MIRI.config*:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>

    <multiInstanceClassifier name="miml.classifiers.mi.MISMOWrapper">
      <listOptions>
        -L 1.0e-3 -P 1.0e-12 -N 0 -V 5
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MISMO.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
```

```
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible run this configuration with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MISMO.config
```

The output generated and showed in Table 4.12 is saved in *results/toMI/BR_MISMO.csv*.

Algorithm	MIMLClassifierToMI
Classifier	miml.classifiers.mi.MISMOWrapper
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_MISMO.config
Train_time_ms	9479
Test_time_ms	304
Hamming Loss	0.1654135338345864
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.1241063889670701
Macro-averaged F-Measure	0.10265376359559185

Table 4.12: Output generated by the MISMO report

4.3.2.9 MISVM classifier

This classifier [55] implements Stuart Andrews' SVM (Maximum pattern Margin Formulation of MIL). It can be easily configurable using <*listOptions*> element. Concretely, it accepts the following parameters:

- *C*: the complexity constant C.
- *N*: indicates if the dataset must be normalized (value 0), standardized (value 1) or neither (value 2).
- *I*: the maximum number of iterations to perform.
- *K*: full name of the kernel to use. It is important to set one which be able to handle Multi-Instance data.

MISVM classifier only accepts binary relevance as valid transformation method.

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_MISVM.config*:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MISVM">
      <listOptions>
        -C 3 -N 2 -I 750
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MISVM.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible run this configuration with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MISVM.config
```

The generated output, located in *results/toMI/BR_MISVM*, is showed in Table 4.13.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MISVM
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_MISVM.config
Train_time_ms	1572798
Test_time_ms	41
Hamming Loss	0.7349624060150376
Subset Accuracy	0.0
Macro-averaged Precision	0.09721418931945246
Macro-averaged F-Measure	0.16321472196815207

Table 4.13: Output generated by the MISVM report

4.3.2.10 MITI classifier

This classifier (Multi instance Tree Inducer) [54] is based a decision tree learned using Blockeel et al.'s algorithm [56]. It can be easily configurable using *<listOptions>* element. It can be configured with the same parameters as MIRI classifier. MITI classifier only accepts binary relevance as valid transformation method

The configuration file to execute this algorithm is located in *configurations/toMI/MIML-toMI_BR_MISVM.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance"/>
    <multiInstanceClassifier name="weka.classifiers.mi.MITI">
      <listOptions>
        -M 1 -U -A 2 -S 123
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/BR_MITI.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible run this configuration with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_BR_MITI.config
```

The generated output, located in *results/toMI/BR_MITI*, is showed in Table 4.14.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MITI
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLBinaryRelevance
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_BR_MITI.config
Train_time_ms	4944
Test_time_ms	159
Hamming Loss	0.18703007518796985
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.05325814536340852
Macro-averaged F-Measure	0.053593117408906876

Table 4.14: Output generated by the MITI report

4.3.2.11 **MIWrapper classifier**

MIWrapper It is a simple wrapper method for applying standard propositional learners to multi-instance data [57]. It can be easily configurable using `<listOptions>` element. The list of possible parameters is as follows:

- *P*: it selects the method used in testing:
  - *1*: arithmetic average
  - *2*: geometric average
  - *3*: max probability of positive bag.
- *A*: the type of weight setting for each single-instance:
  - *0*: it keeps the weight to be the same as the original value.
  - *1*: weight = 1.0.
  - *2*: weight = 1.0/Total number of single-instance in the corresponding bag.
  - *3*: weight = total number of single-instance / (Total number of bags * total number of single-instance in the corresponding bag).
- *W*: full name of base classifier.

The configuration file to execute this algorithm is located in *configurations/toMI/MIML-toMI_BR_MIWrapper.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLLabelPowerset"/>
      <multiInstanceClassifier name="weka.classifiers.mi.MIWrapper">
      <listOptions>
        -P 2 -A 1 -W weka.classifiers.rules.ZeroR
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorCV">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_birds.arff</file>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/LP_MIWrapper.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

In this case, the experiment has been configured to work with label powerset transformation method, changing the value of attribute *name* in <*transformationMethod*> element. It is used cross validation as validation method using 5 folds and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. The standard deviation considering results of the different folds is not shown.

It is possible run this configuration with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_LP_MIWrapper.config
```

The generated output, located in *results/toMI/LP_MIWrapper*, is showed in Table 4.15.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.MIWrapper
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLLabelPowerset
Dataset	miml_birds.arff
Configuration file	MIMLtoMI_LP_MIWrapper.config
Train_time_ms(avg)	9.4
Test_time_ms(avg)	14.0
Hamming Loss	0.1324462471969398
Subset Accuracy	0.08515037593984963
Macro-averaged Precision	0.087135602163303
Macro-averaged F-Measure	0.09499406048420739

Table 4.15: Output generated by the MIWrapper report

4.3.2.12 **SimpleMI classifier**

This classifier reduces MI data into mono-instance data. It can be easily configurable using <*listOptions*> element. These are the options that can be configured in the classifier:

- *M*: the method used in transformation:
  - *1*: arithmetic average.
  - *2*: geometric center.
  - *3*: using minimax combined features of a bag.
- *W*: full name of base classifier.

SimpleMI classifier only accepts binary relevance as valid transformation method

The configuration file to execute this algorithm is located in *configurations/toMI/MIMLtoMI_BR_SimpleMI.config*:

```
<configuration>

  <classifier name="miml.classifiers.miml.mimlTOmi.MIMLClassifierToMI">
    <transformationMethod name="miml.classifiers.miml.mimlTOmi.MIMLLabelPowerset"/>
    <multiInstanceClassifier name="weka.classifiers.mi.SimpleMI">
      <listOptions>
        -W weka.classifiers.rules.ZeroR -M 2
      </listOptions>
        </multiInstanceClassifier>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toMI/LP_SimpleMI.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible run this configuration with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toMI/MIMLtoMI_LP_SimpleMI.config
```

The generated output, located in *results/toMI/LP_SimpleMI*, is showed in Table 4.16.

Algorithm	MIMLClassifierToMI
Classifier	weka.classifiers.mi.SimpleMI
Transformation method	miml.classifiers.miml.mimlTOmi.MIMLLabelPowerset
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoMI_LP_SimpleMI.config
Train_time_ms	3
Test_time_ms	12
Hamming Loss	0.12124060150375944
Subset Accuracy	0.10714285714285714
Macro-averaged Precision	0.06954887218045112
Macro-averaged F-Measure	0.07823613086770981

Table 4.16: Output generated by the SimpleMI report

4.3.3 MIML algorithms transforming MIML problem to ML problem

In this section, it is shown a set of examples with the different algorithms that transform the MIML problem into an ML problem and then, it is used a multi-label algorithm to solve the problem.

In a same way as before, it is possible to consult the MULAN algorithms that can be used in this kind of problems in Table 4.3.

In the configuration file, it is necessary to specify the transformation algorithm that converts the MIML problem into ML problem and the ML classifier that you want to apply. The transformation methods that has available the library are shown in the section 4.2.3.1 and 4.2.3.2.

Below, it is shown the classifier configuration that it is very similar to the one detailed in the previous section. It contains two elements: `<multiLabelClassifier>` element to indicate the ML classifier which solves the problem and `<transformationMethod>` element to indicate the transformation method which converts MIML problem to ML problem.

```
 <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
   <multiLabelClassifier name="xxxxxxxxxxx" />
   <transformationMethod name="xxxxxxxxxxx"/>
 </classifier>
```

4.3.3.1 BinaryRelevance Classifier

BinaryRelevance classifier builds one binary model per label. In the case of Mulan classifiers, these can be configured through the group of labels `<parameters>` and `<parameter>`, using the attributes `class` and `value`. It is very important to bear in mind that in order to avoid any error during the execution of the experiment it is necessary that the configuration is adjusted to any constructor that the classifier has: in this case the Mulan BinaryRelevance classifier has a constructor that needs a parameter of class *weka.classifiers.Classifier*; to specify this, we use attributes pairs `class` and `value` inside a `<parameter>` element, in the first one, the type of parameter in question will be indicated referring to its class and in the second one the specific value of the parameter. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor.

This classifier accepts the following parameter:

- *classifier*: *weka.classifiers.Classifier* class. Base level classification algorithm that will be used for training each of the binary models.

The configuration file to execute this algorithm is located in *configurations/toML/MIML-toML_AT_BR.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.BinaryRelevance">
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.lazy.IBk"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.ArithmeticTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
```

```
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>


  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/AT_BR.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that the classifier specified for binary relevance is the algorithm IBk of Weka's classifiers. Moreover, it is used holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense that standard deviation is shown.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_AT_BR.config
```

The generated output, located in *results/toML/AT_BR*, is showed in Table 4.17.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.BinaryRelevance
Transformation method	miml.transformation.mimlTOml.ArithmeticTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_AT_BR.config
Train_time_ms	14
Test_time_ms	137
Hamming Loss	0.1860902255639097
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.07097288676236044
Macro-averaged F-Measure	0.06823810281144978

Table 4.17: Output generated by the BinaryRelevance report

4.3.3.2 **BRkNN Classifier**

BRkNN [61] is a simple binary relevance implementation of the KNN algorithm. It can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *numOfNeighbours*: *int* value. The number of neighbours to use.

- *ext*: *mulan.classifier.lazy.BRkNN$ExtensionType* enum value. Extension to use, it can take the next values:
  - *NONE*: standard binary relevance.
  - *EXTA*: predict top ranked label in case of empty prediction set.
  - *EXTB*: predict top n ranked labels based on size of labelset in neighbours.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_AT_BRkNN.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.lazy.BRkNN">
      <parameters>
        <parameter class="int.class" value="5"/>
        <parameter class="mulan.classifier.lazy.BRkNN$ExtensionType" value="EXTB"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.ArithmeticTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/AT_BRkNN.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="true">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with the *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and each measure will be calculated by label. If the method of validation used is holdout, it has not sense to set a true the standard deviation.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_AT_BRkNN.config
```

The generated output, located in *results/toML/AT_BRkNN*, is showed in Table 4.18.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.lazy.BRkNN
Transformation method	miml.transformation.mimlTOml.ArithmeticTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_AT_BRkNN.config
Train_time_ms	2
Test_time_ms	31
Hamming Loss	0.17857142857142855
Subset Accuracy	0.0
Macro-averaged Precision	0.13137410834779253
Macro-averaged Precision-BRCR	0.25
Macro-averaged Precision-PAWR	0.16666666666666666
Macro-averaged Precision-PSFL	1.0
Macro-averaged Precision-RBNU	0.0
Macro-averaged Precision-DEJU	0.0
Macro-averaged Precision-OSFL	0.15384615384615385
Macro-averaged Precision-HETH	0.0
Macro-averaged Precision-CBCH	0.1875
Macro-averaged Precision-VATH	0.21428571428571427
Macro-averaged Precision-HEWA	0.38095238095238093
...	...

Table 4.18: Output generated by the BRkNN report

4.3.3.3 ClassifierChains Classifier

The Classifier Chains model (CC) involves L binary transformations—one for each label—as in BR [24]. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameter:

- *classifier*: *weka.classifiers.Classifier* class. Base level classification algorithm that will be used for training each of the binary models.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_AT_CC.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.ClassifierChain">
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.trees.J48"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.ArithmeticTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/AT_CC.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_AT_CC.config
```

The generated output, located in *results/toML/AT_CC*, is showed in Table 4.19.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.ClassifierChain
Transformation method	miml.transformation.mimlTOml.ArithmeticTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_AT_CC.config
Train_time_ms	454
Test_time_ms	16
Hamming Loss	0.15977443609022554
Subset Accuracy	0.05357142857142857
Macro-averaged Precision	0.10373398531293268
Macro-averaged F-Measure	0.09977011494252871

Table 4.19: Output generated by the ClassifierChain report

4.3.3.4 DMLkNN Classifier

This classifier implementing the Dependent Multi Label k Nearest Neighbours [62] which is derived from Multi Label kNN but taking into account the dependencies between labels. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *numOfNeighbours*: *int* value. The number of neighbours to use.
- *smooth*: *double* value. Smoothing factor to use.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_AT_DMLkNN.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.lazy.DMLkNN">
      <parameters>
        <parameter class="int.class" value="5"/>
        <parameter class="double.class" value="0.8"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.ArithmeticTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/AT_DMLkNN.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_AT_DMLkNN.config
```

The generated output, located in *results/toML/AT_DMLkNN*, is showed in Table 4.20.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.lazy.DMLkNN
Transformation method	miml.transformation.mimlTOml.ArithmeticTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_AT_DMLkNN.config
Train_time_ms	46
Test_time_ms	41
Hamming Loss	0.1268796992481203
Subset Accuracy	0.03571428571428571
Macro-averaged Precision	0.12205513784461151
Macro-averaged F-Measure	0.08601889338731444

Table 4.20: Output generated by the DMLkNN report

4.3.3.5 PrunedSets Classifier

Pruned Sets (PS) [28] is similar to label powerset but it focuses on the most important relationships of labels by pruning the infrequently occurring labelsets, reducing the complexity of the algorithm. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *classifier*: *weka.classifiers.Classifier* class. Base single label classification algorithm.
- *aP*: *int* value. Number of instances required for a labelset to be included.
- *aStrategy*: *mulan.classifier.transformation.PrunedSets$Strategy* enum value. Strategy for processing infrequent labelsets, it can take the next values:
  - *A*: rank subsets firstly by the number of labels they contain and secondly by the times they occur, then keep top *b* ranked.
  - *B*: keep all subsets of size greater than *b*.
- *aB*: *int* value. Parameter of the strategy for processing infrequent labelsets.

The configuration file to execute this algorithm is located in *configurations/toML/MIML-toML_MMT_PS.config*:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.PrunedSets">
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.trees.J48"/>
        <parameter class="int.class" value="4"/>
        <parameter class="mulan.classifier.transformation.PrunedSets$Strategy" value="A"/>
        <parameter class="int.class" value="3"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.MinMaxTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/MMT_PS.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_MMT_PS.config
```

The generated output, located in *results/toML/MMT_PS*, is showed in Table 4.21.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.PrunedSets
Transformation method	miml.transformation.mimlTOml.MinMaxTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_MMT_PS.config
Train_time_ms	29
Test_time_ms	17
Hamming Loss	0.12312030075187973
Subset Accuracy	0.03571428571428571
Macro-averaged Precision	0.06983805668016194
Macro-averaged F-Measure	0.07819548872180451

Table 4.21: Output generated by the PrunedSets report

4.3.3.6 EnsembleOfClassifierChains Classifier

It is a implementation of a ensemble of Classifier Chains [24] classifiers. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *classifier*: *weka.classifiers.Classifier* class. Base classifier for each ClassifierChain model.
- *aNumOfModels*: *int* value. Number of models.
- *doUseConfidences*: *boolean* value. Whether to use confidences or not.
- *doUseSamplingWithReplacement*: *boolean* value. Whether to use sampling with replacement or not.

The configuration file to execute this algorithm is located in *configurations/toML/MIML-toML_AT_ECC.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.EnsembleOfClassifierChains"
        >
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.lazy.IBk"/>
        <parameter class="int.class" value="10"/>
        <parameter class="boolean.class" value="true"/>
        <parameter class="boolean.class" value="true"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.ArithmeticTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/AT_ECC.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_AT_ECC.config
```

The generated output, located in *results/toML/AT_ECC*, is showed in Table 4.22.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.EnsembleOfClassifierChains
Transformation method	miml.transformation.mimlTOml.ArithmeticTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_AT_ECC.config
Train_time_ms	81
Test_time_ms	1010
Hamming Loss	0.1832706766917293
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.06850517903149482
Macro-averaged F-Measure	0.06518358296059126

Table 4.22: Output generated by the EnsembleOfClassifierChains report

4.3.3.7 **EnsembleOfPrunedSets Classifier**

An implementation of a ensemble of a Pruned Sets [28] classifiers. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *aPercentage*: *double* value. Percentage of data to sample.
- *aNumOfModels*: *int* value. Number of models in the ensemble.
- *aThreshold*: *double* value. Threshold for producing bipartitions.
- *aP*: *int* value. Number of instances required for a labelset to be included.
- *aStrategy*: *mulan.classifier.transformation.PrunedSets$Strategy* enum value. Strategy for processing infrequent labelsets, it can take the next values:
  - *A*: rank subsets firstly by the number of labels they contain and secondly by the times they occur, then keep top *b* ranked.
  - *B*: keep all subsets of size greater than *b*.
- *aB*: *int* value. Parameter of the strategy for processing infrequent labelsets.
- *baselearner*: *weka.classifiers.Classifier* class. Base learner.

An example of configuration file could be:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.EnsembleOfPrunedSets">
      <parameters>
        <parameter class="double.class" value="60"/>
        <parameter class="int.class" value="10"/>
        <parameter class="double.class" value="0.6"/>
        <parameter class="int.class" value="3"/>
        <parameter class="mulan.classifier.transformation.PrunedSets$Strategy" value="B"/>
        <parameter class="int.class" value="3"/>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.lazy.IBk"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.ArithmeticTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/AT_EPS.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_AT_EPS.config
```

The generated output, located in *results/toML/AT_EPS*, is showed in Table 4.23.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.EnsembleOfPrunedSets
Transformation method	miml.transformation.mimlTOml.ArithmeticTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_AT_EPS.config
Train_time_ms	18
Test_time_ms	18
Hamming Loss	0.12312030075187971
Subset Accuracy	0.03571428571428571
Macro-averaged Precision	0.09122807017543859
Macro-averaged F-Measure	0.09034618632141853

Table 4.23: Output generated by the EnsembleOfPrunedSets report

4.3.3.8 HOMER Classifier

Hierarchy Of Multi-label classifiERs (HOMER) is a method designed for domains with large number of labels. It transform a multi-label classification problem into a tree-shaped hierarchy of simpler multi-label problems [63]. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *mll*: *mulan.classifier.MultiLabelLearner* class. Multi Label learner.
- *clusters*: *int* value. Number of partitions.
- *method*: *mulan.classifier.meta.HierarchyBuilder$Method* enum value. Partitioning method, it can take the next values:
  - *Random*: random balanced distribution of labels.
  - *Clustering*: distribution based on label similarity.
  - *BalancedClustering*: balanced distribution based on label similarity.

When one of the parameters is a Multi Label classifier, it is possible to configure it following the same strategy through the labels `<parameters>` and `<parameters>`, as it is shown in the following configuration that is located in *configurations/toML/MIMLtoML_GT_HOMER.config*:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.meta.HOMER">
      <parameters>
        <parameter class="mulan.classifier.MultiLabelLearner" value="mulan.classifier.
            transformation.BinaryRelevance">
          <parameters>
            <parameter class="weka.classifiers.Classifier" value="weka.classifiers.trees.J48
                "/>
          </parameters>
        </parameter>
        <parameter class="int.class" value="3"/>
        <parameter class="mulan.classifier.meta.HierarchyBuilder$Method" value="
            BalancedClustering"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.GeometricTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/GT_HOMER.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_GT_HOMER.config
```

The generated output, located in *results/toML/GT_HOMER*, is showed in Table 4.24.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.meta.HOMER
Transformation method	miml.transformation.mimlTOml.GeometricTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_GT_HOMER.config
Train_time_ms	209
Test_time_ms	17
Hamming Loss	0.1973684210526315
Subset Accuracy	0.0
Macro-averaged Precision	0.0698766146134567
Macro-averaged F-Measure	0.0700643654591023

Table 4.24: Output generated by the HOMER report

4.3.3.9 **IBLR_ML Classifier**

Instance-Based Learning by Logistic Regression for Multi Label [26] use Bayesian techniques to consider the labels associated with nearest neighbours of the new instance as additional characteristics. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *numNeighbours*: *int* value. Number of nearest neighbours considered.
- *addFeatures*: *boolean* value. When true, *IBLR-ML+* is used, *IBLR-ML* implementation with some features.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_GT_IBLR_ML.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.lazy.IBLR_ML">
      <parameters>
        <parameter class="int.class" value="5"/>
        <parameter class="boolean.class" value="true"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.GeometricTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorCV">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_birds.arff</file>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/GT_IBLR_ML.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used cross validation with 5 folds as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label will not be shown. The standard deviation of results of each measure for the different folds will not be shown.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_GT_IBLR_ML.config
```

The generated output, located in *results/toML/GT_IBLR_ML*, is showed in Table 4.25.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.lazy.IBLR_ML
Transformation method	miml.transformation.mimlTOml.GeometricTransformation
Dataset	miml_birds.arff
Configuration file	MIMLtoML_GT_IBLR_ML.config
Train_time_ms(avg)	550.8
Test_time_ms(avg)	20.6
Hamming Loss	0.16325023084025853
Subset Accuracy	0.024937343358395987
Macro-averaged Precision	0.24936516890178217
Macro-averaged F-Measure	0.27304135405965185

Table 4.25: Output generated by the IBLR_ML report

4.3.3.10 **LabelPowerset Classifier**

It is a implementation of the label powerset (LP) algorithm. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameter:

- *classifier*: *weka.classifiers.Classifier* class. Base single label classification algorithm that will be used for training each of the binary models.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_GT_LP.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.LabelPowerset">
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.lazy.IBk"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.GeometricTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
```

```
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/GT_LP.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label will not be shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_GT_LP.config
```

The generated output, located in *results/toML/GT_LP*, is showed in Table 4.26.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.LabelPowerset
Transformation method	miml.transformation.mimlTOml.GeometricTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_GT_LP.config
Train_time_ms	4
Test_time_ms	12
Hamming Loss	0.18796992481202998
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.05834457939721098
Macro-averaged F-Measure	0.05542919164828074

Table 4.26: Output generated by the LabelPowerset report

4.3.3.11 MLkNN Classifier

This classifier [25] derived from the traditional K-nearest neighbour (KNN) algorithm. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *numNeighbours*: *int* value. Number of nearest neighbours considered.
- *smooth*: *double* value. Smoothing factor.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_GT_MLkNN.config*:

```xml
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.lazy.MLkNN">
      <parameters>
        <parameter class="int.class" value="5"/>
        <parameter class="double.class" value="1.1"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.GeometricTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorCV">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_birds.arff</file>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/GT_MLkNN.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used cross validation with 5 folds as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. The standard deviation of results of different folds for each measure will be not shown.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_GT_MLkNN.config
```

The generated output, located in *results/toML/GT_MLkNN*, is showed in Table 4.27.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.lazy.MLkNN
Transformation method	miml.transformation.mimlTOml.GeometricTransformation
Dataset	miml_birds.arff
Configuration file	MIMLtoML_GT_MLkNN.config
Train_time_ms(avg)	25.4
Test_time_ms(avg)	14.4
Hamming Loss	0.08986941036802534
Subset Accuracy	0.14548872180451128
Macro-averaged Precision	0.33440989007552474
Macro-averaged F-Measure	0.24172374823575846

Table 4.27: Output generated by the MLkNN report

4.3.3.12 **MultiLabelStacking Classifier**

Implementation of the *2BR* or Multi-Label stacking method [53]. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *baseClassifier*: *weka.classifiers.Classifier* class. Classifier used in the base-level.
- *metaClassifier*: *weka.classifiers.Classifier* class. Classifier used in the meta-level.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_GT_MLStacking.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.MultiLabelStacking">
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.trees.J48"/>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.trees.LMT"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.GeometricTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/GT_MLStacking.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label will not be shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_GT_MLStacking.config
```

The generated output, located in *results/toML/GT_MLStacking*, is showed in Table 4.28.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.MultiLabelStacking
Transformation method	miml.transformation.mimlTOml.GeometricTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_GT_MLStacking.config
Train_time_ms	181314
Test_time_ms	19
Hamming Loss	0.6174812030075191
Subset Accuracy	0.0
Macro-averaged Precision	0.12205513784461149
Macro-averaged F-Measure	0.1126981462115026

Table 4.28: Output generated by the MultiLabelStacking report

4.3.3.13 RAkEL Classifier

This classifier (RAndomk-labELsets) [27] randomly breaks the set of labels into several small-sized labelsets. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameters:

- *baseLearner*: *mulan.classifier.MultiLabelLearner* class. Multi Label base learner.
- *models*: *int* value. Number of models to use.
- *subset*: *int* value. Size of subsets.
- *threshold*: *double* value. Threshold to use.

The configuration file to execute this algorithm is located in *configurations/toML/MIML-toML_MMT_RAkEL.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.meta.RAkEL">
      <parameters>
        <parameter class="mulan.classifier.MultiLabelLearner" value="mulan.classifier.
            transformation.BinaryRelevance">
          <parameters>
            <parameter class="weka.classifiers.Classifier" value="weka.classifiers.trees.J48
                "/>
          </parameters>
        </parameter>
        <parameter class="int.class" value="5"/>
        <parameter class="int.class" value="10"/>
        <parameter class="double.class" value="0.6"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.MinMaxTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/MMT_RAkEL.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label will not be shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_MMT_RAkEL.config
```

The generated output, located in *results/toML/MMT_RAkEL*, is showed in Table 4.29.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.meta.RAkEL
Transformation method	miml.transformation.mimlTOml.MinMaxTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_MMT_RAkEL.config
Train_time_ms	556
Test_time_ms	15
Hamming Loss	0.1795112781954886
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.07211516553621818
Macro-averaged F-Measure	0.06632803919085986

Table 4.29: Output generated by the RAkEL report

4.3.3.14 **Pairwise Classifier**

Implementation of the Ranking by Pairwise Comparisons (RPC) [59] algorithm, whose key idea is to reduce the problem of label ranking to several binary classification problem. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameter:

- *classifier*: *weka.classifiers.Classifier* class. The binary classification algorithm to use.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_MMT_RPC.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.Pairwise">
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.lazy.LWL"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.MinMaxTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorCV">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_birds.arff</file>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/MMT_RPC.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used cross validation with 5 folds as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label will not be shown. The standard deviation of results of different folds for each measure will not be shown.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_MMT_RPC.config
```

The generated output, located in *results/toML/MMT_RPC*, is showed in Table 4.30.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.Pairwise
Transformation method	miml.transformation.mimlTOml.MinMaxTransformation
Dataset	miml_birds.arff
Configuration file	MIMLtoML_MMT_RPC.config
Train_time_ms(avg)	31.2
Train_time_ms(std)	8.109253973085316
Test_time_ms(avg)	307.8
Test_time_ms(std)	22.63095225570502
Average Precision	0.6278645441017462
Average Precision Std	0.03852900641736514
Coverage	5.135213032581453
Coverage Std	0.37465494706092933
OneError	0.38621553884711785
OneError Std	0.049193005823091

Table 4.30: Output generated by the Pairwise report

4.3.3.15 CalibratedLabelRanking classifier

Implementation of the Calibrated Label Ranking (CLR) [60] algorithm. The key idea of this classifier is to introduce an artificial calibration label that, in each example, separates the relevant from the irrelevant labels. Similarly to the other ML classifiers, it can be configured using the `<parameters>` element and `<parameter>` element for each parameter. Each parameter has two attributes to specify: the attributes `class` which indicates the parameter class and the attribute `value` which indicates parameter value. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor. Concretely, this algorithm accepts the following parameter:

- *classifier*: *weka.classifiers.Classifier* class. The binary classification algorithm to use.

The configuration file to execute this algorithm is located in *configurations/toML/MIMLtoML_MMT_CLR.config*:

```
<configuration>
  <classifier name="miml.classifiers.miml.mimlTOml.MIMLClassifierToML">
    <multiLabelClassifier name="mulan.classifier.transformation.CalibratedLabelRanking">
      <parameters>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.lazy.IBk"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.MinMaxTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/toML/MMT_CLR.csv</fileName>
    <standardDeviation>true</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

The configuration of experiment determines that it is used holdout with *birds* dataset as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label are not shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/toML/MIMLtoML_MMT_CLR.config
```

The generated output, located in *results/toML/MMT_CLR*, is showed in Table 4.31.

Algorithm	MIMLClassifierToML
Classifier	mulan.classifier.transformation.CalibratedLabelRanking
Transformation method	miml.transformation.mimlTOml.MinMaxTransformation
Dataset	miml_birds_random_80train.arff
Configuration file	MIMLtoML_MMT_CLR.config
Train_time_ms	24
Test_time_ms	359
Hamming Loss	0.18984962406015032
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.056551635499003904
Macro-averaged F-Measure	0.05442451807394772

Table 4.31: Output generated by the CalibratedLabelRanking report

4.3.4 MIML algorithms without transforming the problem

These classifiers are able to directly manage a dataset with a representation of the information in MIML format, not being necessary to do any previous transformation.

Currently, the library counts with two different implementations of this type of algorithms: the first one is the MultiInstance MultiLabel k-nearest neighbour [65] and the second one is an ensemble algorithm using bagging [50].

4.3.4.1 MIMLkNN Algorithm

MIMLkNN [2] solves MIML problems using the popular k-nearest neighbour (kNN) techniques. This classifier not only considers its neighbours, but also considers its citers which regard it as their own neighbours. This idea of utilizing citers to help learn from MIML examples is motivated from the Citation-kNN approach [43], where citers are found to be beneficial to learn from examples with Multi instance representation.

This classifier accepts the following parameters:

- *nReferences*: number of references or neighbours that the classifier has to consider.
- *nCiters*: number of citers that the classifier has to consider.
- *metric*: type of metric used to measure the distance among the different bags that make up the dataset in its spatial representation.

Following, a configuration example of the MIMLkNN algorithm is shown.

```
<configuration>
  <classifier name="miml.classifiers.miml.lazy.MIMLkNN">
    <nReferences>4</nReferences>
    <nCiters>6</nCiters>
    <metric name="miml.core.distance.AverageHausdorff"></metric>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds_random_80train.arff</trainFile>
      <testFile>data/miml_birds_random_20test.arff</testFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/MIMLkNN.csv</fileName>
    <standardDeviation>false</standardDeviation>
    <header>true</header>
    <measures perLabel="false">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
    </measures>
  </report>
</configuration>
```

In this case, it is necessary to use the `<nReferences>`, `<nCiters>` and `<metric>` elements to configure the different parameters of algorithm (they have been commented previously).

The `<metric>` element accepts one of three different metrics which the library has implemented: *AverageHausdorff*, *MinimalHausdorff* and *MaximalHausdorff*. All of them are included in the *miml.core.distance* package. The library also has the (*IDistance*) interface, which defines the different methods that a metric must have if you want to develop a new one.

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout as validation method and four specific measures will be shown in the output file where a header will be specified and the measures by label will not be shown. If the method of validation used is holdout, it has not sense to set a true value the standard deviation parameter.

It is possible to run the algorithm using the previous configuration file with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/MIMLClassifier/MIMLkNN.config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLkNN.csv*, is shown in Table 4.32.

Algorithm	MIMLkNN
Dataset	miml_birds_random_80train.arff
Configuration File	MIMLkNN.config
Train_time_ms	454
Test_time_ms	314
Hamming Loss	0.35338345864661663
Subset Accuracy	0.017857142857142856
Macro-averaged Precision	0.10580937898915906
Macro-averaged F-Measure	0.13529004368001735

Table 4.32: Output generated by the MIMLkNN report

4.3.4.2 **MIMLBagging Algorithm**

This algorithm is an adaptation of the traditional bagging strategy of the machine learning [50]. Consists of generating m different classifiers, each of which will work with a different dataset formed from the original, by means of a uniform sampling and with replacement (or not).

MIMLBagging is parameterized by the following options:

- *threshold*: threshold used fro predictions.

- *seed*: seed for randomization.
- *sampleWithReplacement*: determines whether the classifier will consider sampling with replacement.
- *useConfidences*: determines whether confidences [0,1] or relevance 0,1 is used to compute bipartition.
- *samplePercentage*: percentage used in sampling.
- *numClassifiers*: number of classifiers in the ensemble.
- *baseLearner*: base classifier used in the ensemble.

The configuration file to execute this algorithm is located in *configurations/MIML/MIMLClassifierEnsemble*:

```
<configuration>
  <classifier name="miml.classifiers.miml.meta.MIMLBagging">
    <threshold>0.5</threshold>
    <seed>1</seed>
    <sampleWithReplacement>true</sampleWithReplacement>
    <useConfidences>false</useConfidences>
    <samplePercentage>50</samplePercentage>
    <numClassifiers>4</numClassifiers>
    <baseLearner name="miml.classifiers.miml.lazy.MIMLkNN">
      <nReferences>2</nReferences>
      <nCiters>2</nCiters>
      <metric name="miml.core.distance.AverageHausdorff">
      </metric>
    </baseLearner>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorCV">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_birds.arff</file>
      <xmlFile>data/miml_birds.xml</xmlFile>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/MIMLBagging.csv</fileName>
    <standardDeviation>true</standardDeviation>
    <header>true</header>
    <measures perLabel="true">
      <measure>Hamming Loss</measure>
      <measure>Subset Accuracy</measure>
      <measure>Macro-averaged Precision</measure>
      <measure>Macro-averaged F-Measure</measure>
      <measure>Geometric Mean Average Interpolated Precision</measure>
    </measures>
  </report>
</configuration>
```

In this case, the base learner will be indicated in the attribute *name* of the `<baseLearner>` element. The classifier used is *MIMLkNN* which has been configurated in previous section. The rest of parameters of algorithm are configurated with the elements: `<threshold>`, `<seed>`,

`<sampleWithReplacement>`, `<useConfidence>`, `<samplePercentage>` and `<numClassifiers>`. All these parameters have been defined previously in the specification of this algorithm.

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined cross validation with 5 folds as validation method and five specific measures will be shown in the output file where a header will be specified and the different measures by label will be shown. Moreover, the standard deviation of the results of each fold for each measure also will be shown.

It is possible to run it with the following command:

```
$ java -cp target/miml-1.0-jar-with-dependencies.jar miml.run.RunAlgorithm -c
    configurations/MIMLClassifier/MIMLClassifierEnsemble.config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLClassifierEnsemble.csv*, is showed in Table 4.33.

Algorithm	MIMLBagging
Dataset	miml_birds.arff
Configuration File	MIMLClassifierEnsemble.config
Train_time_ms(avg)	573.8
Train_time_ms(std)	86.89625998856339
Test_time_ms(avg)	663.8
Test_time_ms(std)	79.59748739753033
Hamming Loss	0.24252407334124784
Hamming Loss Std	0.059348755339779874
Subset Accuracy	0.003508771929824561
Subset Accuracy Std	0.007017543859649122
Macro-averaged Precision	0.13188726413529264
Macro-averaged Precision Std	0.015318545773144282
Macro-averaged Precision-BRCR	0.3380952380952381
Macro-averaged Precision-BRCR Std	0.051316326530612244
...	...

Table 4.33: Output generated by the MIMLBagging report

4.4 Developing a new classification MIML algorithm in the library

MIML library provides the necessary components to develop new algorithms easily. On the one hand, new proposals of MI algorithms included in Weka or ML algorithms included in Mulan can be easily incorporated using the configuration file and giving its appropriate specification. On the other hand, proposed of MIML algorithms can also be easily included taking advantage of the functionality available in the library such as: problem transformation methods, management of MIML data sets, evaluation methods and the generation of output reports.

In this section, all necessary steps are shown to make your own MIML classifier from the library functionalities. The development of the MIMLkNN algorithm, already implemented in the library, is shown as example in this section.

4.4.1 Classifier location

Any classification algorithm should be included within the package *miml.classifiers.miml*. Currently, in this package there are the following categories: *lazy*, *meta*, *mimlTOmi*, *mimilTOml*. New categories could be included. In our case, the proposal would be included in the *lazy* subpackage.

Then, the class that represents the classifier is created in the package selected. In the case of the example shown, the *MIMLkNN* class is included in *miml.classifiers.miml.lazy* package.

```java
package miml.classifiers.miml.lazy;

public class MIMLkNN {
}
```

4.4.2 Classifier development

Once the algorithm class has been created in its corresponding package, the classifier development can begin. The first necessary step is to extend the *MIMLClassifier* class (for it, it is necessary to import it from the *miml.classifiers.miml* package). This class contains the general methods shared by all the MIML classification algorithms; In addition, it also implements a series of essential interfaces (*IMIMLCLassifier* and *IConfiguration*) that indicate the methods which are necessary to develop in our algorithm. These methods are: *buildInternal()*, *makePredictionInternal()* and *configure()*.

```java
package miml.classifiers.miml.lazy;

import org.apache.commons.configuration2.Configuration;

import miml.classifiers.miml.MIMLClassifier;
import miml.data.Bag;
import miml.data.MIMLInstances;
import mulan.classifier.InvalidDataException;
import mulan.classifier.MultiLabelOutput;

public class MIMLkNN extends MIMLClassifier {


       private static final long serialVersionUID = -3730384229928987460L;

       /**
        * No-argument constructor for xml configuration.
        */
       public MIMLkNN() {
       }

       @Override
       protected void buildInternal(MIMLInstances trainingSet) throws Exception {

       }

       @Override
       protected MultiLabelOutput makePredictionInternal(Bag instance) throws
           Exception, InvalidDataException {
              return null;
       }

       @Override
       public void configure(Configuration configuration) {

       }

}
```

As it can be seen, in addition to the specified methods, a long-type variable named *serialVersionUID* has also been created. This is because our class is serializable and although it is not required to implement this variable, it is strongly recommended to avoid possible errors at run time. Moreover, it is necessary that the algorithm implements, at least, an empty constructor, which is used by the *ConfigLoader* class.

Below, there is a brief explanation of what is expected to be implemented in each method:

- `public void configure(Configuration configuration)`: it receives a *Configuration* object (belonging to the *org.apache.commons.configuration2* package). This method loads the configuration given by the configuration file. The element which receives if the `<classifier></classifier>` elements with all its subelements. Therefore, all parameters that are considered configurable must be read and assigned in this method.

```java
public void configure(Configuration configuration) {

        this.nReferences = configuration.getInt("nReferences", 1);
        this.nCiters = configuration.getInt("nCiters", 1);

        try {
                // Get the name of the metric class
                String metricName = configuration.getString("metric[@name]",
                    "core.distance.AverageHausdorff");
                // Instance class
                Class<? extends IDistance> metricClass = (Class<? extends
                    IDistance>) Class.forName(metricName);

                this.metric = metricClass.newInstance();
        } catch (Exception e) {
                e.printStackTrace();
                System.exit(1);
        }
}
```

  In this case, two values of type `int` have been read from the file and have been assigned in variables that previously have been created in the class (*nReferences* and *nCiters*). In addition, the distance metric to be used in the algorithm has also been read and instantiated.

- `protected void buildInternal(MIMLInstances trainingSet)`: it receives a *MIMLInstances* object (included in the *miml.data* package). Here you must build the learning model from the training dataset you receive as parameter.

```java
protected void buildInternal(MIMLInstances trainingSet) throws Exception {
        if (trainingSet == null) {
                throw new ArgumentNullException("trainingSet");
        }

        this.dataset = trainingSet;
        d_size = trainingSet.getNumBags();

        // Change num_references if its necessary
        if (d_size <= num_references)
              num_references = d_size - 1;

        // Initialize matrices
        t_matrix = new double[d_size][numLabels];
        phi_matrix = new double[d_size][numLabels];

        calculateDatasetDistances();
        calculateReferenceMatrix();

        for (int i = 0; i < d_size; ++i) {
              Integer[] neighbours = getUnionNeighbours(i);
              // Update matrices
              phi_matrix[i] = calculateRecordLabel(neighbours).clone();
              t_matrix[i] = getBagLabels(i).clone();
        }

        weights_matrix = getWeightsMatrix();

}
```

- `protected MultiLabelOutput makePredictionInternal(Bag instance)`: it receives a bag of instances (from *miml.data* package). Thus, the classifier built in previous step is used to predict the bag class. This method returns a *MultiLabelOutput* from the MULAN library. The way to represent the output of a MIML Learner is very varied, for more detail read the MULAN documentation about the *MultiLabelOutput* class.

```java
@Override
protected MultiLabelOutput makePredictionInternal(Bag instance) throws
    Exception, InvalidDataException
        // Create a new distances matrix
        double[][] distanceMatrixCopy = distance_matrix.clone();
        distance_matrix = new double[d_size + 1][d_size + 1];

        ...

        MultiLabelOutput finalDecision = new MultiLabelOutput(predictions,
            confidences);
        // Restore original distance matrix
        distance_matrix = distanceMatrixCopy.clone();

        return finalDecision;
}
```

  In this case, the predictions and confidence values are used to represent the predicted output of the model.

All classes that are necessary for the development of these methods should be able to be imported without problem if the installation guide of the library detailed in chapter 3 has been followed correctly.

Once these methods have been implemented, the algorithm is included in the library and directly follows the same configuration as the rest of the algorithms, for the evaluator and the report you can use any method available in the library, without having to implement anything special and being able to easily carry out a similar comparative study with the rest of the algorithms already included in the library. Focusing all efforts on implementing the new classifier.

It can be seen that it has not been necessary to include specific information to work with the data set, these classes are used of the available classes in the library and only, it is necessary to specify them in the configuration file.

CHAPTER 5

Reporting bugs

Feel free to open an issue at Github if anything is not working as expected https//github.com/kdis-lab/miml/issues. Merge request are also encouraged, it will be carfully reviewed and merged if everything is all right.

APPENDIX 6

API reference

6.1 Package miml.core.distance

Package Contents *Page*

6.1.1 Interface IDistance

Interface to implements the metrics used to measure the distance between `MIMLBag` (6.5.1) of a data sets.

6.1.1.1 Declaration

public interface IDistance
extends java.io.Serializable

6.1.1.2 All known subinterfaces

MinimalHausdorff (6.1.4), MaximalHausdorff (6.1.3), AverageHausdorff (6.1.2)

6.1.1.3 All classes known to implement interface

MinimalHausdorff (6.1.4), MaximalHausdorff (6.1.3), AverageHausdorff (6.1.2)

6.1.1.4 Method summary

distance(Instances, Instances) Get the distance between two bags in the form of a set of `Instances` .
distance(MIMLBag, MIMLBag) Get the distance between two `MIMLBag` (6.5.1).

6.1.1.5 Methods

- **distance**

  double distance(weka.core.Instances first,weka.core.Instances second) **throws** java.lang.Exception

  - **Description**
    Get the distance between two bags in the form of a set of `Instances` .
  - **Parameters**
    * `first` – First bag as instances.
    * `second` – Second Bag as Instances.
  - **Returns** – Distance between two bags.
  - **Throws**
    * `java.lang.Exception` – if occurred an error during distance calculation.

- **distance**

  double distance(miml.data.MIMLBag first,miml.data.MIMLBag second) **throws** java.lang.Exception

  - **Description**
    Get the distance between two `MIMLBag` (6.5.1).
  - **Parameters**
    * `first` – First bag.
    * `second` – Second bag.
  - **Returns** – Distance between two bags.
  - **Throws**
    * `java.lang.Exception` – if occurred an error during distance calculation,

6.1.2 Class AverageHausdorff

Class that implements Average Hausdorff metric to measure the distance between 2 bags of a data set.

6.1.2.1 Declaration

public class AverageHausdorff
extends java.lang.Object **implements** IDistance

6.1.2.2 Field summary

serialVersionUID Generated Serial version UID.

6.1.2.3 Constructor summary

AverageHausdorff()

6.1.2.4 Method summary

distance(Instances, Instances)
distance(MIMLBag, MIMLBag)

6.1.2.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.

6.1.2.6 Constructors

- **AverageHausdorff**

  public AverageHausdorff()

6.1.2.7 Methods

- **distance**

  double distance(weka.core.Instances first,weka.core.Instances second) **throws** java.lang.Exception

  - **Description copied from IDistance** (6.1.1)
    Get the distance between two bags in the form of a set of `Instances` .
  - **Parameters**
    - `first` – First bag as instances.
    - `second` – Second Bag as Instances.
  - **Returns** – Distance between two bags.
  - **Throws**
    - `java.lang.Exception` – if occurred an error during distance calculation.

- **distance**

  double distance(miml.data.MIMLBag first,miml.data.MIMLBag second) **throws** java.lang.
  Exception

  - **Description copied from IDistance** (6.1.1)
    Get the distance between two `MIMLBag` (6.5.1).
  - **Parameters**
    - `first` – First bag.
    - `second` – Second bag.
  - **Returns** – Distance between two bags.
  - **Throws**
    - `java.lang.Exception` – if occurred an error during distance calculation,

6.1.3 Class MaximalHausdorff

Class that implements Maximal Hausdorff metric to measure the distance between 2 bags of a data set.

6.1.3.1 Declaration

public class MaximalHausdorff
extends java.lang.Object **implements** IDistance

6.1.3.2 Field summary

serialVersionUID Generated Serial version UID.

6.1.3.3 Constructor summary

MaximalHausdorff()

6.1.3.4 Method summary

distance(Instances, Instances)
distance(MIMLBag, MIMLBag)

6.1.3.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.

6.1.3.6 Constructors

- **MaximalHausdorff**

  public MaximalHausdorff()

6.1.3.7 Methods

- **distance**

  double distance(weka.core.Instances first,weka.core.Instances second) **throws** java.lang.Exception

  - **Description copied from IDistance** (6.1.1)
    Get the distance between two bags in the form of a set of `Instances` .
  - **Parameters**
    - `first` – First bag as instances.
    - `second` – Second Bag as Instances.
  - **Returns** – Distance between two bags.
  - **Throws**
    - `java.lang.Exception` – if occurred an error during distance calculation.

- **distance**

  double distance(miml.data.MIMLBag first,miml.data.MIMLBag second) **throws** java.lang.
  Exception

  - **Description copied from IDistance** (6.1.1)
    Get the distance between two `MIMLBag` (6.5.1).
  - **Parameters**
    - `first` – First bag.
    - `second` – Second bag.
  - **Returns** – Distance between two bags.
  - **Throws**
    - `java.lang.Exception` – if occurred an error during distance calculation,

6.1.4 Class MinimalHausdorff

Class that implements Minimal Hausdorff metric to measure the distance between 2 bags of a data set.

6.1.4.1 Declaration

public class MinimalHausdorff
 extends java.lang.Object **implements** IDistance

6.1.4.2 Field summary

serialVersionUID Generated Serial version UID.

6.1.4.3 Constructor summary

MinimalHausdorff()

6.1.4.4 Method summary

distance(Instances, Instances)
distance(MIMLBag, MIMLBag)

6.1.4.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.

6.1.4.6 Constructors

- **MinimalHausdorff**

  public MinimalHausdorff()

6.1.4.7 Methods

- **distance**

  double distance(weka.core.Instances first,weka.core.Instances second) **throws** java.lang.Exception

  – **Description copied from IDistance** (6.1.1)
    Get the distance between two bags in the form of a set of `Instances` .
  – **Parameters**
    * `first` – First bag as instances.
    * `second` – Second Bag as Instances.
  – **Returns** – Distance between two bags.
  – **Throws**
    * `java.lang.Exception` – if occurred an error during distance calculation.

- **distance**

  double distance(miml.data.MIMLBag first,miml.data.MIMLBag second) **throws** java.lang.
  Exception

  – **Description copied from IDistance** (6.1.1)
    Get the distance between two `MIMLBag` (6.5.1).
  – **Parameters**
    * `first` – First bag.
    * `second` – Second bag.
  – **Returns** – Distance between two bags.
  – **Throws**
    * `java.lang.Exception` – if occurred an error during distance calculation,

6.2 Package miml.evaluation

Package Contents *Page*

6.2.1 Interface IEvaluator

Interface for run and evaluate a experiment.

6.2.1.1 Declaration

public interface IEvaluator

6.2.1.2 All known subinterfaces

EvaluatorHoldout (6.2.3), EvaluatorCV (6.2.2)

6.2.1.3 All classes known to implement interface

EvaluatorHoldout (6.2.3), EvaluatorCV (6.2.2)

6.2.1.4 Method summary

getEvaluation() Gets the evaluation generated by the experiment.
runExperiment(IMIMLClassifier) Run an experiment.

6.2.1.5 Methods

- **getEvaluation**

  java.lang.Object getEvaluation()

  - **Description**
    Gets the evaluation generated by the experiment.
  - **Returns** – The evaluation.

- **runExperiment**

  void runExperiment(miml.classifiers.miml.IMIMLClassifier classifier) **throws** java.lang.Exception

  - **Description**
    Run an experiment.
  - **Parameters**
    - `classifier` – The classifier used in the experiment.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

6.2.2 Class EvaluatorCV

Class that allow evaluate an algorithm applying a cross-validation method.

6.2.2.1 Declaration

public class EvaluatorCV
extends java.lang.Object **implements** miml.core.IConfiguration, IEvaluator

6.2.2.2 Field summary

data The data used in the experiment.
multipleEvaluation The evaluation method used in cross-validation.
numFolds The number of folds.
seed The seed for the partition.
testTime Test time in milliseconds.
trainTime Train time in milliseconds.

6.2.2.3 Constructor summary

EvaluatorCV() No-argument constructor for xml configuration.
EvaluatorCV(MIMLInstances, int) Instantiates a new Holdout evaluator.

6.2.2.4 Method summary

configure(Configuration)
getAvgTestTime() Gets the average time of all folds in test.
getAvgTrainTime() Gets the average time of all folds in train.
getData() Gets the data used in the experiment.
getEvaluation()
getNumFolds() Gets the num folds used in the experiment.
getSeed() Gets the seed used in the experiment.
getStdTestTime() Gets the standard deviation time of all folds in test.
getStdTrainTime() Gets the standard deviation time of all folds in train.
getTestTime() Gets the time spent in testing in each fold.
getTrainTime() Gets the time spent in training in each fold.
meanArray(long[]) Calculate the mean of given array.
runExperiment(IMIMLClassifier)
setNumFolds(int) Sets the num folds used in the experiment.
setSeed(int) Sets the seed used in the experiment.
stdArray(long[]) Calculate the standard deviation of given array.

6.2.2.5 Fields

- `protected mulan.evaluation.MultipleEvaluation` **multipleEvaluation**
  – The evaluation method used in cross-validation.
- `protected miml.data.MIMLInstances` **data**
  – The data used in the experiment.
- `protected int` **numFolds**
  – The number of folds.
- `protected int` **seed**
  – The seed for the partition.

- `protected long[]` **trainTime**
  - Train time in milliseconds.
- `protected long[]` **testTime**
  - Test time in milliseconds.

6.2.2.6 Constructors

- **EvaluatorCV**

  public EvaluatorCV()

  - **Description**
    No-argument constructor for xml configuration.
- **EvaluatorCV**

  public EvaluatorCV(miml.data.MIMLInstances data,**int** numFolds)

  - **Description**
    Instantiates a new Holdout evaluator.
  - **Parameters**
    - `data` – The data used in the experiment.
    - `numFolds` – The number of folds used in the cross-validation.

6.2.2.7 Methods

- **configure**

  void configure(org.apache.commons.configuration2.Configuration configuration)

  - **Description copied from miml.core.IConfiguration** (6.7.1)
    Method to configure the class with the given configuration.
  - **Parameters**
    - `configuration` – Configuration used to configure the class.
- **getAvgTestTime**

  public double getAvgTestTime()

  - **Description**
    Gets the average time of all folds in test.
  - **Returns** – The average time of all folds.
- **getAvgTrainTime**

  public double getAvgTrainTime()

- **Description**
  Gets the average time of all folds in train.
- **Returns** – The average time of all folds.

- **getData**

  public miml.data.MIMLInstances getData()

  - **Description**
    Gets the data used in the experiment.
  - **Returns** – The data.

- **getEvaluation**

  java.lang.Object getEvaluation()

  - **Description copied from IEvaluator** (6.2.1)
    Gets the evaluation generated by the experiment.
  - **Returns** – The evaluation.

- **getNumFolds**

  public int getNumFolds()

  - **Description**
    Gets the num folds used in the experiment.
  - **Returns** – The num folds.

- **getSeed**

  public int getSeed()

  - **Description**
    Gets the seed used in the experiment.
  - **Returns** – The seed.

- **getStdTestTime**

  public double getStdTestTime()

  - **Description**
    Gets the standard deviation time of all folds in test.
  - **Returns** – The standard deviation time of all folds.

- **getStdTrainTime**

  public double getStdTrainTime()

  - **Description**
    Gets the standard deviation time of all folds in train.
  - **Returns** – The standard deviation time of all folds.

- **getTestTime**

  public long[] getTestTime()

  - **Description**
    Gets the time spent in testing in each fold.
  - **Returns** – The test time.

- **getTrainTime**

  public long[] getTrainTime()

  - **Description**
    Gets the time spent in training in each fold.
  - **Returns** – The train time.

- **meanArray**

  protected double meanArray(**long**[] array)

  - **Description**
    Calculate the mean of given array.
  - **Parameters**
    * `array` – The array with long values.
  - **Returns** – The mean of all array's values.

- **runExperiment**

  void runExperiment(miml.classifiers.miml.IMIMLClassifier classifier) **throws** java.lang.Exception

  - **Description copied from IEvaluator** (6.2.1)
    Run an experiment.
  - **Parameters**
    * `classifier` – The classifier used in the experiment.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

- **setNumFolds**

  public void setNumFolds(**int** numFolds)

  - **Description**
    Sets the num folds used in the experiment.
  - **Parameters**

    * `numFolds` – The new num folds.

- **setSeed**

  public void setSeed(**int** seed)

  - **Description**
    Sets the seed used in the experiment.
  - **Parameters**
    * `seed` – The new seed

- **stdArray**

  protected double stdArray(**long**[] array)

  - **Description**
    Calculate the standard deviation of given array.
  - **Parameters**
    * `array` – the array with long values.
  - **Returns** – The standard deviation of all array's values.

6.2.3 Class EvaluatorHoldout

Class that allow evaluate an algorithm applying a holdout method.

6.2.3.1 Declaration

public class EvaluatorHoldout
extends java.lang.Object **implements** miml.core.IConfiguration, IEvaluator

6.2.3.2 Field summary

evaluation The evaluation method used in holdout.
seed Seed for randomization
testData The test data used in the experiment.
testTime Test time in milliseconds.
trainData The data used in the experiment.
trainTime Train time in milliseconds.

6.2.3.3 Constructor summary

EvaluatorHoldout() No-argument constructor for xml configuration.
EvaluatorHoldout(MIMLInstances, double) Instantiates a new Holdout evaluator.
EvaluatorHoldout(MIMLInstances, MIMLInstances) Instantiates a new Holdout evaluator.

6.2.3.4 Method summary

configure(Configuration)
getData() Gets the data used in the experiment.
getEvaluation()
getSeed() Gets the seed used in the experiment.
getTestTime() Gets the time spent in testing.
getTrainTime() Gets the time spent in training.
runExperiment(IMIMLClassifier)
setSeed(int) Sets the seed used in the experiment.

6.2.3.5 Fields

- `protected mulan.evaluation.Evaluation` **evaluation**
  - The evaluation method used in holdout.
- `protected miml.data.MIMLInstances` **trainData**
  - The data used in the experiment.
- `protected miml.data.MIMLInstances` **testData**
  - The test data used in the experiment.
- `protected long` **trainTime**
  - Train time in milliseconds.
- `protected long` **testTime**
  - Test time in milliseconds.
- `protected int` **seed**
  - Seed for randomization

6.2.3.6 Constructors

- **EvaluatorHoldout**

  public EvaluatorHoldout()

  - **Description**
    No-argument constructor for xml configuration.

- **EvaluatorHoldout**

  public EvaluatorHoldout(miml.data.MIMLInstances mimlDataSet,**double** percentageTrain)
    throws java.lang.Exception

  - **Description**
    Instantiates a new Holdout evaluator.
  - **Parameters**
    - `mimlDataSet` – the dataset to be used
    - `percentageTrain` – the percentage of train

- **Throws**
  - `java.lang.Exception` – if occur an error during holdout experiment

- **EvaluatorHoldout**

  public EvaluatorHoldout(miml.data.MIMLInstances trainData,miml.data.MIMLInstances testData)

  - **Description**
    Instantiates a new Holdout evaluator.
  - **Parameters**
    - `trainData` – The train data used in the experiment.
    - `testData` – The test data used in the experiment.

6.2.3.7 Methods

- **configure**

  void configure(org.apache.commons.configuration2.Configuration configuration)

  - **Description copied from miml.core.IConfiguration** (6.7.1)
    Method to configure the class with the given configuration.
  - **Parameters**
    - `configuration` – Configuration used to configure the class.

- **getData**

  public miml.data.MIMLInstances getData()

  - **Description**
    Gets the data used in the experiment.
  - **Returns** – The data.

- **getEvaluation**

  java.lang.Object getEvaluation()

  - **Description copied from IEvaluator** (6.2.1)
    Gets the evaluation generated by the experiment.
  - **Returns** – The evaluation.

- **getSeed**

  public int getSeed()

  - **Description**
    Gets the seed used in the experiment.
  - **Returns** – The seed.

- **getTestTime**

  public long getTestTime()

  - **Description**
    Gets the time spent in testing.
  - **Returns** – The test time.

- **getTrainTime**

  public long getTrainTime()

  - **Description**
    Gets the time spent in training.
  - **Returns** – The train time.

- **runExperiment**

  void runExperiment(miml.classifiers.miml.IMIMLClassifier classifier) **throws** java.lang.Exception

  - **Description copied from IEvaluator** (6.2.1)
    Run an experiment.
  - **Parameters**
    * `classifier` – The classifier used in the experiment.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

- **setSeed**

  public void setSeed(**int** seed)

  - **Description**
    Sets the seed used in the experiment.
  - **Parameters**
    * `seed` – The new seed.

6.3 Package miml.classifiers.miml

Package Contents *Page*

6.3.1 Interface IMIMLClassifier

Common interface for MIML classifiers.

6.3.1.1 Declaration

public interface IMIMLClassifier
extends mulan.classifier.MultiLabelLearner, java.io.Serializable

6.3.1.2 All known subinterfaces

MIMLClassifier (6.3.2), MIMLBagging (6.6.1), MIMLClassifierToMI (6.9.2), MultiLabelKNN_MIMLWrapper (6.13.7), MLkNN_MIMLWrapper (6.13.6), MIMLkNN (6.13.5), IBLR_ML_MIMLWrapper (6.13.4), DM-LkNN_MIMLWrapper (6.13.3), BRkNN_MIMLWrapper (6.13.1), MIMLClassifierToML (6.14.1)

6.3.1.3 All classes known to implement interface

MIMLClassifier (6.3.2)

6.3.1.4 Method summary

build(MIMLInstances) Builds the learner model from specified `MIMLInstances` (6.5.2) data.
makeCopy()
makePrediction(Instance)
setDebug(boolean)

6.3.1.5 Methods

- **build**

  void build(miml.data.MIMLInstances trainingSet) **throws** java.lang.Exception

  - **Description**
    Builds the learner model from specified `MIMLInstances` (6.5.2) data.
  - **Parameters**
    - `trainingSet` – Set of training data, upon which the learner model should be built.
  - **Throws**
    - `java.lang.Exception` – If learner model was not created successfully.

- **makeCopy**

  mulan.classifier.MultiLabelLearner makeCopy() **throws** java.lang.Exception

- **makePrediction**

  mulan.classifier.MultiLabelOutput makePrediction(weka.core.Instance arg0) **throws** java.lang.Exception, mulan.classifier.InvalidDataException, mulan.classifier.ModelInitializationException

- **setDebug**

  void setDebug(**boolean** arg0)

6.3.2 Class MIMLClassifier

This java class is based on the mulan.data.Statistics.java class provided in the Mulan java framework for multi-label learning *Tsoumakas, G., Katakis, I., Vlahavas, I. (2010) "Mining Multi-label Data", Data Mining and Knowledge Discovery Handbook, O. Maimon, L. Rokach (Ed.), Springer, 2nd edition, 2010.* Our contribution is mainly related with providing a framework to work with MIML data.

6.3.2.1 Declaration

public abstract class MIMLClassifier
extends java.lang.Object **implements** miml.core.IConfiguration, IMIMLClassifier

6.3.2.2 All known subclasses

MIMLBagging (6.6.1), MIMLClassifierToMI (6.9.2), MultiLabelKNN_MIMLWrapper (6.13.7), MLkNN_MIMLWrapper (6.13.6), MIMLkNN (6.13.5), IBLR_ML_MIMLWrapper (6.13.4), DMLkNN_MIMLWrapper (6.13.3), BRkNN_MIMLWrapper (6.13.1), MIMLClassifierToML (6.14.1)

6.3.2.3 Field summary

featureIndices An array containing the indexes of the feature attributes within the `Instances` object of the training data in increasing order.
isDebug Whether debugging is on/off.
isModelInitialized Boolean that indicate if the model has been initialized.
labelIndices An array containing the indexes of the label attributes within the `Instances` object of the training data in increasing order.
labelNames An array containing the names of the label attributes within the `Instances` object of the training data in increasing order.
numLabels The number of labels the learner can handle.
serialVersionUID Generated Serial version UID.

6.3.2.4 Constructor summary

MIMLClassifier()

6.3.2.5 Method summary

build(MIMLInstances)
build(MultiLabelInstances)
buildInternal(MIMLInstances) Learner specific implementation of building the model from `MultiLabelInstances` training data set.
debug(String) Writes the debug message string to the console output if debug for the learner is enabled.
getDebug() Get whether debugging is turned on.
isModelInitialized() Gets whether learner's model is initialized by `build(MultiLabelInstances)` .
isUpdatable()
makeCopy()
makePrediction(Instance)
makePredictionInternal(MIMLBag) Learner specific implementation for predicting on specified data based on trained model.
setDebug(boolean)

6.3.2.6 Fields

- `private static final long` **serialVersionUID**
  – Generated Serial version UID.
- `protected boolean` **isModelInitialized**
  – Boolean that indicate if the model has been initialized.
- `protected int` **numLabels**
  – The number of labels the learner can handle. The number of labels is determined from the training data when learner is build.
- `protected int[]` **labelIndices**
  – An array containing the indexes of the label attributes within the `Instances` object of the training data in increasing order. The same order will be followed in the arrays of predictions given by each learner in the `MultiLabelOutput` object.
- `protected java.lang.String[]` **labelNames**
  – An array containing the names of the label attributes within the `Instances` object of the training data in increasing order. The same order will be followed in the arrays of predictions given by each learner in the `MultiLabelOutput` object.
- `protected int[]` **featureIndices**
  – An array containing the indexes of the feature attributes within the `Instances` object of the training data in increasing order.
- `private boolean` **isDebug**
  – Whether debugging is on/off.

6.3.2.7 Constructors

- **MIMLClassifier**

public MIMLClassifier()

6.3.2.8 Methods

- **build**

  void build(miml.data.MIMLInstances trainingSet) **throws** java.lang.Exception

  - **Description copied from IMIMLClassifier** (6.3.1)
    Builds the learner model from specified `MIMLInstances` (6.5.2) data.
  - **Parameters**
    * `trainingSet` – Set of training data, upon which the learner model should be built.
  - **Throws**
    * `java.lang.Exception` – If learner model was not created successfully.

- **build**

  public final void build(mulan.data.MultiLabelInstances trainingSet) **throws** java.lang.Exception

- **buildInternal**

  protected abstract void buildInternal(miml.data.MIMLInstances trainingSet) **throws** java.lang.Exception

  - **Description**
    Learner specific implementation of building the model from `MultiLabelInstances` training data set. This method is called from `build(MultiLabelInstances)` method, where behavior common across all learners is applied.
  - **Parameters**
    * `trainingSet` – The training data set.
  - **Throws**
    * `java.lang.Exception` – if learner model was not created successfully.

- **debug**

  protected void debug(java.lang.String msg)

  - **Description**
    Writes the debug message string to the console output if debug for the learner is enabled.
  - **Parameters**
    * `msg` – The debug message

- **getDebug**

  public boolean getDebug()

  - **Description**
    Get whether debugging is turned on.
  - **Returns** – `True` if debugging output is on

- **isModelInitialized**

  protected boolean isModelInitialized()

  – **Description**

    Gets whether learner's model is initialized by `build(MultiLabelInstances)` . This is used to check if `makePrediction(Instance)` can be processed.

  – **Returns** – `true` if the model has been initialized.

- **isUpdatable**

  public boolean isUpdatable()

- **makeCopy**

  mulan.classifier.MultiLabelLearner makeCopy() **throws** java.lang.Exception

- **makePrediction**

  mulan.classifier.MultiLabelOutput makePrediction(weka.core.Instance arg0) **throws** java.lang.Exception, mulan.classifier.InvalidDataException, mulan.classifier.ModelInitializationException

- **makePredictionInternal**

  protected abstract mulan.classifier.MultiLabelOutput makePredictionInternal(miml.data.MIMLBag instance) **throws** java.lang.Exception, mulan.classifier.InvalidDataException

  – **Description**

    Learner specific implementation for predicting on specified data based on trained model. This method is called from `makePrediction(Instance)` which guards for model initialization and apply common handling/behavior.

  – **Parameters**

    * `instance` – The data instance to predict on.

  – **Returns** – The output of the learner for the given instance.

  – **Throws**

    * `java.lang.Exception` – If an error occurs while making the prediction.
    * `mulan.classifier.InvalidDataException` – If specified instance data is invalid and can not be processed by the learner.

- **setDebug**

  void setDebug(**boolean** arg0)

6.4 Package miml.data.statistics

Package Contents *Page*

6.4.1 Class MIMLStatistics

Class with methods to obtain information about a MIML dataset. This java class is based on MLStatistic and MILStatistic.

6.4.1.1 Declaration

public class MIMLStatistics
 extends java.lang.Object

6.4.1.2 Field summary

dataSet A MIML data set
milstatistics Class with methods to obtain information about a MI dataset.
mlstatistics Class with methods to obtain information about a ML dataset.

6.4.1.3 Constructor summary

MIMLStatistics(MIMLInstances) Constructor.

6.4.1.4 Method summary

averageIR(double[]) Computes the average of any IR vector.
averageSkew(HashMap) Computes the average labelSkew.
calculateCooncurrence(MIMLInstances) This method calculates a matrix with the cooncurrences of pairs of labels.
calculatePhiChi2(MIMLInstances) Calculates Phi and Chi-square correlation matrix.
cardinality() Computes the Cardinality as the average number of labels per pattern.
cooncurrenceToCSV() Returns cooCurrenceMatrix in CSV representation.
cooncurrenceToString() Returns cooCurrenceMatrix in textual representation.
correlationsToCSV(double[][]) Returns Phi correlations in CSV representation.
correlationsToString(double[][]) Returns Phi correlations in textual representation.
density() Computes the density as the cardinality/numLabels.
distributionBagsToCSV() Returns distributionBags in CSV representation.

distributionBagsToCSV(HashMap) Returns labelSkew in CSV representation.
distributionBagsToString() Returns distributionBags in textual representation.
distributionBagsToString(HashMap) Returns labelSkew in textual representation.
getChi2() Gets the Chi2 correlation matrix.
getDataSet() Returns the dataset used to calculate the statistics.
getPhi() Gets the Phi correlation matrix.
getPhiHistogram() Calculates a histogram of Phi correlations.
innerClassIR() Computes the innerClassIR for each label as negativePatterns/positivePatterns.
interClassIR() Computes the interClassIR for each label positiveExamplesOfMajorityLabel/positivePatternsLabel.
labelCombCount() Returns the HashMap containing the distinct labelsets and their frequencies.
labelSetFrequency(LabelSet) Returns the frequency of a label set in the dataset.
labelSets() Returns a set with the distinct label sets of the dataset.
labelSkew() Computes the IR for each labelSet as (patterns of majorityLabelSet)/(patterns of the labelSet).
pMax() Returns pMax, the proportion of examples associated with the most frequently occurring labelset.
printPhiDiagram(double) This method prints data, useful for the visualization of Phi per dataset.
priors() Returns the prior probabilities of the labels.
pUnique() Returns proportion of unique label combinations (pPunique) value defined as the proportion of labelsets which are unique across the total number of examples.
setDataSet(MIMLInstances) Set the dataset used.
skewRatio() Computes the skewRatio as peak/base.
toCSV() Returns statistics in CSV representation.
topPhiCorrelatedLabels(int, int) Returns the indices of the labels that have the strongest Phi correlation with the label which is given as a parameter.
toString() Returns statistics in textual representation.
uncorrelatedLabels(int, double) Returns the indices of the labels whose Phi coefficient values lie between -bound <= phi <= bound.
varianceIR(double[]) Computes the variance of any IR vector.

6.4.1.5 **Fields**

- `miml.data.MIMLInstances` **dataSet**
  - A MIML data set
- `protected MIStatistics` **milstatistics**
  - Class with methods to obtain information about a MI dataset.
  - See also
    - `MIStatistics` (6.4.2)
- `protected MLStatistics` **mlstatistics**
  - Class with methods to obtain information about a ML dataset.
  - See also
    - `MLStatistics` (6.4.3)

6.4.1.6 Constructors

- **MIMLStatistics**

  public MIMLStatistics(miml.data.MIMLInstances dataSet)

  - **Description**
    Constructor.
  - **Parameters**
    * `dataSet` – A MIML data set.

6.4.1.7 Methods

- **averageIR**

  public double averageIR(**double**[] IR)

  - **Description**
    Computes the average of any IR vector.
  - **Parameters**
    * `IR` – An IR vector previously computed
  - **Returns** – double

- **averageSkew**

  public double averageSkew(java.util.HashMap skew)

  - **Description**
    Computes the average labelSkew.
  - **Parameters**
    * `skew` – The IR for each labelSet previously computed.
  - **Returns** – Average labelSkew.

- **calculateCooncurrence**

  public double[][] calculateCooncurrence(miml.data.MIMLInstances mlDataSet)

  - **Description**
    This method calculates a matrix with the cooncurrences of pairs of labels. It requires the method calculateStats to be previously called.
  - **Parameters**
    * `mlDataSet` – A multi-label dataset.
  - **Returns** – A cooncurrences matrix of pairs of labels.

- **calculatePhiChi2**

  public void calculatePhiChi2(miml.data.MIMLInstances dataSet) **throws** java.lang.Exception

  - **Description**
    Calculates Phi and Chi-square correlation matrix.
  - **Parameters**
    * `dataSet` – A multi-label dataset.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

- **cardinality**

  public double cardinality()

  - **Description**
    Computes the Cardinality as the average number of labels per pattern. It requires the method calculateStats to be previously called.
  - **Returns** – double

- **cooncurrenceToCSV**

  public java.lang.String cooncurrenceToCSV()

  - **Description**
    Returns cooCurrenceMatrix in CSV representation. It requires the method calculate-Cooncurrence to be previously called.
  - **Returns** – CooCurrenceMatrix in CSV representation.

- **cooncurrenceToString**

  public java.lang.String cooncurrenceToString()

  - **Description**
    Returns cooCurrenceMatrix in textual representation. It requires the method calculate-Cooncurrence to be previously called.
  - **Returns** – CooCurrenceMatrix in textual representation.

- **correlationsToCSV**

  public java.lang.String correlationsToCSV(**double**[][] matrix)

  - **Description**
    Returns Phi correlations in CSV representation. It requires the method calculatePhiChi2 to be previously called.
  - **Parameters**
    * `matrix` – Matrix with Phi correlations.
  - **Returns** – Phi correlations in CSV representation.

- **correlationsToString**

  public java.lang.String correlationsToString(**double**[][] matrix)

- **Description**
  Returns Phi correlations in textual representation. It requires the method calculatePhiChi2 to be previously called.
- **Parameters**
  * `matrix` – Matrix with Phi correlations.
- **Returns** – Phi correlations in textual representation.

- **density**

  public double density()

  - **Description**
    Computes the density as the cardinality/numLabels. It the method calculateStats to be previously called.
  - **Returns** – density.

- **distributionBagsToCSV**

  protected java.lang.String distributionBagsToCSV()

  - **Description**
    Returns distributionBags in CSV representation.
  - **Returns** – CSV with bags distribution.

- **distributionBagsToCSV**

  protected java.lang.String distributionBagsToCSV(java.util.HashMap skew)

  - **Description**
    Returns labelSkew in CSV representation.
  - **Parameters**
    * `skew` – The IR for each labelSet previously computed.
  - **Returns** – LabelSkew in CSV representation.

- **distributionBagsToString**

  protected java.lang.String distributionBagsToString()

  - **Description**
    Returns distributionBags in textual representation.
  - **Returns** – String with bags distribution.

- **distributionBagsToString**

  protected java.lang.String distributionBagsToString(java.util.HashMap skew)

  - **Description**
    Returns labelSkew in textual representation.

  - **Parameters**
    * `skew` – The IR for each labelSet previously computed.
  - **Returns** – LabelSkew in textual representation.

- **getChi2**

  public double[][] getChi2()

  - **Description**
    Gets the Chi2 correlation matrix. It requires the method calculatePhiChi2 to be previously called.
  - **Returns** – chi2.

- **getDataSet**

  public miml.data.MIMLInstances getDataSet()

  - **Description**
    Returns the dataset used to calculate the statistics.
  - **Returns** – A MIML data set.

- **getPhi**

  public double[][] getPhi()

  - **Description**
    Gets the Phi correlation matrix. It requires the method calculatePhiChi2 to be previously called.
  - **Returns** – phi.

- **getPhiHistogram**

  public double[] getPhiHistogram()

  - **Description**
    Calculates a histogram of Phi correlations. It requires the method calculatePhi to be previously called.
  - **Returns** – An array with Phi correlations.

- **innerClassIR**

  public double[] innerClassIR()

  - **Description**
    Computes the innerClassIR for each label as negativePatterns/positivePatterns. It requires the method calculateStats to be previously called.
  - **Returns** – An IR for each label: negativePatterns/positivePatterns.

- **interClassIR**

public double[] interClassIR()

- **Description**
  Computes the interClassIR for each label positiveExamplesOfMajorityLabel/positivePatternsLabel. It requires the method calculateStats to be previously called.
- **Returns** – An IR between binary labels: maxPositiveClassExamples/positiveExamplesLabel.

• **labelCombCount**

public java.util.HashMap labelCombCount()

- **Description**
  Returns the HashMap containing the distinct labelsets and their frequencies. It requires the method calculateStats to be previously called.
- **Returns** – HashMap with distinct labelsest and their frequencies.

• **labelSetFrequency**

public int labelSetFrequency(mulan.data.LabelSet x)

- **Description**
  Returns the frequency of a label set in the dataset. It requires the method calculateStats to be previously called.
- **Parameters**
  - `x` – A labelset.
- **Returns** – The frequency of the given labelset.

• **labelSets**

public java.util.Set labelSets()

- **Description**
  Returns a set with the distinct label sets of the dataset. It requires the method calculateStats to be previously called.
- **Returns** – Set of distinct label sets.

• **labelSkew**

public java.util.HashMap labelSkew()

- **Description**
  Computes the IR for each labelSet as (patterns of majorityLabelSet)/(patterns of the labelSet). It requires the method calculateStats to be previously called.
- **Returns** – HashMap<LabelSet, Double>

• **pMax**

public double pMax()

  - **Description**

    Returns pMax, the proportion of examples associated with the most frequently occurring labelset. It requires the method calculateStats to be previously called. More information in Jesse Read. 2010. Scalable Multi-label Classification. Ph.D. Dissertation. University of Waikato.

  - **Returns** – pMax.

- **printPhiDiagram**

  public void printPhiDiagram(**double** step)

  - **Description**

    This method prints data, useful for the visualization of Phi per dataset. It prints int(1/step) + 1 pairs of values. The first value of each pair is the phi value and the second is the average number of labels that correlate to the rest of the labels with correlation higher than the specified Phi value. It requires the method calculatePhi to be previously called.

  - **Parameters**

    ∗ `step` – The Ohi value increment step.

- **priors**

  public double[] priors()

  - **Description**

    Returns the prior probabilities of the labels. It requires the method calculateStats to be previously called.

  - **Returns** – An array of double with prior probabilities of labels.

- **pUnique**

  public double pUnique()

  - **Description**

    Returns proportion of unique label combinations (pPunique) value defined as the proportion of labelsets which are unique across the total number of examples. It requires the method calculateStats to be previously called. More information in Jesse Read. 2010. Scalable Multi-label Classification. Ph.D. Dissertation. University of Waikato.

  - **Returns** – Proportion of unique label combinations.

- **setDataSet**

  public void setDataSet(miml.data.MIMLInstances dataSet)

  - **Description**

    Set the dataset used.

  - **Parameters**

    ∗ `dataSet` – A MIML data set.

- **skewRatio**

  public double skewRatio()

  - **Description**
    Computes the skewRatio as peak/base. It requires the method calculateStats to be previously called.
  - **Returns** – SkewRatio as peak/base.

- **toCSV**

  public java.lang.String toCSV()

  - **Description**
    Returns statistics in CSV representation. It requires the method calculateStats to be previously called.
  - **Returns** – Statistics in CSV representation.

- **topPhiCorrelatedLabels**

  public int[] topPhiCorrelatedLabels(**int** labelIndex,**int** k)

  - **Description**
    Returns the indices of the labels that have the strongest Phi correlation with the label which is given as a parameter. The second parameter is the number of labels that will be returned. It requires the method calculatePhi to be previously called.
  - **Parameters**
    - `labelIndex` – The label index.
    - `k` – The number of labels that will be returned. The number of labels that will be returned.
  - **Returns** – The indices of the k most correlated labels.

- **toString**

  public java.lang.String toString()

  - **Description**
    Returns statistics in textual representation. It requires the method calculateStats to be previously called.
  - **Returns** – Statistics in textual representation.

- **uncorrelatedLabels**

  public int[] uncorrelatedLabels(**int** labelIndex,**double** bound)

  - **Description**
    Returns the indices of the labels whose Phi coefficient values lie between -bound <= phi <= bound. It requires the method calculatePhi to be previously called.
  - **Parameters**

    * `labelIndex` – The label index.
    * `bound` – The bound.
  - **Returns** – The indices of the labels whose Phi coefficient values lie between -bound <= phi <= bound.

- **varianceIR**

  public double varianceIR(**double**[] IR)

  - **Description**
    Computes the variance of any IR vector.
  - **Parameters**
    * `IR` – An IR vector previously computed.
  - **Returns** – Variance of any IR vector.

6.4.2 Class MIStatistics

Class with methods to obtain information about a MI dataset such as the number of attributes per bag, the average number of instances per bag, and the distribution of number of instances per bag...

6.4.2.1 Declaration

public class MIStatistics
extends java.lang.Object

6.4.2.2 Field summary

attributesPerBag The number of attributes per bag.
avgInstancesPerBag The average number of instances per bag.
dataSet Instances dataset
distributionBags The distribution of number of instances per bag.
maxInstancesPerBag The maximum number of instances per bag.
minInstancesPerBag The minimum number of instances per bag.
numBags The number of bags.
totalInstances The total of instances.

6.4.2.3 Constructor summary

MIStatistics(Instances)

6.4.2.4 Method summary

calculateStats() Calculates various MIML statistics, such as instancesPerBag and attributesPerBag.
distributionBagsToCSV() Returns distributionBags in CSV representation.
distributionBagsToString() Returns distributionBags in textual representation.
toCSV() Returns statistics in CSV representation.
toString() Returns statistics in textual representation.

6.4.2.5 Fields

- `int` **minInstancesPerBag**
  - The minimum number of instances per bag.
- `int` **maxInstancesPerBag**
  - The maximum number of instances per bag.
- `double` **avgInstancesPerBag**
  - The average number of instances per bag.
- `int` **attributesPerBag**
  - The number of attributes per bag.
- `int` **numBags**
  - The number of bags.
- `int` **totalInstances**
  - The total of instances.
- `java.util.HashMap` **distributionBags**
  - The distribution of number of instances per bag.
- `weka.core.Instances` **dataSet**
  - Instances dataset

6.4.2.6 Constructors

- **MIStatistics**

  public MIStatistics(weka.core.Instances dataSet)

6.4.2.7 Methods

- **calculateStats**

  protected void calculateStats()

  - **Description**
    Calculates various MIML statistics, such as instancesPerBag and attributesPerBag.
- **distributionBagsToCSV**

  protected java.lang.String distributionBagsToCSV()

  - **Description**
    Returns distributionBags in CSV representation.
  - **Returns** – DistributionBags in CSV representation.

- **distributionBagsToString**

  protected java.lang.String distributionBagsToString()

  - **Description**
    Returns distributionBags in textual representation.
  - **Returns** – DistributionBags in textual representation.

- **toCSV**

  public java.lang.String toCSV()

  - **Description**
    Returns statistics in CSV representation.
  - **Returns** – Statistics in CSV representation.

- **toString**

  public java.lang.String toString()

  - **Description**
    Returns statistics in textual representation.
  - **Returns** – Statistics in textual representation.

6.4.3 Class MLStatistics

Class with methods to obtain information about a ML dataset. This java class is based on the mulan.data.Statistics.java class provided in the Mulan java framework for multi-label learning Tsoumakas, G., Katakis, I., Vlahavas, I. (2010) "Mining Multi-label Data", Data Mining and Knowledge Discovery Handbook, O. Maimon, L. Rokach (Ed.), Springer, 2nd edition, 2010. Our contribution is mainly related with methods to measure the degree of imbalance and a fixed bug in the method printPhiDiagram.

6.4.3.1 Declaration

public class MLStatistics
 extends java.lang.Object

6.4.3.2 Field summary

base The lowest labelSet count.
chi2 Chi square matrix values where 0 = complete independence.
cooncurrenceMatrix Cooncurrence matrix.
distributionLabelsPerExample The number of examples having 0, 1, 2,... , numLabel labels.
labelCombinations LabelSets in the dataset.
maxCount Number of labelSets with the peak value.
mlDataSet Multi label dataset
numAttributes The number of attributes.

numExamples The number of examples.
numLabels The number of labels.
numNominal The number of nominal predictive attributes.
numNumeric The number of numeric attributes.
nUnique Number of labelSets with only one pattern.
peak The highest labelSet count.
phi Phi matrix values in [-1,1] where -1 = inverse relation, 0 = no relation, 1 = direct relation.
positiveExamplesPerLabel The number of positive examples per label.

6.4.3.3 Constructor summary

MLStatistics(MultiLabelInstances) Constructor.

6.4.3.4 Method summary

averageIR(double[]) Computes the average of any IR vector.
averageSkew(HashMap) Computes the average labelSkew.
calculateCooncurrence(MultiLabelInstances) This method calculates a matrix with the cooncurrences of pairs of labels.
calculatePhiChi2(MultiLabelInstances) Calculates Phi and Chi-square correlation matrix.
calculateStats() Calculates various ML statistics.
cardinality() Computes the Cardinality as the average number of labels per pattern.
cooncurrenceToCSV() Returns cooCurrenceMatrix in CSV representation.
cooncurrenceToString() Returns cooCurrenceMatrix in textual representation.
correlationsToCSV(double[][]) Returns Phi correlations in CSV representation.
correlationsToString(double[][]) Returns Phi correlations in textual representation.
density() Computes the density as the cardinality/numLabels.
distributionBagsToCSV(HashMap) Returns labelSkew in CSV representation.
distributionBagsToString(HashMap) Returns labelSkew in textual representation.
getChi2() Gets the Chi2 correlation matrix.
getPhi() Gets the Phi correlation matrix.
getPhiHistogram() Calculates a histogram of Phi correlations.
innerClassIR() Computes the innerClassIR for each label as negativePatterns/positivePatterns.
interClassIR() Computes the interClassIR for each label positiveExamplesOfMajorityLabel/positivePatternsLabel.
labelCombCount() Returns the HashMap containing the distinct labelsets and their frequencies.
labelSetFrequency(LabelSet) Returns the frequency of a label set in the dataset.
labelSets() Returns a set with the distinct label sets of the dataset.
labelSkew() Computes the IR for each labelSet as (patterns of majorityLabelSet)/(patterns of the labelSet).
pMax() Returns pMax, the proportion of examples associated with the most frequently occurring labelset.
printPhiDiagram(double) This method prints data, useful for the visualization of Phi per dataset.
priors() Returns the prior probabilities of the labels.
pUnique() Returns proportion of unique label combinations (pPunique) value defined as the proportion of labelsets which are unique across the total number of examples.
skewRatio() Computes the skewRatio as peak/base.

toCSV() Returns statistics in CSV representation.
topPhiCorrelatedLabels(int, int) Returns the indices of the labels that have the strongest Phi correlation with the label which is given as a parameter.
toString() Returns statistics in textual representation.
uncorrelatedLabels(int, double) Returns the indices of the labels whose Phi coefficient values lie between -bound <= phi <= bound.
varianceIR(double[]) Computes the variance of any IR vector.

6.4.3.5 Fields

- `protected int` **numLabels**
  - The number of labels.
- `protected int` **numExamples**
  - The number of examples.
- `protected int` **numAttributes**
  - The number of attributes.
- `protected int` **numNominal**
  - The number of nominal predictive attributes.
- `protected int` **numNumeric**
  - The number of numeric attributes.
- `protected int[]` **positiveExamplesPerLabel**
  - The number of positive examples per label.
- `protected int[]` **distributionLabelsPerExample**
  - The number of examples having 0, 1, 2,... , numLabel labels.
- `protected java.util.HashMap` **labelCombinations**
  - LabelSets in the dataset.
- `protected int` **peak**
  - The highest labelSet count.
- `protected int` **base**
  - The lowest labelSet count.
- `protected int` **nUnique**
  - Number of labelSets with only one pattern.
- `protected int` **maxCount**
  - Number of labelSets with the peak value.
- `double[][]` **cooncurrenceMatrix**
  - Cooncurrence matrix.
- `double[][]` **phi**
  - Phi matrix values in [-1,1] where -1 = inverse relation, 0 = no relation, 1 = direct relation.
- `double[][]` **chi2**

  - Chi square matrix values where 0 = complete independence. Values larger than 6.63 show label dependence at 0.01 level of significance (99%). Values larger than 3.84 show label dependence at 0.05 level of significance (95%).

- `private mulan.data.MultiLabelInstances` **mlDataSet**
  - Multi label dataset

6.4.3.6 Constructors

- **MLStatistics**

  public MLStatistics(mulan.data.MultiLabelInstances mlDataSet)

  - **Description**
    Constructor.
  - **Parameters**
    * `mlDataSet` – MultiLabel dataset.

6.4.3.7 Methods

- **averageIR**

  public double averageIR(**double**[] IR)

  - **Description**
    Computes the average of any IR vector.
  - **Parameters**
    * `IR` – An IR vector previously computed
  - **Returns** – double

- **averageSkew**

  public double averageSkew(java.util.HashMap skew)

  - **Description**
    Computes the average labelSkew.
  - **Parameters**
    * `skew` – The IR for each labelSet previously computed.
  - **Returns** – double

- **calculateCooncurrence**

  public double[][] calculateCooncurrence(mulan.data.MultiLabelInstances mlDataSet)

  - **Description**
    This method calculates a matrix with the cooncurrences of pairs of labels. It requires the method calculateStats to be previously called.

  - **Parameters**
    * `mlDataSet` – A multi-label dataset.
  - **Returns** – A cooncurrences matrix of pairs of labels.

- **calculatePhiChi2**

  public void calculatePhiChi2(mulan.data.MultiLabelInstances dataSet) **throws** java.lang.Exception

  - **Description**
    Calculates Phi and Chi-square correlation matrix.
  - **Parameters**
    * `dataSet` – A multi-label dataset.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

- **calculateStats**

  protected void calculateStats()

  - **Description**
    Calculates various ML statistics.

- **cardinality**

  public double cardinality()

  - **Description**
    Computes the Cardinality as the average number of labels per pattern. It requires the method calculateStats to be previously called.
  - **Returns** – double

- **cooncurrenceToCSV**

  public java.lang.String cooncurrenceToCSV()

  - **Description**
    Returns cooCurrenceMatrix in CSV representation. It requires the method calculate-Cooncurrence to be previously called.
  - **Returns** – string

- **cooncurrenceToString**

  public java.lang.String cooncurrenceToString()

  - **Description**
    Returns cooCurrenceMatrix in textual representation. It requires the method calculate-Cooncurrence to be previously called.
  - **Returns** – string

- **correlationsToCSV**

  public java.lang.String correlationsToCSV(**double**[][] matrix)

  - **Description**
    Returns Phi correlations in CSV representation. It requires the method calculatePhiChi2 to be previously called.
  - **Parameters**
    * `matrix` – Matrix with Phi correlations.
  - **Returns** – String

- **correlationsToString**

  public java.lang.String correlationsToString(**double**[][] matrix)

  - **Description**
    Returns Phi correlations in textual representation. It requires the method calculatePhiChi2 to be previously called.
  - **Parameters**
    * `matrix` – Matrix with Phi correlations.
  - **Returns** – string

- **density**

  public double density()

  - **Description**
    Computes the density as the cardinality/numLabels. It the method calculateStats to be previously called.
  - **Returns** – double

- **distributionBagsToCSV**

  protected java.lang.String distributionBagsToCSV(java.util.HashMap skew)

  - **Description**
    Returns labelSkew in CSV representation.
  - **Parameters**
    * `skew` – The IR for each labelSet previously computed.
  - **Returns** – string

- **distributionBagsToString**

  protected java.lang.String distributionBagsToString(java.util.HashMap skew)

  - **Description**
    Returns labelSkew in textual representation.
  - **Parameters**

    * `skew` – The IR for each labelSet previously computed.
  - **Returns** – string

- **getChi2**

  public double[][] getChi2()

  - **Description**
    Gets the Chi2 correlation matrix. It requires the method calculatePhiChi2 to be previously called.
  - **Returns** – chi2

- **getPhi**

  public double[][] getPhi()

  - **Description**
    Gets the Phi correlation matrix. It requires the method calculatePhiChi2 to be previously called.
  - **Returns** – phi

- **getPhiHistogram**

  public double[] getPhiHistogram()

  - **Description**
    Calculates a histogram of Phi correlations. It requires the method calculatePhi to be previously called.
  - **Returns** – An array with Phi correlations.

- **innerClassIR**

  public double[] innerClassIR()

  - **Description**
    Computes the innerClassIR for each label as negativePatterns/positivePatterns. It requires the method calculateStats to be previously called.
  - **Returns** – An IR for each label: negativePatterns/positivePatterns.

- **interClassIR**

  public double[] interClassIR()

  - **Description**
    Computes the interClassIR for each label positiveExamplesOfMajorityLabel/positivePatternsLabel. It requires the method calculateStats to be previously called.
  - **Returns** – An IR between binary labels: maxPositiveClassExamples/positiveExamplesLabel.

- **labelCombCount**

  public java.util.HashMap labelCombCount()

  - **Description**
    Returns the HashMap containing the distinct labelsets and their frequencies. It requires the method calculateStats to be previously called.
  - **Returns** – HashMap with distinct labelsest and their frequencies.

- **labelSetFrequency**

  public int labelSetFrequency(mulan.data.LabelSet x)

  - **Description**
    Returns the frequency of a label set in the dataset. It requires the method calculateStats to be previously called.
  - **Parameters**
    - `x` – A labelset.
  - **Returns** – The frequency of the given labelset.

- **labelSets**

  public java.util.Set labelSets()

  - **Description**
    Returns a set with the distinct label sets of the dataset. It requires the method calculateStats to be previously called.
  - **Returns** – Set of distinct label sets.

- **labelSkew**

  public java.util.HashMap labelSkew()

  - **Description**
    Computes the IR for each labelSet as (patterns of majorityLabelSet)/(patterns of the labelSet). It requires the method calculateStats to be previously called.
  - **Returns** – HashMap<LabelSet, Double>

- **pMax**

  public double pMax()

  - **Description**
    Returns pMax, the proportion of examples associated with the most frequently occurring labelset. It requires the method calculateStats to be previously called. More information in Jesse Read. 2010. Scalable Multi-label Classification. Ph.D. Dissertation. University of Waikato.
  - **Returns** – double

- **printPhiDiagram**

  public void printPhiDiagram(**double** step)

  - **Description**

    This method prints data, useful for the visualization of Phi per dataset. It prints int(1/step) + 1 pairs of values. The first value of each pair is the phi value and the second is the average number of labels that correlate to the rest of the labels with correlation higher than the specified Phi value. It requires the method calculatePhi to be previously called.
  - **Parameters**
    - `step` – The Ohi value increment step.

- **priors**

  public double[] priors()

  - **Description**

    Returns the prior probabilities of the labels. It requires the method calculateStats to be previously called.
  - **Returns** – An array of double with prior probabilities of labels.

- **pUnique**

  public double pUnique()

  - **Description**

    Returns proportion of unique label combinations (pPunique) value defined as the proportion of labelsets which are unique across the total number of examples. It requires the method calculateStats to be previously called. More information in Jesse Read. 2010. Scalable Multi-label Classification. Ph.D. Dissertation. University of Waikato.
  - **Returns** – double

- **skewRatio**

  public double skewRatio()

  - **Description**

    Computes the skewRatio as peak/base. It requires the method calculateStats to be previously called.
  - **Returns** – double

- **toCSV**

  public java.lang.String toCSV()

  - **Description**

    Returns statistics in CSV representation. It requires the method calculateStats to be previously called.

  - **Returns** – string

- **topPhiCorrelatedLabels**

  public int[] topPhiCorrelatedLabels(**int** labelIndex,**int** k)

  - **Description**
    Returns the indices of the labels that have the strongest Phi correlation with the label which is given as a parameter. The second parameter is the number of labels that will be returned. It requires the method calculatePhi to be previously called.
  - **Parameters**
    * `labelIndex` – The label index.
    * `k` – The number of labels that will be returned. The number of labels that will be returned.
  - **Returns** – The indices of the k most correlated labels.

- **toString**

  public java.lang.String toString()

  - **Description**
    Returns statistics in textual representation. It requires the method calculateStats to be previously called.
  - **Returns** – string

- **uncorrelatedLabels**

  public int[] uncorrelatedLabels(**int** labelIndex,**double** bound)

  - **Description**
    Returns the indices of the labels whose Phi coefficient values lie between -bound <= phi <= bound. It requires the method calculatePhi to be previously called.
  - **Parameters**
    * `labelIndex` – The label index.
    * `bound` – The bound.
  - **Returns** – The indices of the labels whose Phi coefficient values lie between -bound <= phi <= bound.

- **varianceIR**

  public double varianceIR(**double**[] IR)

  - **Description**
    Computes the variance of any IR vector.
  - **Parameters**
    * `IR` – An IR vector previously computed.
  - **Returns** – double.

6.5 Package miml.data

Package Contents *Page*

6.5.1 Class MIMLBag

Class inheriting from DenseInstance to represent a MIML bag.

6.5.1.1 Declaration

public class MIMLBag
extends weka.core.DenseInstance **implements** weka.core.Instance

6.5.1.2 Field summary

serialVersionUID Generated Serial version UID.

6.5.1.3 Constructor summary

MIMLBag(Instance) Constructor.

6.5.1.4 Method summary

getBagAsInstances() Gets a bag in the form of a set of instances considering just the relational information.
getInstance(int) Returns an instance of the Bag with index bagIndex.
getNumAttributesInABag() Gets the number of attributes of in the relational attribute of a Bag.
getNumAttributesWithRelational() Gets the total number of attributes of the Bag.
getNumInstances() Gets the number of instances of the Bag.
setValue(int, int, double) Sets the value of attrIndex attribute of the the instanceIndex to a certain value.

6.5.1.5 Fields

- `private static final long` **serialVersionUID**
  – Generated Serial version UID.

6.5.1.6 Constructors

- **MIMLBag**

  public MIMLBag(weka.core.Instance instance) **throws** java.lang.Exception

  - **Description**
    Constructor.
  - **Parameters**
    - `instance` – A Weka's Instance to be transformed into a Bag.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

6.5.1.7 Methods

- **getBagAsInstances**

  public weka.core.Instances getBagAsInstances() **throws** java.lang.Exception

  - **Description**
    Gets a bag in the form of a set of instances considering just the relational information. Neither the identifier attribute of the Bag nor label attributes are included. For instance, given the relation toy above, the output of the method is the relation bag.
    @relation toy
    @attribute id {bag1,bag2}
    @attribute bag relational
    @attribute f1 numeric
    @attribute f2 numeric
    @attribute f3 numeric
    @end bag
    @attribute label1 {0,1}
    @attribute label2 {0,1}
    @attribute label3 {0,1}
    @attribute label4 {0,1}
    @relation bag
    @attribute f1 numeric
    @attribute f2 numeric
    @attribute f3 numeric
  - **Returns** – Instances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **getInstance**

  public weka.core.Instance getInstance(**int** bagIndex)

  - **Description**
    Returns an instance of the Bag with index bagIndex.

- **Parameters**
    - `bagIndex` – The index number.
- **Returns** – Instance.

- **getNumAttributesInABag**

  public int getNumAttributesInABag()

  - **Description**
    Gets the number of attributes of in the relational attribute of a Bag. For instance, in the relation above, the output of the method is 3.
    @relation toy
    @attribute id {bag1,bag2}
    @attribute bag relational
    @attribute f1 numeric
    @attribute f2 numeric
    @attribute f3 numeric
    @end bag
    @attribute label1 {0,1}
    @attribute label2 {0,1}
    @attribute label3 {0,1}
    @attribute label4 {0,1}

  - **Returns** – The number of attributes.

- **getNumAttributesWithRelational**

  public int getNumAttributesWithRelational()

  - **Description**
    Gets the total number of attributes of the Bag. This number includes attributes corresponding to labels. Instead the relational attribute, the number of attributes contained in the relational attribute is considered. For instance, in the relation above, the output of the method is 8.
    @relation toy
    @attribute id {bag1,bag2}
    @attribute bag relational
    @attribute f1 numeric
    @attribute f2 numeric
    @attribute f3 numeric
    @end bag
    @attribute label1 {0,1}
    @attribute label2 {0,1}
    @attribute label3 {0,1}
    @attribute label4 {0,1}

  - **Returns** – Total number of attributes of the Bag.

- **getNumInstances**

  public int getNumInstances()

- **Description**
  Gets the number of instances of the Bag.
- **Returns** – The number of instances of the Bag.

• **setValue**

public void setValue(**int** instanceIndex,**int** attrIndex,**double** value)

- **Description**
  Sets the value of attrIndex attribute of the the instanceIndex to a certain value.
- **Parameters**
  - `instanceIndex` – The index of the instance.
  - `attrIndex` – The index of the attribute.
  - `value` – The value to be set.

6.5.2 Class MIMLInstances

Class inheriting from MultiLabelnstances to represent MIML data.

6.5.2.1 Declaration

public class MIMLInstances
extends mulan.data.MultiLabelInstances

6.5.2.2 Field summary

serialVersionUID Generated Serial version UID.

6.5.2.3 Constructor summary

MIMLInstances(Instances, LabelsMetaData) Constructor.
MIMLInstances(Instances, String) Constructor.
MIMLInstances(String, int) Constructor.
MIMLInstances(String, String) Constructor.

6.5.2.4 Method summary

addBag(MIMLBag) Adds a Bag of Instances to the dataset.
addInstance(MIMLBag, int) Adds a Bag of Instances to the dataset in a certain index.
getBag(int) Gets a `MIMLBag` (6.5.1) (i.e. pattern) with a certain bagIndex.
getBagAsInstances(int) Gets a `MIMLBag` (6.5.1) with a certain bagIndex in the form of a set of `Instances` considering just the relational information.
getInstance(int, int) Gets an instance of a bag.
getMLDataSet() Returns the dataset as MultiLabelInstances.
getNumAttributes() Gets the number of attributes of the dataset considering label attributes and the relational attribute with bags as a single attribute.
getNumAttributesInABag() Gets the number of attributes per bag.

getNumAttributesWithRelational() Gets the total number of attributes of the dataset.
getNumBags() Gets the number of bags of the dataset.
getNumInstances(int) Gets the number of instances of a bag.
insertAttributesToBags(ArrayList) Adds a set of attributes to the relational attribute with values ?
insertAttributeToBags(Attribute) Adds an attribute to the relational attribute with value ?

6.5.2.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.

6.5.2.6 Constructors

- **MIMLInstances**

  public MIMLInstances(weka.core.Instances dataSet,mulan.data.LabelsMetaData labelsMetaData) **throws** mulan.data.InvalidDataFormatException

  - **Description**
    Constructor.
  - **Parameters**
    * `dataSet` – A dataset of `Instances` with relational information.
    * `labelsMetaData` – Information about labels.
  - **Throws**
    * `mulan.data.InvalidDataFormatException` – To be handled in an upper level.

- **MIMLInstances**

  public MIMLInstances(weka.core.Instances dataSet,java.lang.String xmlLabelsDefFilePath) **throws** mulan.data.InvalidDataFormatException

  - **Description**
    Constructor.
  - **Parameters**
    * `dataSet` – A dataset of `Instances` with relational information.
    * `xmlLabelsDefFilePath` – Path of .xml file with information about labels.
  - **Throws**
    * `mulan.data.InvalidDataFormatException` – To be handled in an upper level.

- **MIMLInstances**

  public MIMLInstances(java.lang.String arffFilePath,**int** numLabelAttributes) **throws** mulan.data.InvalidDataFormatException

  - **Description**
    Constructor.

- **Parameters**
  - `arffFilePath` – Path of .arff file with Instances.
  - `numLabelAttributes` – Number of label attributes.
- **Throws**
  - `mulan.data.InvalidDataFormatException` – To be handled in an upper level.

- **MIMLInstances**

  public MIMLInstances(java.lang.String arffFilePath,java.lang.String xmlLabelsDefFilePath)
  throws mulan.data.InvalidDataFormatException

  - **Description**
    Constructor.
  - **Parameters**
    - `arffFilePath` – Path of .arff file with Instances.
    - `xmlLabelsDefFilePath` – Path of .xml file with information about labels.
  - **Throws**
    - `mulan.data.InvalidDataFormatException` – To be handled in an upper level.

6.5.2.7 Methods

- **addBag**

  public void addBag(MIMLBag bag)

  - **Description**
    Adds a Bag of Instances to the dataset.
  - **Parameters**
    - `bag` – A Bag of Instances.

- **addInstance**

  public void addInstance(MIMLBag bag,**int** index)

  - **Description**
    Adds a Bag of Instances to the dataset in a certain index.
  - **Parameters**
    - `bag` – A Bag of Instances.
    - `index` – The index to insert the Bag.

- **getBag**

  public MIMLBag getBag(**int** bagIndex) **throws** java.lang.Exception

  - **Description**
    Gets a `MIMLBag` (6.5.1) (i.e. pattern) with a certain bagIndex.
  - **Parameters**

    * `bagIndex` – Index of the bag.
  - **Returns** – Bag If bagIndex exceeds the number of bags in the dataset. To be handled in an upper level.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

- **getBagAsInstances**

  public weka.core.Instances getBagAsInstances(**int** bagIndex) **throws** java.lang.Exception

  - **Description**

    Gets a `MIMLBag` (6.5.1) with a certain bagIndex in the form of a set of `Instances` considering just the relational information. Neither identification attribute of the Bag nor label attributes are included.
  - **Parameters**
    * `bagIndex` – Index of the bag.
  - **Returns** – A bag or an instance from the index of the dataset.
  - **Throws**
    * `java.lang.Exception` – If bagIndex exceeds the number of bags in the dataset. To be handled in an upper level.

- **getInstance**

  public weka.core.Instance getInstance(**int** bagIndex,**int** instanceIndex) **throws** java.lang.IndexOutOfBoundsException

  - **Description**

    Gets an instance of a bag.
  - **Parameters**
    * `bagIndex` – The index of the bag in the data set.
    * `instanceIndex` – Is the index of the instance in the bag.
  - **Returns** – Instance.
  - **Throws**
    * `java.lang.IndexOutOfBoundsException` – To be handled in an upper level.

- **getMLDataSet**

  public mulan.data.MultiLabelInstances getMLDataSet()

  - **Description**

    Returns the dataset as MultiLabelInstances.
  - **Returns** – MultiLabelInstances.

- **getNumAttributes**

  public int getNumAttributes()

- **Description**
  Gets the number of attributes of the dataset considering label attributes and the relational attribute with bags as a single attribute. For instance, in relation above, the returned value is 6. @relation toy
  @attribute id {bag1,bag2}
  @attribute bag relational
  @attribute f1 numeric
  @attribute f2 numeric
  @attribute f3 numeric
  @end bag
  @attribute label1 {0,1}
  @attribute label2 {0,1}
  @attribute label3 {0,1}
  @attribute label4 {0,1}

- **Returns** – The number of attributes of the dataset.

• **getNumAttributesInABag**

public int getNumAttributesInABag()

- **Description**
  Gets the number of attributes per bag. In MIML all bags have the same number of attributes.* For instance, in the relation above, the output of the method is 3.
  @relation toy
  @attribute id {bag1,bag2}
  @attribute bag relational
  @attribute f1 numeric
  @attribute f2 numeric
  @attribute f3 numeric
  @end bag
  @attribute label1 {0,1}
  @attribute label2 {0,1}
  @attribute label3 {0,1}
  @attribute label4 {0,1}

- **Returns** – The number of attributes per bag.

• **getNumAttributesWithRelational**

public int getNumAttributesWithRelational()

- **Description**
  Gets the total number of attributes of the dataset. This number includes attributes corresponding to labels. Instead the relational attribute, the number of attributes contained in the relational attribute is considered. For instance, in the relation above, the output of the method is 8.
  @relation toy
  @attribute id {bag1,bag2}
  @attribute bag relational
  @attribute f1 numeric
  @attribute f2 numeric

@attribute f3 numeric
@end bag
@attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}

– **Returns** – The total number of attributes of the dataset.

- **getNumBags**

public int getNumBags()

– **Description**
Gets the number of bags of the dataset.
– **Returns** – The number of bags of the dataset.

- **getNumInstances**

public int getNumInstances(**int** bagIndex) **throws** java.lang.Exception

– **Description**
Gets the number of instances of a bag.
– **Parameters**
  * `bagIndex` – A bag index.
– **Returns** – The number of instances of a bag
– **Throws**
  * `java.lang.Exception` – To be handled in an upper level.

- **insertAttributesToBags**

public MIMLInstances insertAttributesToBags(java.util.ArrayList Attributes) **throws** mulan.data.
InvalidDataFormatException

– **Description**
Adds a set of attributes to the relational attribute with values ? at the last position of the relational attribute.
– **Parameters**
  * `Attributes` – ArrayList of attributes to add.
– **Returns** – new dataset.
– **Throws**
  * `mulan.data.InvalidDataFormatException` – if occurred an error creating new dataset.

- **insertAttributeToBags**

public MIMLInstances insertAttributeToBags(weka.core.Attribute newAttr) **throws** mulan.data.
InvalidDataFormatException

- **Description**

  Adds an attribute to the relational attribute with value ? at the last position.
- **Parameters**
  - `newAttr` – The attribute to be added.
- **Returns** – new dataset.
- **Throws**
  - `mulan.data.InvalidDataFormatException` – if occurred an error creating new dataset.

6.5.3 Class MLSave

Class with methods to write to file a multi-label dataset. MIML format is also supported.

6.5.3.1 Declaration

public final class MLSave
extends java.lang.Object

6.5.3.2 Constructor summary

MLSave()

6.5.3.3 Method summary

saveArff(Instances, String) Writes an arff file with an Instances dataset.
saveArff(MIMLInstances, String) Writes an arff file with a multi-label dataset.
saveArff(MultiLabelInstances, String) Writes an arff file with a multi-label dataset.
saveXml(ArrayList, String) Writes an xml file.
saveXml(Instances, String) Writes an xml file with label definitions of an instances dataset.
saveXml(MultiLabelInstances, String) Writes an xml file with label definitions of a multi-label dataset.

6.5.3.4 Constructors

- **MLSave**

  private MLSave()

6.5.3.5 Methods

- **saveArff**

  public static void saveArff(weka.core.Instances instances,java.lang.String pathName) **throws** java.io.IOException

  - **Description**
    Writes an arff file with an Instances dataset.
  - **Parameters**
    - `instances` – A dataset.
    - `pathName` – Name and path for file to write.
  - **Throws**
    - `java.io.IOException` – To be handled in an upper level.

- **saveArff**

  public static void saveArff(MIMLInstances instances,java.lang.String pathName) **throws** java.io.IOException

  - **Description**
    Writes an arff file with a multi-label dataset. MIML format is also supported.
  - **Parameters**
    - `instances` – A multi-label dataset.
    - `pathName` – Name and path for file to write.
  - **Throws**
    - `java.io.IOException` – To be handled in an upper level.

- **saveArff**

  public static void saveArff(mulan.data.MultiLabelInstances instances,java.lang.String pathName) **throws** java.io.IOException

  - **Description**
    Writes an arff file with a multi-label dataset. MIML format is also supported.
  - **Parameters**
    - `instances` – A multi-label dataset.
    - `pathName` – Name and path for file to write.
  - **Throws**
    - `java.io.IOException` – To be handled in an upper level.

- **saveXml**

  public static void saveXml(java.util.ArrayList labelNames,java.lang.String pathName)

  - **Description**
    Writes an xml file.
  - **Parameters**

    * `labelNames` – An ArrayList<String>with label names.
    * `pathName` – Name and path for file to write.

- **saveXml**

  public static void saveXml(weka.core.Instances instances,java.lang.String pathName) **throws** java.io.IOException, mulan.data.LabelsBuilderException

  – **Description**

    Writes an xml file with label definitions of an instances dataset.

  – **Parameters**
    * `instances` – A dataset.
    * `pathName` – Name and path for file to write.

  – **Throws**
    * `java.io.IOException` – To be handled in an upper level.
    * `mulan.data.LabelsBuilderException` – To be handled in an upper level.

- **saveXml**

  public static void saveXml(mulan.data.MultiLabelInstances instances,java.lang.String pathName) **throws** java.io.IOException, mulan.data.LabelsBuilderException

  – **Description**

    Writes an xml file with label definitions of a multi-label dataset. MIML format is also supported.

  – **Parameters**
    * `instances` – A multi-label dataset.
    * `pathName` – Name and path for file to write.

  – **Throws**
    * `java.io.IOException` – To be handled in an upper level.
    * `mulan.data.LabelsBuilderException` – To be handled in an upper level.

6.6 Package miml.classifiers.miml.meta

Package Contents *Page*

6.6.1 Class MIMLBagging

Class implementing an ensemble algorithm using bagging. For more information, see *Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.*

6.6.1.1 Declaration

public class MIMLBagging
extends miml.classifiers.miml.MIMLClassifier

6.6.1.2 Field summary

baseLearner Base learner.
ensemble The ensemble of MultiLabelLearners.
numClassifiers Number of classifiers in the ensemble.
samplePercentage The size of the sample.
sampleWithReplacement Determines whether the classifier will consider sampling with replacement.
seed Seed for randomization.
serialVersionUID Generated Serial version UID.
threshold Threshold for predictions.
useConfidences Determines whether confidences [0,1] or relevance {0,1} is used to compute bipartition.

6.6.1.3 Constructor summary

MIMLBagging() No-argument constructor for xml configuration.
MIMLBagging(IMIMLClassifier, int) Constructor of the class.

6.6.1.4 Method summary

buildInternal(MIMLInstances)
configure(Configuration)
getNumClassifiers() Returns the number of classifiers of the ensemble.
getSamplePercentage() Returns the percentage of instances used for sampling with replacement.
getThreshold() Returns the value of the threshold.
isSampleWithReplacement() Returns true if the algorithm is configured with sampling and false otherwise.
isUseConfidences() Returns whether the classifier uses confidences of bipartitions to combine classifiers in the ensemble.
makePredictionInternal(MIMLBag)
setSamplePercentage(int) Sets the percentage of instances used for sampling with replacement*.
setSampleWithReplacement(boolean) Configure the classifier to use/not use sampling with replacement.
setSeed(int) Sets the seed value.
setThreshold(double) Sets the value of the threshold.
setUseConfidences(boolean) Stablishes whether confidences or bipartions are used to combine classifiers in the ensemble.

6.6.1.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.
- `protected double` **threshold**
  - Threshold for predictions.
- `protected int` **seed**
  - Seed for randomization.
- `boolean` **sampleWithReplacement**
  - Determines whether the classifier will consider sampling with replacement. By default it is false.
- `boolean` **useConfidences**
  - Determines whether confidences [0,1] or relevance {0,1} is used to compute bipartition.
- `int` **samplePercentage**
  - The size of the sample.
- `protected int` **numClassifiers**
  - Number of classifiers in the ensemble.
- `protected miml.classifiers.miml.IMIMLClassifier` **baseLearner**
  - Base learner.
- `protected miml.classifiers.miml.IMIMLClassifier[]` **ensemble**
  - The ensemble of MultiLabelLearners. To be initialized by the builder method.

6.6.1.6 Constructors

- **MIMLBagging**

  public MIMLBagging()

  - **Description**
    No-argument constructor for xml configuration.
- **MIMLBagging**

  public MIMLBagging(miml.classifiers.miml.IMIMLClassifier baseLearner,**int** numClassifiers)

  - **Description**
    Constructor of the class. Its default setting is: @li sampleWithReplacement=false @li threshold=0.5.
  - **Parameters**
    - `baseLearner` – The base learner to be used.
    - `numClassifiers` – The number of base classifiers in the ensemble.

6.6.1.7 Methods

- **buildInternal**

  protected abstract void buildInternal(miml.data.MIMLInstances trainingSet) **throws** java.lang.
  Exception

  - **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)
    Learner specific implementation of building the model from `MultiLabelInstances` training data set. This method is called from `build(MultiLabelInstances)` method, where behavior common across all learners is applied.
  - **Parameters**
    * `trainingSet` – The training data set.
  - **Throws**
    * `java.lang.Exception` – if learner model was not created successfully.

- **configure**

  public void configure(org.apache.commons.configuration2.Configuration configuration)

- **getNumClassifiers**

  public int getNumClassifiers()

  - **Description**
    Returns the number of classifiers of the ensemble.
  - **Returns** – Number of classifiers.

- **getSamplePercentage**

  public int getSamplePercentage()

  - **Description**
    Returns the percentage of instances used for sampling with replacement.
  - **Returns** – The sample percentage.

- **getThreshold**

  public double getThreshold()

  - **Description**
    Returns the value of the threshold.
  - **Returns** – double The threshold.

- **isSampleWithReplacement**

  public boolean isSampleWithReplacement()

  - **Description**
    Returns true if the algorithm is configured with sampling and false otherwise.

  - **Returns** – True if the algorithm is configured with sampling and false otherwise.

- **isUseConfidences**

  public boolean isUseConfidences()

  - **Description**
    Returns whether the classifier uses confidences of bipartitions to combine classifiers in the ensemble.
  - **Returns** – True, if is use confidences.

- **makePredictionInternal**

  protected abstract mulan.classifier.MultiLabelOutput makePredictionInternal(miml.data.MIMLBag instance) **throws** java.lang.Exception, mulan.classifier.InvalidDataException

  - **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)
    Learner specific implementation for predicting on specified data based on trained model. This method is called from `makePrediction(Instance)` which guards for model initialization and apply common handling/behavior.
  - **Parameters**
    - `instance` – The data instance to predict on.
  - **Returns** – The output of the learner for the given instance.
  - **Throws**
    - `java.lang.Exception` – If an error occurs while making the prediction.
    - `mulan.classifier.InvalidDataException` – If specified instance data is invalid and can not be processed by the learner.

- **setSamplePercentage**

  public void setSamplePercentage(**int** samplePercentage)

  - **Description**
    Sets the percentage of instances used for sampling with replacement*.
  - **Parameters**
    - `samplePercentage` – The size of the sample referring the original one.

- **setSampleWithReplacement**

  public void setSampleWithReplacement(**boolean** sampleWithReplacement)

  - **Description**
    Configure the classifier to use/not use sampling with replacement.
  - **Parameters**
    - `sampleWithReplacement` – True if the classifier is set to use sampling with replacement.

- **setSeed**

public void setSeed(**int** seed)

- **Description**
  Sets the seed value.
- **Parameters**
  * `seed` – The seed value.

- **setThreshold**

public void setThreshold(**double** threshold)

- **Description**
  Sets the value of the threshold.
- **Parameters**
  * `threshold` – The value of the threshold.

- **setUseConfidences**

public void setUseConfidences(**boolean** useConfidences)

- **Description**
  Stablishes whether confidences or bipartions are used to combine classifiers in the ensemble.
- **Parameters**
  * `useConfidences` – The value of the property.

6.7 Package miml.core

Package Contents *Page*

6.7.1 Interface IConfiguration

Interface used to indicate that a class can be configured.

6.7.1.1 Declaration

public interface IConfiguration

6.7.1.2 All known subinterfaces

EvaluatorHoldout (6.2.3), EvaluatorCV (6.2.2), MIMLClassifier (6.3.2), MIMLBagging (6.6.1), MIMLClassifierToMI (6.9.2), MIMLReport (6.10.3), BaseMIMLReport (6.10.2), MultiLabelKNN_MIMLWrapper (6.13.7), MLkNN_MIMLWrapper (6.13.6), MIMLkNN (6.13.5), IBLR_ML_MIMLWrapper (6.13.4), DMLkNN_MIMLWrapper (6.13.3), BRkNN_MIMLWrapper (6.13.1), MIMLClassifierToML (6.14.1)

6.7.1.3 All classes known to implement interface

EvaluatorHoldout (6.2.3), EvaluatorCV (6.2.2), MIMLClassifier (6.3.2), MIMLReport (6.10.3)

6.7.1.4 Method summary

configure(Configuration) Method to configure the class with the given configuration.

6.7.1.5 Methods

- **configure**

  void configure(org.apache.commons.configuration2.Configuration configuration)

  - **Description**
    Method to configure the class with the given configuration.
  - **Parameters**
    - `configuration` – Configuration used to configure the class.

6.7.2 Class ConfigLoader

Class used to read a xml file and configure an experiment.

6.7.2.1 Declaration

public class ConfigLoader
extends java.lang.Object

6.7.2.2 Field summary

configuration Configuration object.

6.7.2.3 Constructor summary

ConfigLoader(String) Constructor that sets the configuration file

6.7.2.4 Method summary

getConfiguration() Gets the experiment's configuration.
loadClassifier() Read current configuration to load and configure the classifier.
loadEvaluator() Read current configuration to load and configure the evaluator.
loadReport() Read current configuration to load and configure the report.
setConfiguration(Configuration) Sets the configuration for the experiment.

6.7.2.5 Fields

- `protected org.apache.commons.configuration2.Configuration` **configuration**
  - Configuration object.

6.7.2.6 Constructors

- **ConfigLoader**

  public ConfigLoader(java.lang.String path) **throws** org.apache.commons.configuration2.ex.ConfigurationException

  - **Description**
    Constructor that sets the configuration file
  - **Parameters**
    * `path` – The path of config file.
  - **Throws**
    * `org.apache.commons.configuration2.ex.ConfigurationException` – if occurred an error during the loading of the configuration.

6.7.2.7 Methods

- **getConfiguration**

  public org.apache.commons.configuration2.Configuration getConfiguration()

  - **Description**
    Gets the experiment's configuration.
  - **Returns** – The configuration used during experimentation.

- **loadClassifier**

  public miml.classifiers.miml.IMIMLClassifier loadClassifier() **throws** java.lang.Exception

  - **Description**
    Read current configuration to load and configure the classifier.
  - **Returns** – A MIMLClassifier.
  - **Throws**
    * `java.lang.Exception` – if the classifier couldn't be loaded correctly.

- **loadEvaluator**

  public miml.evaluation.IEvaluator loadEvaluator() **throws** java.lang.Exception

  - **Description**
    Read current configuration to load and configure the evaluator.
  - **Returns** – A evaluator for MIML Classifiers.
  - **Throws**
    * `java.lang.Exception` – if the class loaded can't be loaded.

- **loadReport**

  public miml.report.IReport loadReport() **throws** java.lang.Exception

  - **Description**
    Read current configuration to load and configure the report.
  - **Returns** – the MIML report
  - **Throws**
    * `java.lang.Exception` – if the class can't be loaded.

- **setConfiguration**

  public void setConfiguration(org.apache.commons.configuration2.Configuration configuration)

  - **Description**
    Sets the configuration for the experiment.
  - **Parameters**
    * `configuration` – A new configuration.

6.7.3 Class ConfigParameters

Class used to save configuration parameters to be used in reports.

6.7.3.1 Declaration

public final class ConfigParameters
extends java.lang.Object

6.7.3.2 Field summary

algorirthmName The algorirthm used in the experimentation.
classifierName The classifier used in the experimentation.
configFileName The config filename used in the experimentation.
dataFileName The name of data file used in the experimentation.
isDegenerative
transformationMethod The transform method used in the experimentation.

6.7.3.3 Constructor summary

ConfigParameters()

6.7.3.4 Method summary

getAlgorirthmName() Gets the algorirthm name.
getClassifierName() Gets the classifier name.
getConfigFileName() Gets the configuration file name.
getDataFileName() Gets the name of data file.
getIsDegenerative() Gets if the method used is degenerative. .
getTransformationMethod() Gets the transform method used in the experiment.
setAlgorirthmName(String) Sets the algorirthm name.
setClassifierName(String) Sets the classifier name.
setConfigFileName(String) Sets the configuration file name.
setDataFileName(String) Sets the data file name.
setIsDegenerative(Boolean) Sets if the method used is degenerative.
setTransformationMethod(String) Sets the transform method used in the experiment.

6.7.3.5 Fields

- `protected static java.lang.String` **algorirthmName**
  - The algorirthm used in the experimentation.
- `protected static java.lang.String` **configFileName**
  - The config filename used in the experimentation.
- `protected static java.lang.String` **dataFileName**
  - The name of data file used in the experimentation.
- `protected static java.lang.String` **classifierName**
  - The classifier used in the experimentation.
- `protected static java.lang.String` **transformationMethod**
  - The transform method used in the experimentation.
- `protected static java.lang.Boolean` **isDegenerative**

6.7.3.6 Constructors

- **ConfigParameters**

public ConfigParameters()

6.7.3.7 Methods

- **getAlgorirthmName**

  public static java.lang.String getAlgorirthmName()

  - **Description**
    Gets the algorirthm name.
  - **Returns** – The algorirthm name.

- **getClassifierName**

  public static java.lang.String getClassifierName()

  - **Description**
    Gets the classifier name.
  - **Returns** – The classifier name.

- **getConfigFileName**

  public static java.lang.String getConfigFileName()

  - **Description**
    Gets the configuration file name.
  - **Returns** – The configuration file name.

- **getDataFileName**

  public static java.lang.String getDataFileName()

  - **Description**
    Gets the name of data file.
  - **Returns** – The name of data file.

- **getIsDegenerative**

  public static java.lang.Boolean getIsDegenerative()

  - **Description**
    Gets if the method used is degenerative. .
  - **Returns** – True if the method used is degenerative

- **getTransformationMethod**

  public static java.lang.String getTransformationMethod()

  - **Description**
    Gets the transform method used in the experiment.
  - **Returns** – The transform method used in the experiment.

- **setAlgorirthmName**

  public static void setAlgorirthmName(java.lang.String algorirthmName)

  - **Description**
    Sets the algorirthm name.
  - **Parameters**
    * `algorirthmName` – The new algorirthm name.

- **setClassifierName**

  public static void setClassifierName(java.lang.String classifierName)

  - **Description**
    Sets the classifier name.
  - **Parameters**
    * `classifierName` – The classifier name.

- **setConfigFileName**

  public static void setConfigFileName(java.lang.String configFileName)

  - **Description**
    Sets the configuration file name.
  - **Parameters**
    * `configFileName` – The new configuration file name.

- **setDataFileName**

  public static void setDataFileName(java.lang.String dataFileName)

  - **Description**
    Sets the data file name.
  - **Parameters**
    * `dataFileName` – the new data file name

- **setIsDegenerative**

  public static void setIsDegenerative(java.lang.Boolean isDegenerative)

  - **Description**
    Sets if the method used is degenerative.
  - **Parameters**
    * `isDegenerative` – If the method used is degenerative.

- **setTransformationMethod**

  public static void setTransformationMethod(java.lang.String transformationMethod)

  - **Description**
    Sets the transform method used in the experiment.
  - **Parameters**
    * `transformationMethod` – The transform method used in the experiment.

6.7.4 Class Params

This class contains the list of classes and objects needed to create a new instance of a Multi Label classifier through a specific constructor.

6.7.4.1 Declaration

public class Params
extends java.lang.Object

6.7.4.2 Field summary

classes List of classes needed by the Multi Label classifier's constructor.
objects List of the values for the classes array

6.7.4.3 Constructor summary

Params(Class[], Object[]) Generic constructor

6.7.4.4 Method summary

getClasses()
getObjects()
setClasses(Class[])
setObjects(Object[])

6.7.4.5 Fields

- `private java.lang.Class[]` **classes**
  - List of classes needed by the Multi Label classifier's constructor.
- `private java.lang.Object[]` **objects**
  - List of the values for the classes array

6.7.4.6 Constructors

- **Params**

  public Params(java.lang.Class[] classes,java.lang.Object[] objects)

  - **Description**
    Generic constructor
  - **Parameters**
    * `classes` – The list of classes needed by the Multi Label classifier's constructor.
    * `objects` – The list of the values for the classes array.

6.7.4.7 Methods

- **getClasses**

  public java.lang.Class[] getClasses()

  - **Returns** – the classes

- **getObjects**

  public java.lang.Object[] getObjects()

  - **Returns** – the objects

- **setClasses**

  public void setClasses(java.lang.Class[] classes)

  - **Parameters**
    * `classes` – the classes to set

- **setObjects**

  public void setObjects(java.lang.Object[] objects)

  - **Parameters**
    * `objects` – the objects to set

6.7.5 Class Utils

This class has utilies that can be used anywhere in the library.

6.7.5.1 Declaration

public final class Utils
extends java.lang.Object

6.7.5.2 Constructor summary

Utils()

6.7.5.3 Method summary

readMultiLabelLearnerParams(Configuration) Read the configuration parameters for a specific Multi Label classifier's constructor
resample(Instances, double, boolean, int) Obtains a sample of the original data.
splitData(MIMLInstances, double, int) Split data given a percentage.

6.7.5.4 Constructors

- **Utils**

  public Utils()

6.7.5.5 Methods

- **readMultiLabelLearnerParams**

  public static Params readMultiLabelLearnerParams(org.apache.commons.configuration2.
  Configuration configuration)

  - **Description**
    Read the configuration parameters for a specific Multi Label classifier's constructor
  - **Parameters**
    - `configuration` – Configuration used to configure the class
  - **Returns** – Params class which contains the parameters of classifier's constructor

- **resample**

  public static weka.core.Instances resample(weka.core.Instances data,**double** percentage,**boolean**
  sampleWithReplacement,**int** seed) **throws** java.lang.Exception

  - **Description**
    Obtains a sample of the original data.
  - **Parameters**
    - `data` – Instances with the dataset.
    - `percentage` – percentage of instances that will contain the new dataset.
    - `sampleWithReplacement` – If true the sample will be with replacement.
    - `seed` – Seed for randomization. Necessary if instances have not been previously shuffled with randomize.
  - **Returns** – Instances.
  - **Throws**
    - `java.lang.Exception` – To be handled.

- **splitData**

  public static java.util.List splitData(miml.data.MIMLInstances mimlDataSet,**double**
  percentageTrain,**int** seed) **throws** java.lang.Exception

  - **Description**
    Split data given a percentage.
  - **Parameters**
    - `mimlDataSet` – The MIML dataset to be splited.
    - `percentageTrain` – The percentage (0-100) to be used in train.
    - `seed` – Seed use to randomize.
  - **Returns** – A list with the dataset splited.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

6.8 Package miml.transformation.mimlTOmi

Package Contents *Page*

6.8.1 Class BRTransformation

Class that uses Binary Relevance transformation to convert MIMLInstances to MIL Instances with relational attribute.

6.8.1.1 Declaration

public class BRTransformation
extends java.lang.Object **implements** java.io.Serializable

6.8.1.2 Field summary

BRT Binary Relevance Transformation.
dataSet MIML dataSet.
serialVersionUID For serialization.

6.8.1.3 Constructor summary

BRTransformation(MIMLInstances) Constructor.

6.8.1.4 Method summary

transformBag(int, int) Removes all label attributes except labelToKeep.
transformBag(MIMLBag, int) Removes all label attributes except labelToKeep.
transformBag(MIMLBag, int[], int) Remove all label attributes except label at position indexToKeep.
transformBags(int) Remove all label attributes except labelToKeep.
transformBags(MIMLInstances, int[], int) Remove all label attributes except that at indexOfLabelToKeep.

6.8.1.5 Fields

- `private static final long` **serialVersionUID**
  - For serialization.
- `protected mulan.transformations.BinaryRelevanceTransformation` **BRT**
  - Binary Relevance Transformation.
- `protected miml.data.MIMLInstances` **dataSet**
  - MIML dataSet.

6.8.1.6 Constructors

- **BRTransformation**

  public BRTransformation(miml.data.MIMLInstances dataSet)

  - **Description**
    Constructor.
  - **Parameters**
    - `dataSet` – MIMLInstances dataset.

6.8.1.7 Methods

- **transformBag**

  public weka.core.Instance transformBag(**int** bagIndex,**int** labelToKeep) **throws** java.lang.
  Exception

  - **Description**
    Removes all label attributes except labelToKeep.
  - **Parameters**
    - `bagIndex` – The bagIndex of the Bag to be transformed.
    - `labelToKeep` – The label to keep. A value in [0, numLabels-1].
  - **Returns** – Instance.
  - **Throws**
    - `java.lang.Exception` – To be handled in upper level.
- **transformBag**

  public weka.core.Instance transformBag(miml.data.MIMLBag instance,**int** labelToKeep)

  - **Description**
    Removes all label attributes except labelToKeep.
  - **Parameters**
    - `instance` – The instance from which labels are to be removed.
    - `labelToKeep` – The label to keep. A value in [0, numLabels-1].

- **Returns** – Instance

- **transformBag**

public static weka.core.Instance transformBag(miml.data.MIMLBag instance,**int**[] labelIndices, **int** indexToKeep)

  - **Description**
    Remove all label attributes except label at position indexToKeep.
  - **Parameters**
    - `instance` – The instance from which labels are to be removed.
    - `labelIndices` – Array storing, for each label its corresponding label. index.
    - `indexToKeep` – The label index to keep.
  - **Returns** – transformed Instance.

- **transformBags**

public weka.core.Instances transformBags(**int** labelToKeep) **throws** java.lang.Exception

  - **Description**
    Remove all label attributes except labelToKeep.
  - **Parameters**
    - `labelToKeep` – The label to keep. A value in [0, numLabels-1].
  - **Returns** – Instances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformBags**

public static weka.core.Instances transformBags(miml.data.MIMLInstances dataSet,**int**[] labelIndices,**int** indexToKeep) **throws** java.lang.Exception

  - **Description**
    Remove all label attributes except that at indexOfLabelToKeep.
  - **Parameters**
    - `dataSet` – A MIMLInstances dataset.
    - `labelIndices` – Array storing, for each label its corresponding label index.
    - `indexToKeep` – The label index to keep.
  - **Returns** – Instances.
  - **Throws**
    - `java.lang.Exception` – when removal fails.

6.8.2 Class LPTransformation

Class that uses LabelPowerset transformation to convert MIMLInstances to MIL Instances with relational attribute.

6.8.2.1 Declaration

public class LPTransformation
extends java.lang.Object **implements** java.io.Serializable

6.8.2.2 Field summary

LPT LabelPowerSetTransformation.
serialVersionUID For serialization.

6.8.2.3 Constructor summary

LPTransformation() Constructor.

6.8.2.4 Method summary

getLPT() Returns the format of the transformed instances.
transformBag(MIMLBag, int[])
transformBags(MIMLInstances)

6.8.2.5 Fields

- `private static final long` **serialVersionUID**
  - For serialization.
- `protected MIMLLabelPowersetTransformation` **LPT**
  - LabelPowerSetTransformation.

6.8.2.6 Constructors

- **LPTransformation**

  public LPTransformation()

  - **Description**
    Constructor.

6.8.2.7 Methods

- **getLPT**

  public mulan.transformations.LabelPowersetTransformation getLPT()

  - **Description**
    Returns the format of the transformed instances.
  - **Returns** – the format of the transformed instances.

- **transformBag**

  public weka.core.Instance transformBag(miml.data.MIMLBag bag,**int**[] labelIndices) **throws** java.lang.Exception

  - **Parameters**
    * `bag` – The bag to be transformed.
    * `labelIndices` – The labels to remove.
  - **Returns** – Instance.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

- **transformBags**

  public weka.core.Instances transformBags(miml.data.MIMLInstances dataSet) **throws** java.lang.Exception

  - **Parameters**
    * `dataSet` – MIMLInstances dataSet.
  - **Returns** – Instances.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

6.8.3 Class MIMLLabelPowersetTransformation

Class that uses LabelPowerset transformation to convert MIMLInstances to MIL Instances with relational attribute.

6.8.3.1 Declaration

class MIMLLabelPowersetTransformation
extends mulan.transformations.LabelPowersetTransformation

6.8.3.2 Field summary

serialVersionUID

6.8.3.3 Constructor summary

MIMLLabelPowersetTransformation()

6.8.3.4 Method summary

transformInstance(Instance, int[])

6.8.3.5 Fields

- `private static final long` **serialVersionUID**

6.8.3.6 Constructors

- **MIMLLabelPowersetTransformation**

  MIMLLabelPowersetTransformation()

6.8.3.7 Methods

- **transformInstance**

  public weka.core.Instance transformInstance(weka.core.Instance instance,**int**[] labelIndices)
      throws java.lang.Exception

  – **Parameters**
    * `instance` – The instance to be transformed
    * `labelIndices` – The labels to remove.
  – **Returns** – Transformed instance.
  – **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

6.9 Package miml.classifiers.miml.mimlTOmi

Package Contents *Page*

6.9.1 Class MIMLBinaryRelevance

Wrapper for mulan BinaryRelevance to be used in MIML to MI algorithms.

6.9.1.1 Declaration

public class MIMLBinaryRelevance
extends mulan.classifier.transformation.BinaryRelevance

6.9.1.2 Field summary

serialVersionUID Generated Serial version UID.

6.9.1.3 Constructor summary

MIMLBinaryRelevance(Classifier) Creates a new instance.

6.9.1.4 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.

6.9.1.5 Constructors

- **MIMLBinaryRelevance**

  public MIMLBinaryRelevance(weka.classifiers.Classifier classifier)

  - **Description**
    Creates a new instance.
  - **Parameters**
    * `classifier` – The base-level classification algorithm that will be used for training each of the binary models.

6.9.2 Class MIMLClassifierToMI

Class implementing the degenerative algorithm for MIML data to solve it with MI learning. For more information, see *Zhou, Z. H., & Zhang, M. L. (2007). Multi-instance multi-label learning with application to scene classification. In Advances in neural information processing systems (pp. 1609-1616).*

6.9.2.1 Declaration

public class MIMLClassifierToMI
 extends miml.classifiers.miml.MIMLClassifier

6.9.2.2 Field summary

serialVersionUID Generated Serial version UID.
transformationClassifier Generic classifier used for transformation.

6.9.2.3 Constructor summary

MIMLClassifierToMI() No-argument constructor for xml configuration.
MIMLClassifierToMI(MultiLabelLearner) Basic constructor.

6.9.2.4 Method summary

buildInternal(MIMLInstances)
configure(Configuration)
makePredictionInternal(MIMLBag)

6.9.2.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.
- `protected mulan.classifier.MultiLabelLearner` **transformationClassifier**
  - Generic classifier used for transformation.

6.9.2.6 Constructors

- **MIMLClassifierToMI**

  public MIMLClassifierToMI()

  - **Description**
    No-argument constructor for xml configuration.

- **MIMLClassifierToMI**

  public MIMLClassifierToMI(mulan.classifier.MultiLabelLearner transformationClassifier)

  - **Description**
    Basic constructor.
  - **Parameters**
    * `transformationClassifier` – Mulan MultiLabelLearner used as transformation method from MIML to MI.

6.9.2.7 Methods

- **buildInternal**

  protected abstract void buildInternal(miml.data.MIMLInstances trainingSet) **throws** java.lang.Exception

  - **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)

    Learner specific implementation of building the model from `MultiLabelInstances` training data set. This method is called from `build(MultiLabelInstances)` method, where behavior common across all learners is applied.
  - **Parameters**
    - `trainingSet` – The training data set.
  - **Throws**
    - `java.lang.Exception` – if learner model was not created successfully.

- **configure**

  public void configure(org.apache.commons.configuration2.Configuration configuration)

- **makePredictionInternal**

  protected abstract mulan.classifier.MultiLabelOutput makePredictionInternal(miml.data.MIMLBag instance) **throws** java.lang.Exception, mulan.classifier.InvalidDataException

  - **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)

    Learner specific implementation for predicting on specified data based on trained model. This method is called from `makePrediction(Instance)` which guards for model initialization and apply common handling/behavior.
  - **Parameters**
    - `instance` – The data instance to predict on.
  - **Returns** – The output of the learner for the given instance.
  - **Throws**
    - `java.lang.Exception` – If an error occurs while making the prediction.
    - `mulan.classifier.InvalidDataException` – If specified instance data is invalid and can not be processed by the learner.

6.9.3 Class MIMLLabelPowerset

Wrapper for mulan LabelPowerset to be used in MIML to MI algorithms.

6.9.3.1 Declaration

public class MIMLLabelPowerset
extends mulan.classifier.transformation.LabelPowerset

6.9.3.2 Field summary

serialVersionUID Generated Serial version UID.

6.9.3.3 Constructor summary

MIMLLabelPowerset(Classifier) Constructor that initializes the learner with a base classifier.

6.9.3.4 Method summary

buildInternal(MultiLabelInstances)

6.9.3.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.

6.9.3.6 Constructors

- **MIMLLabelPowerset**

public MIMLLabelPowerset(weka.classifiers.Classifier classifier)

  - **Description**
    Constructor that initializes the learner with a base classifier.
  - **Parameters**
    * `classifier` – The base single-label classification algorithm.

6.9.3.7 Methods

- **buildInternal**

protected abstract void buildInternal(mulan.data.MultiLabelInstances arg0) **throws** java.lang. Exception

6.10 Package miml.report

6.10.1 Interface IReport

Interface for generate reports with the format specified.

6.10.1.1 Declaration

public interface IReport

6.10.1.2 All known subinterfaces

MIMLReport (6.10.3), BaseMIMLReport (6.10.2)

6.10.1.3 All classes known to implement interface

MIMLReport (6.10.3)

6.10.1.4 Method summary

saveReport(String) Save in a file the specified report.
toCSV(IEvaluator) Convert to CSV the evaluator results.
toString(IEvaluator) Convert to plain text the evaluator results.

6.10.1.5 Methods

- **saveReport**

  void saveReport(java.lang.String report) **throws** java.io.FileNotFoundException

  - **Description**
    Save in a file the specified report.
  - **Parameters**
    - `report` – The formatted string to be saved.
  - **Throws**
    - `java.io.FileNotFoundException` – To be handled in an upper level.

- **toCSV**

  java.lang.String toCSV(miml.evaluation.IEvaluator evaluator) **throws** java.lang.Exception

  - **Description**
    Convert to CSV the evaluator results.
  - **Parameters**
    - `evaluator` – The evaluator with the measures.
  - **Returns** – String with CSV content.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **toString**

  java.lang.String toString(miml.evaluation.IEvaluator evaluator) **throws** java.lang.Exception

  - **Description**
    Convert to plain text the evaluator results.
  - **Parameters**
    * `evaluator` – The evaluator with the measures.
  - **Returns** – String with the content.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

6.10.2 Class BaseMIMLReport

Class used to generate reports with the format specified.

6.10.2.1 Declaration

public class BaseMIMLReport
extends miml.report.MIMLReport

6.10.2.2 Constructor summary

BaseMIMLReport() No-argument constructor for xml configuration.
BaseMIMLReport(List, String, boolean, boolean, boolean) Basic constructor to initialize the report.

6.10.2.3 Method summary

configure(Configuration)
crossValidationToCSV(EvaluatorCV) Read the cross-validation results and transform to CSV format.
crossValidationToString(EvaluatorCV) Read the cross-validation results and transform to plain text.
holdoutToCSV(EvaluatorHoldout) Read the holdout results and transform to CSV format.
holdoutToString(EvaluatorHoldout) Read the holdout results and transform to plain text.
toCSV(IEvaluator)
toString(IEvaluator)

6.10.2.4 Constructors

- **BaseMIMLReport**

  public BaseMIMLReport()

  - **Description**
    No-argument constructor for xml configuration.

- **BaseMIMLReport**

  public BaseMIMLReport(java.util.List measures,java.lang.String filename,**boolean** std,**boolean** labels,**boolean** header)

  - **Description**
    Basic constructor to initialize the report.
  - **Parameters**
    - `measures` – The list of selected measures which is going to be shown in the report.
    - `filename` – The filename where the report's will be saved.
    - `std` – Whether the standard deviation of measures will be shown or not (only valid for cross-validation evaluator).
    - `labels` – Whether the measures for each label will be shown (only valid for Macros Average measures).
    - `header` – Whether the header will be shown.

6.10.2.5 Methods

- **configure**

  public void configure(org.apache.commons.configuration2.Configuration configuration)

- **crossValidationToCSV**

  protected java.lang.String crossValidationToCSV(miml.evaluation.EvaluatorCV evaluator) **throws** java.lang.Exception

  - **Description**
    Read the cross-validation results and transform to CSV format.
  - **Parameters**
    - `evaluator` – The evaluator.
  - **Returns** – String with CSV content.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **crossValidationToString**

  protected java.lang.String crossValidationToString(miml.evaluation.EvaluatorCV evaluator) **throws** java.lang.Exception

  - **Description**
    Read the cross-validation results and transform to plain text.
  - **Parameters**
    * `evaluator` – The evaluator.
  - **Returns** – String with the content.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level

- **holdoutToCSV**

  protected java.lang.String holdoutToCSV(miml.evaluation.EvaluatorHoldout evaluator) **throws** java.lang.Exception

  - **Description**
    Read the holdout results and transform to CSV format.
  - **Parameters**
    * `evaluator` – The evaluator.
  - **Returns** – String with CSV content.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level

- **holdoutToString**

  protected java.lang.String holdoutToString(miml.evaluation.EvaluatorHoldout evaluator) **throws** java.lang.Exception

  - **Description**
    Read the holdout results and transform to plain text.
  - **Parameters**
    * `evaluator` – The evaluator.
  - **Returns** – String with the content.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

- **toCSV**

  public java.lang.String toCSV(miml.evaluation.IEvaluator evaluator) **throws** java.lang.Exception

- **toString**

  public java.lang.String toString(miml.evaluation.IEvaluator evaluator) **throws** java.lang.Exception

6.10.3 Class MIMLReport

Abstract class for a MIMLReport.

6.10.3.1 Declaration

public abstract class MIMLReport
extends java.lang.Object **implements** IReport, miml.core.IConfiguration

6.10.3.2 All known subclasses

BaseMIMLReport (6.10.2)

6.10.3.3 Field summary

filename The name of the file where report is saved.
header If the header is going to be printed.
labels If macro measures are broken down by labels.
measures The measures shown in the report.
std If measures' standard deviation are shown.

6.10.3.4 Constructor summary

MIMLReport() No-argument constructor for xml configuration.
MIMLReport(List, String, boolean, boolean, boolean) Basic constructor to initialize the report.

6.10.3.5 Method summary

filterMeasures(List) Filter measures chosen to be shown in the experiment report.
getFilename() Gets the filename.
getMeasures() Gets the measures shown in the report.
isHeader() Checks if header is shown.
isLabels() Checks if measure for each label (Macro Average Measures) is shown.
isStd() Checks if std is going to be shown (only cross-validation).
saveReport(String) Save in a file the specified report.
setFilename(String) Sets the name of the file.
setHeader(boolean) Sets if header is shown.
setLabels(boolean) Sets if measure for each label (Macro Average Measures) is shown.
setMeasures(List) Sets the measures shown in the report.
setStd(boolean) Sets if the std is going to be shown (only cross-validation).

6.10.3.6 Fields

- `protected java.util.List` **measures**
  - The measures shown in the report.
- `protected java.lang.String` **filename**
  - The name of the file where report is saved.
- `protected boolean` **std**
  - If measures' standard deviation are shown.
- `protected boolean` **labels**
  - If macro measures are broken down by labels.
- `protected boolean` **header**
  - If the header is going to be printed.

6.10.3.7 Constructors

- **MIMLReport**

  public MIMLReport()

  - **Description**
    No-argument constructor for xml configuration.

- **MIMLReport**

  public MIMLReport(java.util.List measures,java.lang.String filename,**boolean** std,**boolean** labels, **boolean** header)

  - **Description**
    Basic constructor to initialize the report.
  - **Parameters**
    - `measures` – The list of selected measures which is going to be shown in the report.
    - `filename` – The filename where the report's will be saved.
    - `std` – Whether the standard deviation of measures will be shown or not (only valid for cross-validation evaluator).
    - `labels` – Whether the measures for each label will be shown (only valid for Macros Average measures).
    - `header` – Whether the header will be shown.

6.10.3.8 Methods

- **filterMeasures**

  protected java.util.List filterMeasures(java.util.List allMeasures) **throws** java.lang.Exception

  - **Description**
    Filter measures chosen to be shown in the experiment report.
  - **Parameters**
    - `allMeasures` – All the measures which the evaluation has
  - **Returns** – List with the measures filtered
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **getFilename**

  public java.lang.String getFilename()

  - **Description**
    Gets the filename.
  - **Returns** – The filename.

- **getMeasures**

  public java.util.List getMeasures()

  - **Description**
    Gets the measures shown in the report.
  - **Returns** – The measures.

- **isHeader**

  public boolean isHeader()

  - **Description**
    Checks if header is shown.
  - **Returns** – True, if header is shown.

- **isLabels**

  public boolean isLabels()

  - **Description**
    Checks if measure for each label (Macro Average Measures) is shown.
  - **Returns** – True, if measure for each label is shown.

- **isStd**

  public boolean isStd()

  - **Description**
    Checks if std is going to be shown (only cross-validation).
  - **Returns** – True, if std is going to be shown.

- **saveReport**

  public void saveReport(java.lang.String report) **throws** java.io.FileNotFoundException

  - **Description**
    Save in a file the specified report.
  - **Parameters**
    - `report` – The report.
  - **Throws**
    - `java.io.FileNotFoundException` – To be handled in an upper level.

- **setFilename**

  public void setFilename(java.lang.String filename)

  - **Description**
    Sets the name of the file.

- **Parameters**
  - `filename` – The new filename

- **setHeader**

  public void setHeader(**boolean** header)

  - **Description**
    Sets if header is shown.
  - **Parameters**
    - `header` – The new header configuration.

- **setLabels**

  public void setLabels(**boolean** labels)

  - **Description**
    Sets if measure for each label (Macro Average Measures) is shown.
  - **Parameters**
    - `labels` – The new labels configuration.

- **setMeasures**

  public void setMeasures(java.util.List measures) **throws** java.lang.Exception

  - **Description**
    Sets the measures shown in the report.
  - **Parameters**
    - `measures` – The new measures.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **setStd**

  public void setStd(**boolean** std)

  - **Description**
    Sets if the std is going to be shown (only cross-validation).
  - **Parameters**
    - `std` – The new std configuration.

6.11 Package miml.classifiers.mi

Package Contents *Page*

6.11.1 Class MISMOWrapper

Wrapper for MISMO algorithm to work in MIML to MI classifiers.

6.11.1.1 Declaration

public class MISMOWrapper
extends weka.classifiers.mi.MISMO

6.11.1.2 Field summary

serialVersionUID Generated Serial version UID.

6.11.1.3 Constructor summary

MISMOWrapper()

6.11.1.4 Method summary

distributionForInstance(Instance)

6.11.1.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.

6.11.1.6 Constructors

- **MISMOWrapper**

  public MISMOWrapper()

6.11.1.7 Methods

- **distributionForInstance**

  double[] distributionForInstance(weka.core.Instance arg0) **throws** java.lang.Exception

6.12 Package miml.transformation.mimlTOml

6.12.1 Class ArithmeticTransformation

Class that performs an arithmetic transformation to convert a MIMLIntances class to MultiLabelInstances. This arithmetic transformation transforms each Bag into a single Instance being the value of each attribute the mean value of the instances in the bag.

6.12.1.1 Declaration

public class ArithmeticTransformation
extends miml.transformation.mimlTOml.MIMLtoML

6.12.1.2 Field summary

serialVersionUID For serialization

6.12.1.3 Constructor summary

ArithmeticTransformation()
ArithmeticTransformation(MIMLInstances) Constructor.

6.12.1.4 Method summary

transformDataset()
transformDataset(MIMLInstances)
transformInstance(MIMLBag)
transformInstance(MIMLInstances, MIMLBag)

6.12.1.5 Fields

- `private static final long` **serialVersionUID**
  - For serialization

6.12.1.6 Constructors

- **ArithmeticTransformation**

  public ArithmeticTransformation()

- **ArithmeticTransformation**

  public ArithmeticTransformation(miml.data.MIMLInstances dataset) **throws** java.lang.Exception

  - **Description**
    Constructor.
  - **Parameters**
    - `dataset` – MIMLInstances dataset.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

6.12.1.7 Methods

- **transformDataset**

  public abstract mulan.data.MultiLabelInstances transformDataset() **throws** java.lang.Exception

  - **Description copied from MIMLtoML** (6.12.3)
    Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
  - **Returns** – MultiLabelInstances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformDataset**

  public abstract mulan.data.MultiLabelInstances transformDataset(miml.data.MIMLInstances dataset) **throws** java.lang.Exception

  - **Description copied from MIMLtoML** (6.12.3)
    Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
  - **Parameters**
    - `dataset` – The dataset to be transformed
  - **Returns** – MultiLabelInstances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformInstance**

  public abstract weka.core.Instance transformInstance(miml.data.MIMLBag bag) **throws** java.lang.Exception

  - **Description copied from MIMLtoML** (6.12.3)

    Transforms `MIMLBag` (6.5.1) into Instance.
  - **Parameters**
    - `bag` – The Bag to be transformed.
  - **Returns** – Instance
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformInstance**

  public weka.core.Instance transformInstance(miml.data.MIMLInstances dataset,miml.data.
  MIMLBag bag) **throws** java.lang.Exception

6.12.2 Class GeometricTransformation

Class that performs a geometric transformation to convert a MIMLIntances class to MultiLabelInstances. Each Bag is transformed into a single Instance being the value of each attribute the geometric centor of its max and min values computed as (min_value+max_value)/2.

6.12.2.1 Declaration

public class GeometricTransformation
extends miml.transformation.mimlTOml.MIMLtoML

6.12.2.2 Field summary

serialVersionUID For serialization

6.12.2.3 Constructor summary

GeometricTransformation()
GeometricTransformation(MIMLInstances) Constructor

6.12.2.4 Method summary

transformDataset()
transformDataset(MIMLInstances)
transformInstance(MIMLBag)
transformInstance(MIMLInstances, MIMLBag)

6.12.2.5 Fields

- `private static final long` **serialVersionUID**
  - For serialization

6.12.2.6 Constructors

- **GeometricTransformation**

  public GeometricTransformation() **throws** java.lang.Exception

- **GeometricTransformation**

  public GeometricTransformation(miml.data.MIMLInstances dataset) **throws** java.lang.Exception

  - **Description**
    Constructor
  - **Parameters**
    - `dataset` – MIMLInstances dataset.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

6.12.2.7 Methods

- **transformDataset**

  public abstract mulan.data.MultiLabelInstances transformDataset() **throws** java.lang.Exception

  - **Description copied from MIMLtoML** (6.12.3)
    Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
  - **Returns** – MultiLabelInstances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformDataset**

  public abstract mulan.data.MultiLabelInstances transformDataset(miml.data.MIMLInstances dataset) **throws** java.lang.Exception

  - **Description copied from MIMLtoML** (6.12.3)
    Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
  - **Parameters**
    - `dataset` – The dataset to be transformed
  - **Returns** – MultiLabelInstances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformInstance**

  public abstract weka.core.Instance transformInstance(miml.data.MIMLBag bag) **throws** java.lang.Exception

- **Description copied from MIMLtoML** (6.12.3)

  Transforms `MIMLBag` (6.5.1) into Instance.
- **Parameters**
  * `bag` – The Bag to be transformed.
- **Returns** – Instance
- **Throws**
  * `java.lang.Exception` – To be handled in an upper level.

- **transformInstance**

  public weka.core.Instance transformInstance(miml.data.MIMLInstances dataset,miml.data.
  MIMLBag bag) **throws** java.lang.Exception

6.12.3 Class MIMLtoML

Abstract class to transform MIMLInstances into MultiLabelInstances.

6.12.3.1 Declaration

public abstract class MIMLtoML
extends java.lang.Object **implements** java.io.Serializable

6.12.3.2 All known subclasses

MiniMaxTransformation (6.12.4), GeometricTransformation (6.12.2), ArithmeticTransformation (6.12.1)

6.12.3.3 Field summary

dataset Original data set of MIMLInstances.
serialVersionUID For serialization.
template Template to store Instances.
updatedLabelIndices Array of updated label indices.

6.12.3.4 Constructor summary

MIMLtoML()

6.12.3.5 Method summary

minimax(Instances, int) Get the minimal and maximal value of a certain attribute in a data set.
prepareTemplate() Prepares a template to perform the transformation from MIMLInstances to MultiLabelInstances.
transformDataset() Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
transformDataset(MIMLInstances) Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
transformInstance(MIMLBag) Transforms `MIMLBag` (6.5.1) into Instance.

6.12.3.6 Fields

- `private static final long` **serialVersionUID**
  - For serialization.
- `protected int[]` **updatedLabelIndices**
  - Array of updated label indices.
- `protected weka.core.Instances` **template**
  - Template to store Instances.
- `protected miml.data.MIMLInstances` **dataset**
  - Original data set of MIMLInstances.

6.12.3.7 Constructors

- **MIMLtoML**

  public MIMLtoML()

6.12.3.8 Methods

- **minimax**

  public static double[] minimax(weka.core.Instances data,**int** attIndex)

  - **Description**
    Get the minimal and maximal value of a certain attribute in a data set.
  - **Parameters**
    - `data` – The data set.
    - `attIndex` – The index of the attribute.
  - **Returns** – double[] containing in position 0 the min value and in position 1 the max value.

- **prepareTemplate**

  protected void prepareTemplate() **throws** java.lang.Exception

  - **Description**
    Prepares a template to perform the transformation from MIMLInstances to MultiLabelInstances. This template includes: the bag label attribute, all attributes in the relational attribute as independent attributes and label attributes. For instance, in the relation above, the resulting template is showed. @relation toy
    @attribute id {bag1,bag2}
    @attribute bag relational
    @attribute f1 numeric
    @attribute f2 numeric
    @attribute f3 numeric
    @end bag

@attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}
@relation template
@attribute id {bag1,bag2}
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
* @attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}

  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformDataset**

  public abstract mulan.data.MultiLabelInstances transformDataset() **throws** java.lang.Exception

  - **Description**
    Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
  - **Returns** – MultiLabelInstances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformDataset**

  public abstract mulan.data.MultiLabelInstances transformDataset(miml.data.MIMLInstances dataset) **throws** java.lang.Exception

  - **Description**
    Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
  - **Parameters**
    - `dataset` – The dataset to be transformed
  - **Returns** – MultiLabelInstances.
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

- **transformInstance**

  public abstract weka.core.Instance transformInstance(miml.data.MIMLBag bag) **throws** java.lang.Exception

  - **Description**
    Transforms `MIMLBag` (6.5.1) into Instance.
  - **Parameters**
    - `bag` – The Bag to be transformed.
  - **Returns** – Instance
  - **Throws**
    - `java.lang.Exception` – To be handled in an upper level.

6.12.4 Class MiniMaxTransformation

Class that performs a miniMaxc transformation to convert a MIMLIntances class to MultiLabelInstances. Each Bag is transformed into a single Instance in which, for each attribute of the bag, its min and max value are included. For instance, For instance, in the relation above, the resulting template is showed. @relation toy
@attribute id {bag1,bag2}
@attribute bag relational
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
@end bag
@attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}
@relation miniMaxTransformation
@attribute id {bag1,bag2}
@attribute f1_min numeric
@attribute f1_max numeric
@attribute f2_min numeric
@attribute f2_max numeric
@attribute f3_min numeric
@attribute f3_max numeric
* @attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}

6.12.4.1 Declaration

public class MiniMaxTransformation
extends miml.transformation.mimlTOml.MIMLtoML

6.12.4.2 Field summary

serialVersionUID For serialization

6.12.4.3 Constructor summary

MiniMaxTransformation()
MiniMaxTransformation(MIMLInstances) Constructor.

6.12.4.4 Method summary

prepareTemplate()
transformDataset()
transformDataset(MIMLInstances)
transformInstance(MIMLBag)
transformInstance(MIMLInstances, MIMLBag)

6.12.4.5 Fields

- `private static final long` **serialVersionUID**
  - For serialization

6.12.4.6 Constructors

- **MiniMaxTransformation**

  public MiniMaxTransformation() **throws** java.lang.Exception

- **MiniMaxTransformation**

  public MiniMaxTransformation(miml.data.MIMLInstances dataset) **throws** java.lang.Exception

  - **Description**
    Constructor.
  - **Parameters**
    * `dataset` – MIMLInstances dataset.
  - **Throws**
    * `java.lang.Exception` – To be handled in an upper level.

6.12.4.7 Methods

- **prepareTemplate**

  protected void prepareTemplate() **throws** java.lang.Exception

  - **Description copied from MIMLtoML** (6.12.3)
    Prepares a template to perform the transformation from MIMLInstances to MultiLabelInstances. This template includes: the bag label attribute, all attributes in the relational attribute as independent attributes and label attributes. For instance, in the relation above, the resulting template is showed. @relation toy
    @attribute id {bag1,bag2}
    @attribute bag relational
    @attribute f1 numeric
    @attribute f2 numeric
    @attribute f3 numeric
    @end bag
    @attribute label1 {0,1}
    @attribute label2 {0,1}
    @attribute label3 {0,1}
    @attribute label4 {0,1}
    @relation template
    @attribute id {bag1,bag2}
    @attribute f1 numeric
    @attribute f2 numeric
    @attribute f3 numeric
    * @attribute label1 {0,1}

@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}

- **Throws**
  - `java.lang.Exception` – To be handled in an upper level.

- **transformDataset**

public abstract mulan.data.MultiLabelInstances transformDataset() **throws** java.lang.Exception

- **Description copied from MIMLtoML** (6.12.3)
  Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
- **Returns** – MultiLabelInstances.
- **Throws**
  - `java.lang.Exception` – To be handled in an upper level.

- **transformDataset**

public abstract mulan.data.MultiLabelInstances transformDataset(miml.data.MIMLInstances dataset) **throws** java.lang.Exception

- **Description copied from MIMLtoML** (6.12.3)
  Transforms `MIMLInstances` (6.5.2) into MultiLabelInstances.
- **Parameters**
  - `dataset` – The dataset to be transformed
- **Returns** – MultiLabelInstances.
- **Throws**
  - `java.lang.Exception` – To be handled in an upper level.

- **transformInstance**

public abstract weka.core.Instance transformInstance(miml.data.MIMLBag bag) **throws** java.lang.Exception

- **Description copied from MIMLtoML** (6.12.3)
  Transforms `MIMLBag` (6.5.1) into Instance.
- **Parameters**
  - `bag` – The Bag to be transformed.
- **Returns** – Instance
- **Throws**
  - `java.lang.Exception` – To be handled in an upper level.

- **transformInstance**

public weka.core.Instance transformInstance(miml.data.MIMLInstances dataset,miml.data.MIMLBag bag) **throws** java.lang.Exception

6.13 Package miml.classifiers.miml.lazy

Package Contents *Page*

6.13.1 Class BRkNN__MIMLWrapper

Wrapper for BRkNN of Mulan Library. BRkNN is the simple BR implementation of the KNN algorithm. For more information, see

Eleftherios Spyromitros, Grigorios Tsoumakas, Ioannis Vlahavas: An Empirical Study of Lazy Multilabel Classification Algorithms. In: Proc. 5th Hellenic Conference on Artificial Intelligence (SETN 2008), 2008.

6.13.1.1 Declaration

public class BRkNN__MIMLWrapper
extends miml.classifiers.miml.lazy.MultiLabelKNN__MIMLWrapper

6.13.1.2 Field summary

extension The type of extension to be used:

- NONE: Standard BR.

serialVersionUID Generated Serial version UID.

6.13.1.3 Constructor summary

BRkNN_MIMLWrapper() No-arg constructor for xml configuration
BRkNN_MIMLWrapper(DistanceFunction_MIMLWrapper) Default constructor.
BRkNN_MIMLWrapper(DistanceFunction_MIMLWrapper, int) A constructor that sets the number of neighbors.
BRkNN_MIMLWrapper(DistanceFunction_MIMLWrapper, int, BRkNN.ExtensionType) Constructor giving the option to select an extension of the base version.

6.13.1.4 Method summary

configure(Configuration)
getExtension() Gets the type of extension to be used (see `BRkNN.ExtensionType`).
setExtension(BRkNN.ExtensionType) Sets the type of extension to be used (see `BRkNN.ExtensionType`).

6.13.1.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.
- `private mulan.classifier.lazy.BRkNN.ExtensionType` **extension**
  - The type of extension to be used:
    * NONE: Standard BR.
    * EXTA: Predict top ranked label in case of empty prediction set.
    * EXTB: Predict top n ranked labels based on size of labelset in neighbors.

6.13.1.6 Constructors

- **BRkNN_MIMLWrapper**

  public BRkNN_MIMLWrapper()

  - **Description**
    No-arg constructor for xml configuration
- **BRkNN_MIMLWrapper**

  public BRkNN_MIMLWrapper(DistanceFunction_MIMLWrapper metric)

  - **Description**
    Default constructor.
  - **Parameters**
    * `metric` – The distance metric between bags considered by the classifier.
- **BRkNN_MIMLWrapper**

  public BRkNN_MIMLWrapper(DistanceFunction_MIMLWrapper metric,**int** numOfNeighbors)

- **Description**
  A constructor that sets the number of neighbors.
- **Parameters**
  - `metric` – The distance metric between bags considered by the classifier.
  - `numOfNeighbors` – the number of neighbors.

- **BRkNN_MIMLWrapper**

public BRkNN_MIMLWrapper(DistanceFunction_MIMLWrapper metric,**int** numOfNeighbors, mulan.classifier.lazy.BRkNN.ExtensionType ext)

- **Description**
  Constructor giving the option to select an extension of the base version.
- **Parameters**
  - `metric` – The distance metric between bags considered by the classifier.
  - `numOfNeighbors` – the number of neighbors
  - `ext` – the extension to use (see `BRkNN.ExtensionType`).

6.13.1.7 Methods

- **configure**

public void configure(org.apache.commons.configuration2.Configuration configuration)

- **getExtension**

public mulan.classifier.lazy.BRkNN.ExtensionType getExtension()

- **Description**
  Gets the type of extension to be used (see `BRkNN.ExtensionType`).
- **Returns** – extension Extension to be used

- **setExtension**

public void setExtension(mulan.classifier.lazy.BRkNN.ExtensionType extension)

- **Description**
  Sets the type of extension to be used (see `BRkNN.ExtensionType`).
- **Parameters**
  - `extension` – The new value of the type of extension.

6.13.2 Class DistanceFunction__MIMLWrapper

6.13.2.1 Declaration

public class DistanceFunction__MIMLWrapper
extends java.lang.Object **implements** java.io.Serializable, weka.core.DistanceFunction

6.13.2.2 Field summary

metric Metric to measure distance between bags.
serialVersionUID Wrapper for using IDistance metrics of MIML package with Mulan Lazy algorithms.

6.13.2.3 Constructor summary

DistanceFunction__MIMLWrapper(IDistance) Constructor that sets the metric to be used.

6.13.2.4 Method summary

distance(Instance, Instance)
distance(Instance, Instance, double)
distance(Instance, Instance, double, PerformanceStats)
distance(Instance, Instance, PerformanceStats)
getAttributeIndices()
getInstances()
getInvertSelection()
getOptions()
listOptions()
postProcessDistances(double[])
setAttributeIndices(String)
setInstances(Instances)
setInvertSelection(boolean)
setMetric(IDistance) Sets the metric to be used.
setOptions(String[])
update(Instance)

6.13.2.5 Fields

- `private static final long` **serialVersionUID**
  – Wrapper for using IDistance metrics of MIML package with Mulan Lazy algorithms.
- `miml.core.distance.IDistance` **metric**
  – Metric to measure distance between bags.

6.13.2.6 Constructors

- **DistanceFunction_MIMLWrapper**

  public DistanceFunction_MIMLWrapper(miml.core.distance.IDistance metric)

  - **Description**
    Constructor that sets the metric to be used.
  - **Parameters**
    * `metric` – The metric to be used.

6.13.2.7 Methods

- **distance**

  double distance(weka.core.Instance arg0,weka.core.Instance arg1)

- **distance**

  double distance(weka.core.Instance arg0,weka.core.Instance arg1,**double** arg2)

- **distance**

  double distance(weka.core.Instance arg0,weka.core.Instance arg1,**double** arg2,weka.core.neighboursearch.PerformanceStats arg3)

- **distance**

  double distance(weka.core.Instance arg0,weka.core.Instance arg1,weka.core.neighboursearch.PerformanceStats arg2) **throws** java.lang.Exception

- **getAttributeIndices**

  java.lang.String getAttributeIndices()

- **getInstances**

  weka.core.Instances getInstances()

- **getInvertSelection**

  boolean getInvertSelection()

- **getOptions**

  public java.lang.String[] getOptions()

- **listOptions**

public java.util.Enumeration listOptions()

- **postProcessDistances**

  void postProcessDistances(**double**[] arg0)

- **setAttributeIndices**

  void setAttributeIndices(java.lang.String arg0)

- **setInstances**

  void setInstances(weka.core.Instances arg0)

- **setInvertSelection**

  void setInvertSelection(**boolean** arg0)

- **setMetric**

  public void setMetric(miml.core.distance.IDistance metric)

  – **Description**
    Sets the metric to be used.
  – **Parameters**
    ∗ `metric` – The metric to be used.

- **setOptions**

  public void setOptions(java.lang.String[] arg0) **throws** java.lang.Exception

- **update**

  void update(weka.core.Instance arg0)

6.13.3 Class DMLkNN_MIMLWrapper

Wrapper for DMkKNN (Dependent Multi-Label k Nearest Neighbours) algorighm of Mulan Library. For more information, see *Zoulficar Younes, Fahed Abdallah, Thierry Denceaux (2008). Multi-label classification algorithm derived from k-nearest neighbor rule with label dependencies. In Proceedings of 16th European Signal Processing Conference (EUSIPCO 2008), Lausanne, Switzerland.*

6.13.3.1 Declaration

public class DMLkNN_MIMLWrapper
extends miml.classifiers.miml.lazy.MultiLabelKNN_MIMLWrapper

6.13.3.2 Field summary

serialVersionUID Generated Serial version UID.
smooth Smoothing parameter controlling the strength of uniform prior
(Default value is set to 1 which yields the Laplace smoothing).

6.13.3.3 Constructor summary

DMLkNN_MIMLWrapper() No-arg constructor for xml configuration
DMLkNN_MIMLWrapper(DistanceFunction_MIMLWrapper) Default constructor.
DMLkNN_MIMLWrapper(int, DistanceFunction_MIMLWrapper) A constructor that sets the number of neighbors.
DMLkNN_MIMLWrapper(int, double, DistanceFunction_MIMLWrapper) A constructor that sets the number of neighbors and the value of smooth.

6.13.3.4 Method summary

configure(Configuration)
getSmooth() Gets the smooth factor considered by the classifier.
setSmooth(double) Sets the smooth factor considered by the classifier.

6.13.3.5 Fields

- `private static final long` **serialVersionUID**
  – Generated Serial version UID.
- `protected double` **smooth**
  – Smoothing parameter controlling the strength of uniform prior (Default value is set to 1 which yields the Laplace smoothing).

6.13.3.6 Constructors

- **DMLkNN_MIMLWrapper**

  public DMLkNN_MIMLWrapper()

  – **Description**
    No-arg constructor for xml configuration
- **DMLkNN_MIMLWrapper**

  public DMLkNN_MIMLWrapper(DistanceFunction_MIMLWrapper metric)

  – **Description**
    Default constructor.
  – **Parameters**
    * `metric` – The distance metric between bags considered by the classifier.
- **DMLkNN_MIMLWrapper**

  public DMLkNN_MIMLWrapper(**int** numOfNeighbors,DistanceFunction_MIMLWrapper metric)

  – **Description**
    A constructor that sets the number of neighbors.

- **Parameters**
  - `metric` – The distance metric between bags considered by the classifier.
  - `numOfNeighbors` – The number of neighbors.

- **DMLkNN_MIMLWrapper**

  public DMLkNN_MIMLWrapper(**int** numOfNeighbors,**double** smooth, DistanceFunction_MIMLWrapper metric)

  - **Description**
    A constructor that sets the number of neighbors and the value of smooth.
  - **Parameters**
    - `metric` – The distance metric between bags considered by the classifier.
    - `numOfNeighbors` – The number of neighbors.
    - `smooth` – The smooth factor.

6.13.3.7 Methods

- **configure**

  public void configure(org.apache.commons.configuration2.Configuration configuration)

- **getSmooth**

  public double getSmooth()

  - **Description**
    Gets the smooth factor considered by the classifier.
  - **Returns** – the smooth factor

- **setSmooth**

  public void setSmooth(**double** smooth)

  - **Description**
    Sets the smooth factor considered by the classifier.
  - **Parameters**
    - `smooth` – the new smooth factor

6.13.4 Class IBLR_ML_MIMLWrapper

Wrapper for IBLR-ML and IBLR-ML+ methods of Mulan Library. For more information, see

Weiwei Cheng, Eyke Hullermeier (2009). Combining instance-based learning and logistic regression for multilabel classification. Machine Learning. 76(2-3):211-225.

6.13.4.1 Declaration

public class IBLR_ML_MIMLWrapper
extends miml.classifiers.miml.lazy.MultiLabelKNN_MIMLWrapper

6.13.4.2 Field summary

addFeatures By default, IBLR-ML is used (addFeatures is false).
serialVersionUID Generated Serial version UID.

6.13.4.3 Constructor summary

IBLR_ML_MIMLWrapper() No-arg constructor for xml configuration
IBLR_ML_MIMLWrapper(DistanceFunction_MIMLWrapper) Default constructor.
IBLR_ML_MIMLWrapper(int, boolean, DistanceFunction_MIMLWrapper) A constructor that sets the number of neighbors and whether IBLR-ML or IBLR-ML+ is used.
IBLR_ML_MIMLWrapper(int, DistanceFunction_MIMLWrapper) A constructor that sets the number of neighbors.

6.13.4.4 Method summary

configure(Configuration)
getAddFeatures() Gets the value of addFeatures.
setAddFeatures(boolean) Sets the value of AddFeatures.

6.13.4.5 Fields

- `private static final long` **serialVersionUID**
  – Generated Serial version UID.
- `private boolean` **addFeatures**
  – By default, IBLR-ML is used (addFeatures is false). One can change to IBLR-ML+ through the constructor.

6.13.4.6 Constructors

- **IBLR_ML_MIMLWrapper**

  public IBLR_ML_MIMLWrapper()

  – **Description**
  No-arg constructor for xml configuration
- **IBLR_ML_MIMLWrapper**

  public IBLR_ML_MIMLWrapper(DistanceFunction_MIMLWrapper metric)

- **Description**
  Default constructor.
- **Parameters**
  - `metric` – The distance metric between bags considered by the classifier.

- **IBLR_ML_MIMLWrapper**

public IBLR_ML_MIMLWrapper(**int** numOfNeighbors,**boolean** addFeatures, DistanceFunction_MIMLWrapper metric)

  - **Description**
    A constructor that sets the number of neighbors and whether IBLR-ML or IBLR-ML+ is used.
  - **Parameters**
    - `metric` – The distance metric between bags considered by the classifier.
    - `numOfNeighbors` – The number of neighbors.
    - `addFeatures` – If false IBLR-ML is used. If true, IBLR-ML+ is used.

- **IBLR_ML_MIMLWrapper**

public IBLR_ML_MIMLWrapper(**int** numOfNeighbors,DistanceFunction_MIMLWrapper metric)

  - **Description**
    A constructor that sets the number of neighbors.
  - **Parameters**
    - `metric` – The distance metric between bags considered by the classifier.
    - `numOfNeighbors` – The number of neighbors.

6.13.4.7 Methods

- **configure**

public void configure(org.apache.commons.configuration2.Configuration configuration)

- **getAddFeatures**

public boolean getAddFeatures()

  - **Description**
    Gets the value of addFeatures. If false IBLR-ML is used. If true, IBLR-ML+ is used.
  - **Returns** – The value of addFeatures.

- **setAddFeatures**

public void setAddFeatures(**boolean** addFeatures)

  - **Description**
    Sets the value of AddFeatures. If false IBLR-ML is used. If true, IBLR-ML+ is used.
  - **Parameters**
    - `addFeatures` – The new value of addFeatures.

6.13.5 Class MIMLkNN

Class implementing the MIMLkNN algorithm for MIML data. For more information, see *Zhang, M. L. (2010, October). A k-nearest neighbor based multi-instance multi-label learning algorithm. In 2010 22nd IEEE International Conference on Tools with Artificial Intelligence (Vol.2, pp. 207-212). IEEE.*

6.13.5.1 Declaration

public class MIMLkNN
 extends miml.classifiers.miml.MIMLClassifier

6.13.5.2 Field summary

d_size Dataset size (number of bags).
dataset MIML data.
distance_matrix Distance matrix between dataset's instances.
metric Metric for measure the distance between bags.
num_citers Number of citers.
num_references Number of references.
phi_matrix The phi matrix.
ref_matrix Instances' references matrix.
serialVersionUID Generated Serial version UID.
t_matrix The t matrix.
weights_matrix Weights matrix.

6.13.5.3 Constructor summary

MIMLkNN() No-argument constructor for xml configuration.
MIMLkNN(IDistance) Instantiates a new MIMLkNN with values by default except distance metric.
MIMLkNN(int, int, IDistance) Basic constructor to initialize the classifier.

6.13.5.4 Method summary

buildInternal(MIMLInstances)
calculateBagReferences(int) Calculate the references of a bag specified by its index.
calculateDatasetDistances() Calculate the distances matrix of current data set with the metric assigned.
calculateRecordLabel(Integer[]) Calculate the number of times each label appears in the bag's neighborhood.
calculateReferenceMatrix() Calculate the references matrix.
configure(Configuration)
getBagLabels(int) Gets the labels of specified bag.
getCiters(int) Calculate and return the citers of a bag specified by its index.
getNumCiters() Returns the number of citers considered to estimate the class prediction of tests bags.
getNumReferences() Returns the number of references considered to estimate the class prediction of tests bags.
getReferences(int) Gets the references of a specified bag.

getUnionNeighbors(int) Gets the union of references and citers (without repetitions) of the bag specified.
getWeightsMatrix() Calculate the weights matrix used for prediction.
linearClassifier(double[], double[]) Classifier that decides if a example belong to a specified label.
makePredictionInternal(MIMLBag)
setNumCiters(int) Sets the number of citers considered to estimate the class prediction of tests bags.
setNumReferences(int) Sets the number of references considered to estimate the class prediction of tests bags.

6.13.5.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.
- `protected int` **num_citers**
  - Number of citers.
- `protected int` **num_references**
  - Number of references.
- `protected miml.core.distance.IDistance` **metric**
  - Metric for measure the distance between bags.
- `protected miml.data.MIMLInstances` **dataset**
  - MIML data.
- `int` **d_size**
  - Dataset size (number of bags).
- `protected double[][]` **distance_matrix**
  - Distance matrix between dataset's instances.
- `protected int[][]` **ref_matrix**
  - Instances' references matrix.
- `protected double[][]` **weights_matrix**
  - Weights matrix.
- `protected double[][]` **t_matrix**
  - The t matrix.
- `protected double[][]` **phi_matrix**
  - The phi matrix.

6.13.5.6 Constructors

- **MIMLkNN**

  public MIMLkNN()

  – **Description**
    No-argument constructor for xml configuration.

- **MIMLkNN**

  public MIMLkNN(miml.core.distance.IDistance metric)

  – **Description**
    Instantiates a new MIMLkNN with values by default except distance metric.
  – **Parameters**
    * `metric` – The metric used by the algorithm to measure the distance.

- **MIMLkNN**

  public MIMLkNN(**int** num_references,**int** num_citers,miml.core.distance.IDistance metric)

  – **Description**
    Basic constructor to initialize the classifier.
  – **Parameters**
    * `num_references` – The number of references considered by the algorithm.
    * `num_citers` – The number of citers considered by the algorithm.
    * `metric` – The metric used by the algorithm to measure the distance.

6.13.5.7 Methods

- **buildInternal**

  protected void buildInternal(miml.data.MIMLInstances trainingSet) **throws** java.lang.Exception

  – **See also**
    * `miml.classifiers.miml.MIMLClassifier.buildInternal(MIMLInstances)`

- **calculateBagReferences**

  protected int[] calculateBagReferences(**int** indexBag) **throws** java.lang.Exception

  – **Description**
    Calculate the references of a bag specified by its index. It's necessary calculate the distance matrix previously.
  – **Parameters**
    * `indexBag` – The index bag.
  – **Returns** – The references' indices of the bag.

  - **Throws**
    - `java.lang.Exception` – A exception.

- **calculateDatasetDistances**

  protected void calculateDatasetDistances() **throws** java.lang.Exception

  - **Description**
    Calculate the distances matrix of current data set with the metric assigned.
  - **Throws**
    - `java.lang.Exception` – The exception.

- **calculateRecordLabel**

  protected double[] calculateRecordLabel(java.lang.Integer[] indices)

  - **Description**
    Calculate the number of times each label appears in the bag's neighborhood.
  - **Parameters**
    - `indices` – The neighboor's indices.
  - **Returns** – The labels' record.

- **calculateReferenceMatrix**

  protected void calculateReferenceMatrix() **throws** java.lang.Exception

  - **Description**
    Calculate the references matrix.
  - **Throws**
    - `java.lang.Exception` – the exception

- **configure**

  public void configure(org.apache.commons.configuration2.Configuration configuration)

- **getBagLabels**

  protected double[] getBagLabels(**int** bagIndex)

  - **Description**
    Gets the labels of specified bag.
  - **Parameters**
    - `bagIndex` – The bag index.
  - **Returns** – The bag labels.

- **getCiters**

  protected int[] getCiters(**int** indexBag)

  - **Description**
    Calculate and return the citers of a bag specified by its index. It's necessary calculate the distance matrix first.
  - **Parameters**
    * `indexBag` – The index bag.
  - **Returns** – The bag's citers.

- **getNumCiters**

  public int getNumCiters()

  - **Description**
    Returns the number of citers considered to estimate the class prediction of tests bags.
  - **Returns** – The num citers.

- **getNumReferences**

  public int getNumReferences()

  - **Description**
    Returns the number of references considered to estimate the class prediction of tests bags.
  - **Returns** – The num references.

- **getReferences**

  protected int[] getReferences(**int** indexBag)

  - **Description**
    Gets the references of a specified bag.
  - **Parameters**
    * `indexBag` – The index bag.
  - **Returns** – The bag's references.

- **getUnionNeighbors**

  protected java.lang.Integer[] getUnionNeighbors(**int** indexBag)

  - **Description**
    Gets the union of references and citers (without repetitions) of the bag specified.
  - **Parameters**
    * `indexBag` – The index bag.
  - **Returns** – Ihe union of references and citers.

- **getWeightsMatrix**

  protected double[][] getWeightsMatrix()

- **Description**
  Calculate the weights matrix used for prediction.
- **Returns** – The weights matrix.

- **linearClassifier**

  protected boolean linearClassifier(**double**[] weights,**double**[] record)

  - **Description**
    Classifier that decides if a example belong to a specified label. It is going to depend of the label weights for that bag and the labels' record of bag's neighbors.
  - **Parameters**
    * `weights` – The weights correspondent to the label.
    * `record` – The labels' record of bag's neighbor to be predicted.
  - **Returns** – True, if belong to a determinated class, false if not.

- **makePredictionInternal**

  protected abstract mulan.classifier.MultiLabelOutput makePredictionInternal(miml.data.MIMLBag instance) **throws** java.lang.Exception, mulan.classifier.InvalidDataException

  - **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)
    Learner specific implementation for predicting on specified data based on trained model. This method is called from `makePrediction(Instance)` which guards for model initialization and apply common handling/behavior.
  - **Parameters**
    * `instance` – The data instance to predict on.
  - **Returns** – The output of the learner for the given instance.
  - **Throws**
    * `java.lang.Exception` – If an error occurs while making the prediction.
    * `mulan.classifier.InvalidDataException` – If specified instance data is invalid and can not be processed by the learner.

- **setNumCiters**

  public void setNumCiters(**int** numCiters)

  - **Description**
    Sets the number of citers considered to estimate the class prediction of tests bags.
  - **Parameters**
    * `numCiters` – The new num citers.

- **setNumReferences**

  public void setNumReferences(**int** numReferences)

  - **Description**
    Sets the number of references considered to estimate the class prediction of tests bags.
  - **Parameters**
    * `numReferences` – The new num references.

6.13.6 Class MLkNN_MIMLWrapper

Wrapper for ML-kNN (Multi-Label k Nearest Neighbours) algorithm of Mulan Library. For more information, see:

Min-Ling Zhang, Zhi-Hua Zhou (2007). ML-KNN: A lazy learning approach to multi-label learning. Pattern Recogn.. 40(7):2038–2048.

6.13.6.1 Declaration

public class MLkNN_MIMLWrapper
extends miml.classifiers.miml.lazy.MultiLabelKNN_MIMLWrapper

6.13.6.2 Field summary

serialVersionUID Generated Serial version UID.
smooth Smooth factor

6.13.6.3 Constructor summary

MLkNN_MIMLWrapper() No-arg constructor for xml configuration
MLkNN_MIMLWrapper(DistanceFunction_MIMLWrapper) Default constructor.
MLkNN_MIMLWrapper(int, DistanceFunction_MIMLWrapper) A constructor that sets the number of neighbors.
MLkNN_MIMLWrapper(int, double, DistanceFunction_MIMLWrapper) A constructor that sets the number of neighbors and the value of smooth.

6.13.6.4 Method summary

configure(Configuration)
getSmooth() Gets the smooth factor considered by the classifier.
setSmooth(double) Sets the smooth factor considered by the classifier.

6.13.6.5 Fields

- `private static final long` **serialVersionUID**
  – Generated Serial version UID.
- `protected double` **smooth**
  – Smooth factor

6.13.6.6 Constructors

- **MLkNN_MIMLWrapper**

  public MLkNN_MIMLWrapper()

  - **Description**
    No-arg constructor for xml configuration

- **MLkNN_MIMLWrapper**

  public MLkNN_MIMLWrapper(DistanceFunction_MIMLWrapper metric)

  - **Description**
    Default constructor.
  - **Parameters**
    - `metric` – The distance metric between bags considered by the classifier.

- **MLkNN_MIMLWrapper**

  public MLkNN_MIMLWrapper(**int** numOfNeighbors,DistanceFunction_MIMLWrapper metric)

  - **Description**
    A constructor that sets the number of neighbors.
  - **Parameters**
    - `metric` – The distance metric between bags considered by the classifier.
    - `numOfNeighbors` – The number of neighbors.

- **MLkNN_MIMLWrapper**

  public MLkNN_MIMLWrapper(**int** numOfNeighbors,**double** smooth,
  DistanceFunction_MIMLWrapper metric)

  - **Description**
    A constructor that sets the number of neighbors and the value of smooth.
  - **Parameters**
    - `metric` – The distance metric between bags considered by the classifier.
    - `numOfNeighbors` – The number of neighbors.
    - `smooth` – The smooth factor.

6.13.6.7 Methods

- **configure**

  public void configure(org.apache.commons.configuration2.Configuration configuration)

- **getSmooth**

public double getSmooth()

- **Description**
  Gets the smooth factor considered by the classifier.
- **Returns** – the smooth factor

• **setSmooth**

public void setSmooth(**double** smooth)

- **Description**
  Sets the smooth factor considered by the classifier.
- **Parameters**
  - `smooth` – the new smooth factor

6.13.7 Class MultiLabelKNN_MIMLWrapper

Wrapper for clas MultiLabelKNN of Mulan to work with MIML data

6.13.7.1 Declaration

public abstract class MultiLabelKNN_MIMLWrapper
extends miml.classifiers.miml.MIMLClassifier

6.13.7.2 All known subclasses

MLkNN_MIMLWrapper (6.13.6), IBLR_ML_MIMLWrapper (6.13.4), DMLkNN_MIMLWrapper (6.13.3), BRkNN_MIMLWrapper (6.13.1)

6.13.7.3 Field summary

classifier Mulan MultiLabelKNN classifier.
metric Metric for measure the distance between bags.
numOfNeighbors Number of neighbors used in the k-nearest neighbor algorithm.
serialVersionUID For serialization.

6.13.7.4 Constructor summary

MultiLabelKNN_MIMLWrapper() No-arg constructor for xml configuration
MultiLabelKNN_MIMLWrapper(DistanceFunction_MIMLWrapper) Constructor to initialize the classifier.
MultiLabelKNN_MIMLWrapper(DistanceFunction_MIMLWrapper, int) Constructor to initialize the classifier.

6.13.7.5 Method summary

buildInternal(MIMLInstances)
configure(Configuration)
getClassifier()
getMetric() Gets the distance metric considered by the classifier.
getNumOfNeighbors() Gets the number of neigbors considered by the classifier.
makePredictionInternal(MIMLBag)
setClassifier(MultiLabelKNN)
setMetric(DistanceFunction) Sets the distance metric considered by the classifier.
setnumOfNeighbors(int) Sets the number of neigbors considered by the classifier.

6.13.7.6 Fields

- `private static final long` **serialVersionUID**
  – For serialization.
- `protected int` **numOfNeighbors**
  – Number of neighbors used in the k-nearest neighbor algorithm.
- `protected DistanceFunction_MIMLWrapper` **metric**
  – Metric for measure the distance between bags.
- `protected mulan.classifier.lazy.MultiLabelKNN` **classifier**
  – Mulan MultiLabelKNN classifier.

6.13.7.7 Constructors

- **MultiLabelKNN__MIMLWrapper**

  public MultiLabelKNN__MIMLWrapper()

  – **Description**
    No-arg constructor for xml configuration
- **MultiLabelKNN__MIMLWrapper**

  public MultiLabelKNN__MIMLWrapper(DistanceFunction__MIMLWrapper metric)

  – **Description**
    Constructor to initialize the classifier. It sets the numberOfNeighbors to 10
  – **Parameters**
    ∗ `metric` – The metric used by the algorithm to measure the distance between bags.
- **MultiLabelKNN__MIMLWrapper**

  public MultiLabelKNN__MIMLWrapper(DistanceFunction__MIMLWrapper metric,**int**
  numOfNeighbors)

  – **Description**
    Constructor to initialize the classifier. It sets the numOfNeighbors to 10
  – **Parameters**
    ∗ `metric` – The metric used by the algorithm to measure the distance between bags.
    ∗ `numOfNeighbors` – The number of neighbors.

6.13.7.8 Methods

- **buildInternal**

  protected abstract void buildInternal(miml.data.MIMLInstances trainingSet) **throws** java.lang.Exception

  - **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)
    Learner specific implementation of building the model from `MultiLabelInstances` training data set. This method is called from `build(MultiLabelInstances)` method, where behavior common across all learners is applied.
  - **Parameters**
    - `trainingSet` – The training data set.
  - **Throws**
    - `java.lang.Exception` – if learner model was not created successfully.

- **configure**

  public void configure(org.apache.commons.configuration2.Configuration configuration)

- **getClassifier**

  public mulan.classifier.lazy.MultiLabelKNN getClassifier()

- **getMetric**

  public weka.core.DistanceFunction getMetric()

  - **Description**
    Gets the distance metric considered by the classifier.
  - **Returns** – The distance metric.

- **getNumOfNeighbors**

  public int getNumOfNeighbors()

  - **Description**
    Gets the number of neigbors considered by the classifier.
  - **Returns** – the number of neigbors

- **makePredictionInternal**

  protected abstract mulan.classifier.MultiLabelOutput makePredictionInternal(miml.data.MIMLBag instance) **throws** java.lang.Exception, mulan.classifier.InvalidDataException

  - **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)
    Learner specific implementation for predicting on specified data based on trained model. This method is called from `makePrediction(Instance)` which guards for model initialization and apply common handling/behavior.

- **Parameters**
  - `instance` – The data instance to predict on.
- **Returns** – The output of the learner for the given instance.
- **Throws**
  - `java.lang.Exception` – If an error occurs while making the prediction.
  - `mulan.classifier.InvalidDataException` – If specified instance data is invalid and can not be processed by the learner.

- **setClassifier**

  public void setClassifier(mulan.classifier.lazy.MultiLabelKNN classifier)

- **setMetric**

  public void setMetric(weka.core.DistanceFunction metric)

  - **Description**
    Sets the distance metric considered by the classifier.
  - **Parameters**
    - `metric` – The new distance metric.

- **setnumOfNeighbors**

  public void setnumOfNeighbors(**int** numOfNeighbors)

  - **Description**
    Sets the number of neigbors considered by the classifier.
  - **Parameters**
    - `numOfNeighbors` – the new number of neigbors

6.14 Package miml.classifiers.miml.mimlTOml

6.14.1 Class MIMLClassifierToML

Class implementing the degenerative algorithm for MIML data to solve it with ML learning. For more information, see *Zhou, Z. H., & Zhang, M. L. (2007). Multi-instance multi-label learning with application to scene classification. In Advances in neural information processing systems (pp. 1609-1616).*

6.14.1.1 Declaration

public class MIMLClassifierToML
 extends miml.classifiers.miml.MIMLClassifier

6.14.1.2 Field summary

baseClassifier A Generic MultiLabel classifier.
mimlDataset The miml dataset.
serialVersionUID Generated Serial version UID.
transformationMethod The transform method.

6.14.1.3 Constructor summary

MIMLClassifierToML() No-argument constructor for xml configuration.
MIMLClassifierToML(MultiLabelLearner, MIMLtoML) Basic constructor to initialize the classifier.

6.14.1.4 Method summary

buildInternal(MIMLInstances)
configure(Configuration)
makePredictionInternal(MIMLBag)

6.14.1.5 Fields

- `private static final long` **serialVersionUID**
  - Generated Serial version UID.
- `protected mulan.classifier.MultiLabelLearner` **baseClassifier**
  - A Generic MultiLabel classifier.
- `protected miml.transformation.mimlTOml.MIMLtoML` **transformationMethod**
  - The transform method.
- `protected miml.data.MIMLInstances` **mimlDataset**
  - The miml dataset.

6.14.1.6 Constructors

- **MIMLClassifierToML**

  public MIMLClassifierToML()

  - **Description**
    No-argument constructor for xml configuration.
- **MIMLClassifierToML**

public MIMLClassifierToML(mulan.classifier.MultiLabelLearner baseClassifier,miml.transformation.mimlTOml.MIMLtoML transformationMethod) **throws** java.lang.Exception

- **Description**

  Basic constructor to initialize the classifier.
- **Parameters**
  - `baseClassifier` – The base classification algorithm.
  - `transformationMethod` – Algorithm used as transformation method from MIML to ML.
- **Throws**
  - `java.lang.Exception` – To be handled in an upper level.

6.14.1.7 Methods

- **buildInternal**

protected abstract void buildInternal(miml.data.MIMLInstances trainingSet) **throws** java.lang.Exception

- **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)

  Learner specific implementation of building the model from `MultiLabelInstances` training data set. This method is called from `build(MultiLabelInstances)` method, where behavior common across all learners is applied.
- **Parameters**
  - `trainingSet` – The training data set.
- **Throws**
  - `java.lang.Exception` – if learner model was not created successfully.

- **configure**

public void configure(org.apache.commons.configuration2.Configuration configuration)

- **makePredictionInternal**

protected abstract mulan.classifier.MultiLabelOutput makePredictionInternal(miml.data.MIMLBag instance) **throws** java.lang.Exception, mulan.classifier.InvalidDataException

- **Description copied from miml.classifiers.miml.MIMLClassifier** (6.3.2)

  Learner specific implementation for predicting on specified data based on trained model. This method is called from `makePrediction(Instance)` which guards for model initialization and apply common handling/behavior.
- **Parameters**
  - `instance` – The data instance to predict on.
- **Returns** – The output of the learner for the given instance.
- **Throws**
  - `java.lang.Exception` – If an error occurs while making the prediction.
  - `mulan.classifier.InvalidDataException` – If specified instance data is invalid and can not be processed by the learner.

6.15 Package miml.run

Package Contents *Page*

6.15.1 Class RunAlgorithm

Class that allow run any algorithm of the library configured by a file configuration.

6.15.1.1 Declaration

public class RunAlgorithm
extends java.lang.Object

6.15.1.2 Constructor summary

RunAlgorithm()

6.15.1.3 Method summary

main(String[]) The main method to configure and run an algorithm.

6.15.1.4 Constructors

- **RunAlgorithm**

  public RunAlgorithm()

6.15.1.5 Methods

- **main**

  public static void main(java.lang.String[] args)

  - **Description**
    The main method to configure and run an algorithm.
  - **Parameters**
    * `args` – The argument (route of config file with the option -c).

6.16 Package miml.tutorial

Package Contents *Page*

6.16.1 Class CrossValidationExperiment

Class implementing an example of using cross-validation with the 3 different kinds of classifier.

6.16.1.1 Declaration

public class CrossValidationExperiment
 extends java.lang.Object

6.16.1.2 Constructor summary

CrossValidationExperiment()

6.16.1.3 Method summary

main(String[])
showUse() Shows the help on command line.

6.16.1.4 Constructors

- **CrossValidationExperiment**

public CrossValidationExperiment()

6.16.1.5 Methods

- **main**

  public static void main(java.lang.String[] args) **throws** java.lang.Exception

- **showUse**

  public static void showUse()

  - **Description**
    Shows the help on command line.

6.16.2 Class HoldoutExperiment

Class implementing an example of using holdout with train/test dataset and a single dataset applying percentage split.

6.16.2.1 Declaration

public class HoldoutExperiment
extends java.lang.Object

6.16.2.2 Constructor summary

HoldoutExperiment()

6.16.2.3 Method summary

main(String[])
showUse() Shows the help on command line.

6.16.2.4 Constructors

- **HoldoutExperiment**

  public HoldoutExperiment()

6.16.2.5 Methods

- **main**

  public static void main(java.lang.String[] args) **throws** java.lang.Exception

- **showUse**

  public static void showUse()

  - **Description**
    Shows the help on command line.

6.16.3 Class InsertingAttributesToBags

Class implementing an example of inserting a new group of attributes to the relational attribute of the dataset with {0,1} values.

6.16.3.1 Declaration

public class InsertingAttributesToBags
extends java.lang.Object

6.16.3.2 Constructor summary

InsertingAttributesToBags()

6.16.3.3 Method summary

main(String[])
showUse() Shows the help on command line.

6.16.3.4 Constructors

- **InsertingAttributesToBags**

  public InsertingAttributesToBags()

6.16.3.5 Methods

- **main**

  public static void main(java.lang.String[] args) **throws** java.lang.Exception

- **showUse**

  public static void showUse()

  – **Description**
    Shows the help on command line.

6.16.4 Class InsertingAttributeToBag

Class implementing an example of inserting a new attribute to the relational attribute of the dataset with {0,1} values.

6.16.4.1 Declaration

public class InsertingAttributeToBag
extends java.lang.Object

6.16.4.2 Constructor summary

InsertingAttributeToBag()

6.16.4.3 Method summary

main(String[])
showUse() Shows the help on command line.

6.16.4.4 Constructors

- **InsertingAttributeToBag**

  public InsertingAttributeToBag()

6.16.4.5 Methods

- **main**

  public static void main(java.lang.String[] args) **throws** java.lang.Exception

- **showUse**

  public static void showUse()

  – **Description**
    Shows the help on command line.

6.16.5 Class ManagingMIMLInstances

Class implementing basic handling of MIML datasets.

6.16.5.1 Declaration

public class ManagingMIMLInstances
extends java.lang.Object

6.16.5.2 Constructor summary

ManagingMIMLInstances()

6.16.5.3 Method summary

main(String[])
showUse() Shows the help on command line.

6.16.5.4 Constructors

- **ManagingMIMLInstances**

  public ManagingMIMLInstances()

6.16.5.5 Methods

- **main**

  public static void main(java.lang.String[] args)

- **showUse**

  public static void showUse()

  - **Description**
    Shows the help on command line.

6.16.6 Class MIMLtoMITranformation

Class for basic handling of MIML to MIL LP and BR transformation.

6.16.6.1 Declaration

public class MIMLtoMITranformation
extends java.lang.Object

6.16.6.2 Constructor summary

MIMLtoMITranformation()

6.16.6.3 Method summary

main(String[])
showUse() Shows the help on command line.

6.16.6.4 Constructors

- **MIMLtoMITranformation**

  public MIMLtoMITranformation()

6.16.6.5 Methods

- **main**

  public static void main(java.lang.String[] args) **throws** java.lang.Exception

- **showUse**

  public static void showUse()

  - **Description**
    Shows the help on command line.

6.16.7 Class MIMLtoMLTransformation

Class for basic handling of the transformation MIML to ML transformations.

6.16.7.1 Declaration

public class MIMLtoMLTransformation
 extends java.lang.Object

6.16.7.2 Constructor summary

MIMLtoMLTransformation()

6.16.7.3 Method summary

main(String[])
showUse() Shows the help on command line.

6.16.7.4 Constructors

- **MIMLtoMLTransformation**

  public MIMLtoMLTransformation()

6.16.7.5 Methods

- **main**

  public static void main(java.lang.String[] args) **throws** java.lang.Exception

- **showUse**

  public static void showUse()

  - **Description**
    Shows the help on command line.

6.17 Package miml

Package Contents *Page*

6.17.1 Class AppTest

Unit test for simple App.

6.17.1.1 Declaration

public class AppTest
extends junit.framework.TestCase

6.17.1.2 Constructor summary

AppTest(String) Create the test case

6.17.1.3 Method summary

suite()
testApp() Rigourous Test :-)

6.17.1.4 Constructors

- **AppTest**

  public AppTest(java.lang.String testName)

  - **Description**
    Create the test case
  - **Parameters**
    - `testName` – name of the test case

6.17.1.5 Methods

- **suite**

  public static junit.framework.Test suite()

  - **Returns** – the suite of tests being tested

- **testApp**

  public void testApp()

  - **Description**
    Rigourous Test :-)

Bibliography

[1] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2):31–71, 1997.

[2] M.L. Zhang. A k-nearest neighbor based multi-instance multi-label learning algorithm. In *Proceedings of the 22nd International Conference on Tools with Artificial Intelligence*, volume 2, pages 207–212, 2010.

[3] S. Kotsiantis, D. Kanellopoulos, and V. Tampakas. Financial application of multi-instance learning: two greek case studies. *Journal of Convergence Information Technology*, 5(8):42–53, 2010.

[4] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837, 2014.

[5] Eva Gibaja and Sebastián Ventura. A tutorial on multilabel learning. *ACM Computing Surveys (CSUR)*, 47(3):52, 2015.

[6] Concha Bielza, Guangdi Li, and Pedro Larranaga. Multi-dimensional classification with bayesian networks. *International Journal of Approximate Reasoning*, 52(6):705–727, 2011.

[7] Guangcan Liu, Zhouchen Lin, and Yong Yu. Multi-output regression on the output manifold. *Pattern Recognition*, 42(11):2737–2743, 2009.

[8] K. Yan, Z. Li, and C. Zhang. A new multi-instance multi-label learning approach for image and text classification. *Multimedia Tools and Applications*, 75(13):7875–7890, 2016.

[9] C. Tong-tong, L. Chan-juan, Z. Hai-lin, Z. Shu-sen, L. Ying, and D. Xin-miao. A multi-instance multi-label scene classification method based on multi-kernel fusion. In *Proceedings of the Conference on Intelligent Systems*, pages 782–787, 2015.

[10] X.S. Xu, X. Xue, and Z. Zhou. Ensemble multi-instance multi-label learning approach for video annotation task. In *Proceedings of the 19th International Conference on Multimedia*, pages 1153–1156. ACM, 2011.

[11] Y.X. Li, S. Ji, S. Kumar, J. Ye, and Z.H. Zhou. Drosophila gene expression pattern annotation through multi-instance multi-label learning. *Transactions on Computational Biology and Bioinformatics*, 9(1):98–112, 2012.

[12] J.S. Wu, S.J. Huang, and Z.H. Zhou. Genome-wide protein function prediction through multi-instance multi-label learning. *Transactions on Computational Biology and Bioinformatics*, 11(5):891–902, 2014.

[13] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[14] G. Tsoumakas, E. Spyromitros-Xioufis, Jozef Vilcek, and I. Vlahavas. Mulan: A java library for multi-label learning. *Journal Machine Learning Research*, 12:2411–2414, 2011.

[15] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes. Meka: a multi-label/multi-target extension to weka. *Journal of Machine Learning Research*, 17(1):667–671, 2016.

[16] LAMDA. Lamda learning and mining from data. http://www.lamda.nju.edu.cn/Data.ashx, 2019. Accessed: 2020-07-19.

[17] Eva Gibaja and Sebastián Ventura. Multi-label learning: a review of the state of the art and ongoing research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery (WIRES)*, 4(6):411–444, 2014.

[18] Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2-3):135–168, 2000.

[19] Konstantinos Trohidis, Grigorios Tsoumakas, George Kalliris, and Ioannis P Vlahavas. Multi-label classification of music into emotions. In *ISMIR*, volume 8, pages 325–330, 2008.

[20] Min-Ling Zhang and Zhi-Hua Zhou. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, 18(10):1338–1351, 2006.

[21] Artus Krohn-Grimberghe, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. Multi-relational matrix factorization using bayesian personalized ranking for social network data. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 173–182. ACM, 2012.

[22] Yi Zhang, Samuel Burer, and W Nick Street. Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, 7(Jul):1315–1338, 2006.

[23] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *Data mining and knowledge discovery handbook*, pages 667–685. Springer, 2009.

[24] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333, 2011.

[25] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40(7):2038–2048, 2007.

[26] Weiwei Cheng and Eyke Hüllermeier. Combining instance-based learning and logistic regression for multilabel classification. *Machine Learning*, 76(2-3):211–225, 2009.

[27] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089, 2011.

[28] Jesse Read, Bernhard Pfahringer, and Geoff Holmes. Multi-label classification using ensembles of pruned sets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 995–1000. IEEE, 2008.

[29] F. Herrera, S. Ventura., R. Bello, C. Cornelis, A. Zafra, D. Sánchez-Tarragó, and S. Vluymans. *Multiple Instance Learning. Foundations and Algorithms.* Springer, 2016.

[30] Dan Zhang, Fei Wang, Luo Si, and Tao Li. Maximum margin multiple instance clustering with applications to image and text clustering. *IEEE Transactions on Neural Networks*, 22(5):739–751, 2011.

[31] Amelia Zafra, Cristóbal Romero, Sebastián Ventura, and Enrique Herrera-Viedma. Multi-instance genetic programming for web index recommendation. *Expert Systems with Applications*, 36(9):11470–11479, 2009.

[32] Jingxin Xu, Simon Denman, Vikas Reddy, Clinton Fookes, and Sridha Sridharan. Real-time video event detection in crowded scenes using mpeg derived features: A multiple instance learning approach. *Pattern Recognition Letters*, 44:113–125, 2014.

[33] Yan Xu, Jun-Yan Zhu, I Eric, Chao Chang, Maode Lai, and Zhuowen Tu. Weakly supervised histopathology cancer image segmentation and classification. *Medical image analysis*, 18(3):591–604, 2014.

[34] Zhi-Hua Zhou. Multi-instance learning from supervised view. *Journal of Computer Science and Technology*, 21(5):800–809, 2006.

[35] Hsiao-Tien Pao, Shun C Chuang, Yeong-Yuh Xu, and Hsin-Chia Fu. An em based multiple instance learning method for image classification. *Expert Systems with Applications*, 35(3):1468–1472, 2008.

[36] Oded Maron. *Learning from ambiguity.* PhD thesis, Massachusetts Institute of Technology, 1998.

[37] Soumya Ray and Mark Craven. Supervised versus multiple instance learning: An empirical comparison. In *Proceedings of the 22nd international conference on Machine learning*, pages 697–704, 2005.

[38] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. In *NIPS'02: Proceedings of Neural Information Processing Systems*, pages 561–568, Vancouver, Canada,, 2002. MIT Press.

[39] Qingping Tao, Stephen Scott, N. V. Vinodchandran, and Thomas Takeo Osugi. Svm-based generalized multiple-instance learning via approximate box counting. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 101–, New York, NY, USA, 2004. ACM.

[40] Qi Zhang and Sally A Goldman. Em-dd: An improved multiple-instance learning technique. In *Advances in neural information processing systems*, NIPS '01, pages 1073–1080, 2001.

[41] John C. Platt. *Advances in Kernel Methods.* MIT Press, Cambridge, MA, USA, 1999.

[42] Thomas Gärtner, Peter A Flach, Adam Kowalczyk, and Alex J Smola. Multi-instance kernels. In *ICML*, volume 2, pages 179–186, 2002.

[43] Jun Wang and Jean-Daniel Zucker. Solving the multiple-instance problem: A lazy learning approach. In Pat Langley, editor, *Proceedings of the 17th International Conference on Machine Learning*, ICML '00, pages 1119–1126, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[44] Peter Auer and Ronald Ortner. A boosting approach to multiple instance learning. In *15th European Conference on Machine Learning*, pages 63–74. Springer, 2004. LNAI 3201.

[45] Xin Xu and Eibe Frank. Logistic regression and boosting for labeled bags of instances. In *Advances in knowledge discovery and data mining*, pages 272–281. Springer, 2004.

[46] Z.H. Zhou, M.L. Zhang, S.J. Huang, and Y.F. Li. Multi-instance multi-label learning. *Artificial Intelligence*, 176(1):2291–2320, 2012.

[47] Zhi H. Zhou and Min L. Zhang. Multi-instance multi-label learning with application to scene classification. In *NIPS*, pages 1609–1616, 2006.

[48] C. Li and G. Shi. Weights optimization for multi-instance multi-label rbf neural networks using steepest descent method. *Neural Computing and Applications*, 22(7-8):1563–1569, 2013.

[49] Jose M Moyano, Eva L Gibaja, and Sebastián Ventura. Mlda: A tool for analyzing multi-label datasets. *Knowledge-Based Systems (KBS)*, 121:1–3, 2017.

[50] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[51] Forrest Briggs, Yonghong Huang, Raviv Raich, Konstantinos Eftaxias, Zhong Lei, William Cukierski, Sarah Frey Hadley, Adam Hadley, Matthew Betts, Xiaoli Z Fern, et al. The 9th annual mlsp competition: New methods for acoustic classification of multiple simultaneous bird species in a noisy environment. In *2013 IEEE international workshop on machine learning for signal processing (MLSP)*, pages 1–8. IEEE, 2013.

[52] Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pages 570–576, Cambridge, MA, USA, 1998. MIT Press.

[53] Grigorios Tsoumakas, Anastasios Dimou, Eleftherios Spyromitros-Xioufis, V. Mezaris, Ioannis Kompatsiaris, and I. Vlahavas. Correlation-based pruning of stacked binary relevance models for multi-label learning. In *Proceedings of the 1st International Workshop on Learning from Multi-Label Data*, pages 101–116, 01 2009.

[54] Luke Bjerring and Eibe Frank. Beyond trees: Adopting miti to learn rules and ensemble classifiers for multi-instance data. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence*. Springer, 2011.

[55] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems 15*, pages 561–568. MIT Press, 2003.

[56] Hendrik Blockeel, David Page, and Ashwin Srinivasan. Multi-instance tree learning. In *Proceedings of the International Conference on Machine Learning*, pages 57–64. ACM, 2005.

[57] E. T. Frank and X. Xu. Applying propositional learning algorithms to multi-instance data. Technical report, University of Waikato, Department of Computer Science, University of Waikato, Hamilton, NZ, 06 2003.

[58] Lin Dong. A comparison of multi-instance learning algorithms. Master's thesis, The University of Waikato, 2006.

[59] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172:1897–1916, 2008.

[60] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza mencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133 – 153, 2008.

[61] E. Spyromitros, G. Tsoumakas, and I.Vlahavas. An empirical study of lazy multilabel classification algorithms. In *Proc. 5th Hellenic Conference on Artificial Intelligence (SETN 2008)*, 2008.

[62] Z. Younes, F. Abdallah, and T. Denoeux. Multi-label classification algorithm derived from k-nearest neighbor rule with label dependencies. In *2008 16th European Signal Processing Conference*, pages 1–5, 2008.

[63] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, 2008.

[64] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco, 1996. Morgan Kaufmann.

[65] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.