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

User Manual
MIML version 1.3
November 17, 2022

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, the 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 for 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, since each object can belong to several classes, Multi-Task (MT) [4] learning represents the output space more flexibly than the traditional paradigms. Among these approaches, one of the most popular is the so-called Multi-Label (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 the 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 majority of these algorithms are implemented with MATLAB so that a software license is needed to execute them. Besides, they do not integrate into a library, and each algorithm has its specific configuration and input and output formats. This fact seriously complicates 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 that makes it 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 that 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 3.1 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 developing 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 is 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

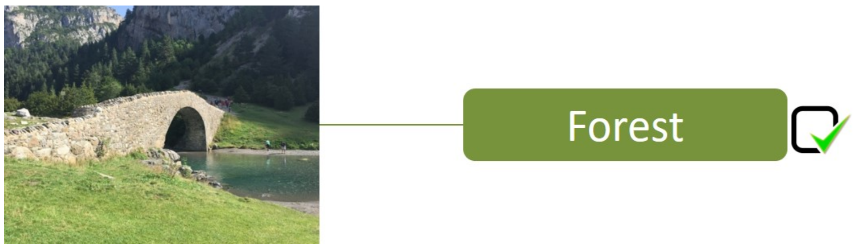


Figure 2.1: Single-label (SL) learning

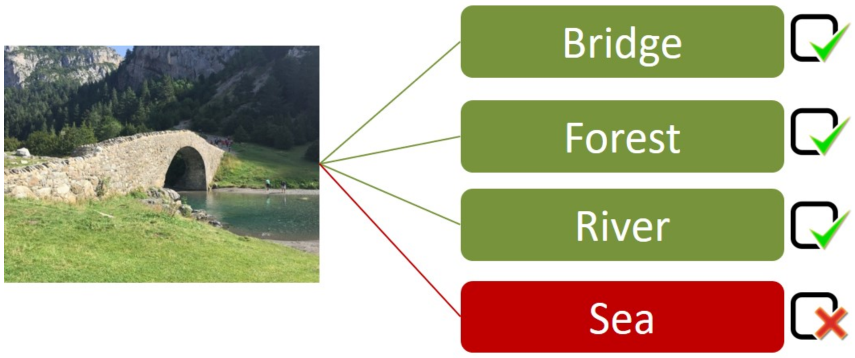


Figure 2.2: Multi-label (ML) learning

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. 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 considering label relationships but with a reasonable computational cost [24]. The second group is composed of algorithms that adapt traditional algorithms to cope directly 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 another category of methods so-called *multi-label ensembles* in which base classifiers are also multi-label classifiers [24] [17]. Many of these methods have yielded high predictive performance. We can cite Random k-lAbELset (RAkEL), which builds an ensemble of LP classifiers through 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 to the need to 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 to solve the problem of modeling 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 representing a pattern through 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 set of regions (instances) and with a class label associated.

In MI learning the aim is learning a function $h_{\mathrm{MI}} : 2^{\mathcal{X}} \to \mathcal{Y}$ 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 $y_i \in \mathcal{Y}$ 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, some algorithms adapt the traditional learning hypothesis to the MI framework [34].

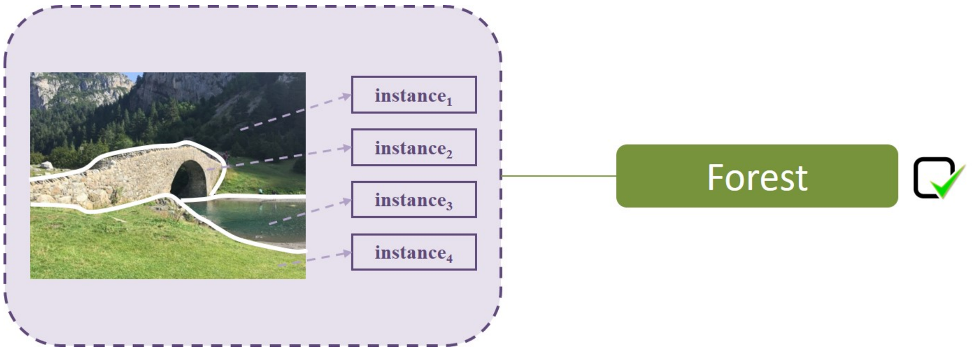


Figure 2.3: Multi-instance (MI) learning

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 is 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 that maximizes diverse density by applying a standard Bayesian classifier by considering bags with a set of instances instead of 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 number 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 with 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 the distance between bags. The Hausdorff and Kullback–Leibler distances 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, which only considers nearest neighbors 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. 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 hypotheses 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 some researchers have applied a hybrid approach to work simultaneously with complex data representation both in input and in output space [34]. In the 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).

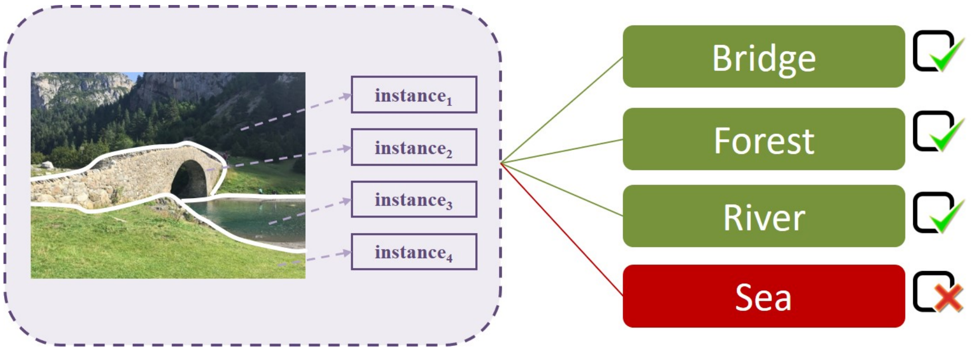


Figure 2.4: Multi-Instance multi-label (MIML) learning

Therefore, in MIML learning the aim is to learn a function $h_{\text{MIML}} : 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ 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 $Y_i \subseteq \mathcal{Y}$ 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, other algorithms address the MIML problem directly.

As MIML learning 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 into MI problem and then the resulting problem is solved by MI algorithms. The second transformation approach consists of transforming the problem to ML problem and 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.

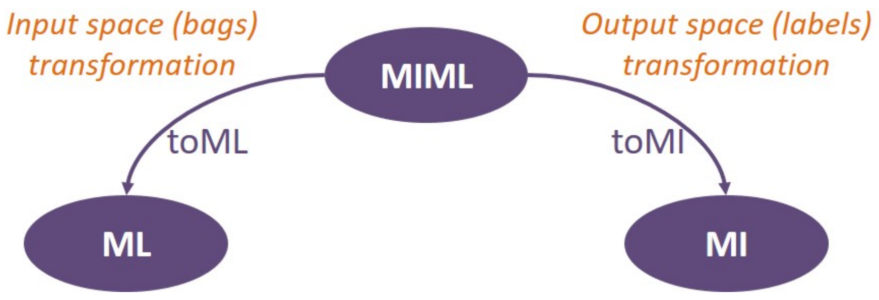

Input space (bags) transformation
Output space (labels) transformation
MIML
toML
toMI
ML
MI

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 loss 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, $D\left\{(X_i, Y_i)|1 \leq i \leq m\right\}$ 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 label sets 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 metrics for MI datasets, such as the 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 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 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 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.

$$0/1 Subset\ accuracy = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} I(Z_i = Y_i)$$

$$Hamming\quad loss = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} \frac{\mid Y_i \Delta\ Z_i \mid}{q}$$

$$Accuracy = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} \frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \cup Z_i \mid}$$

$$Precision = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} \frac{\mid Y_i \cap Z_i \mid}{\mid Z_i \mid}$$

$$Recall = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} \frac{\mid Y_i \cap Z_i \mid}{\mid Y_i \mid}$$

$$F - Measure = 2\frac{precision \cdot recall}{precision + recall}$$

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.

$$one - error = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} [\![arg \min_{\lambda \in Y} \tau_i(\lambda) \notin Y_i]\!]$$

$$coverage = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} \max_{\lambda \in Y_i} \tau_i(\lambda) - 1$$

$$ranking\ loss = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} \frac{1}{|Y_i|\left|\overline{Y_i}\right|} |E| \quad where$$

$$E = \{(\lambda, \lambda')|\tau_i(\lambda) > \tau_i(\lambda'), (\lambda, \lambda') \in Y_i \times \overline{Y_i}\}$$

$$is\ error = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} [\![\sum_{\lambda \in \mathcal{L}} |\tau_i^*(\lambda) - \tau_i(\lambda)| \neq 0]\!]$$

$$avg.\ precision = \frac{1}{\mid T \mid}\sum_{i=1}^{|T|} \frac{1}{|Y_i|} \sum_{\lambda \in Y_i} \frac{|\{\lambda' \in Y_i | \tau_i(\lambda') \leq \tau_i(\lambda)\}|}{\tau_i(\lambda)}$$

Table 2.4: Example-based metrics - rankings

- *Confidences*: Finally, metrics to evaluate confidences can be defined.
  - *Logarithmic Loss*: it punishes more large 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.

$Logarithmic\ Loss = \frac{1}{q \mid T \mid} \sum_{i=1}^{\mid T \mid} \sum_{\lambda \in \gamma} min(-\text{LogLoss}(\lambda, \mathbf{w_i}), \ln(\mid T \mid))$
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

3.1 Getting the library

This section describes the necessary steps to download, install and configure everything you need to use the library with the algorithms included and to start developing your code.

Table 3.1 specifies the versions of dependencies needed by the MIML library.

Dependence	Version
Weka	3.7.10
Mulan	1.5
citationKNN	1.0.1
multiInstanceLearning	1.0.9
multiInstanceFilters	1.0.10
commons-tex	1.6
commons-logging	1.2
commons-lang	3.8.1
commons-configuration	2.4
commons-collections	4.2
commons-beanutils	1.9.3
MWAlgorithms	
javabuilder	

Table 3.1: Versions of dependencies used by the MIML library

There are two different ways to work with the library: 1) executing the *jar* file through a terminal or 2) using a Maven project. The library is released via GitHub, available at `https://github.com/kdis-lab/MIML` where you can find the following folders:

- *documentation*. A folder with the documentation of the library including the user's manual in *pdf* and the Javadoc documentation of the API in *html*.
- *dist*. It contains the distribution of the MIML library as a *jar* file. This option is described in Section 3.2.

- *mavenProject*. It contains the distribution of the MIML library as a Maven project. This option is described in Section 3.3.

From here, it is specified the different steps according to preference to run the library: from *jar* file or using a Maven project illustrated with the Eclipse IDE.

3.2 Using *jar* file

1 Download the MIML library. It is released via GitHub, available at `https://github.com/kdis-lab/MIML`. Here you will find the source code together with configuration files and datasets that you can use in the experimentation.

2 Verify that the following directories and files exist within the main directory:
   - *configurations*. It contains an example of *xml* configuration file for each algorithm included in the library.
   - *data*. It contains some examples of datasets. Particularly, *birds* dataset is included with the distribution of the library.
   - *miml-1.0.jar*. The *jar* file packaged with all its dependencies.

3 The *jar* file contains all the dependencies and can be used as an executable of the library. To run an algorithm given an xml, it is necessary to specify the configuration file path through the 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>`. 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
```

3.3 Using Maven project in Eclipse IDE

1 Download the MIML library. It is released via GitHub, available at `https://github.com/kdis-lab/MIML`. Here you will find the source code together with configuration files and datasets that you can use in the experimentation.

2 Verify that the following directories and files exist within the main directory:
   - *configurations*. It contains an example of *xml* configuration file for each algorithm included in the library.
   - *data*. It contains some examples of datasets. Particularly, *birds* dataset is included with the distribution of the library.
   - *results*. It contains the file reports generated by each example in the configurations folder.
   - *src*. It contains the source code of the distribution.
   - *pom.xml*. File to load the Maven project.

3 Copy the Maven project to your Eclipse workspace.

4 Import the Maven project. To do this, select *File->Import* and then choose *Existing Maven project* (see Figure 3.1). Then, choose the path where the Maven project with the *pom.xml* file were copied (see Figure 3.2).

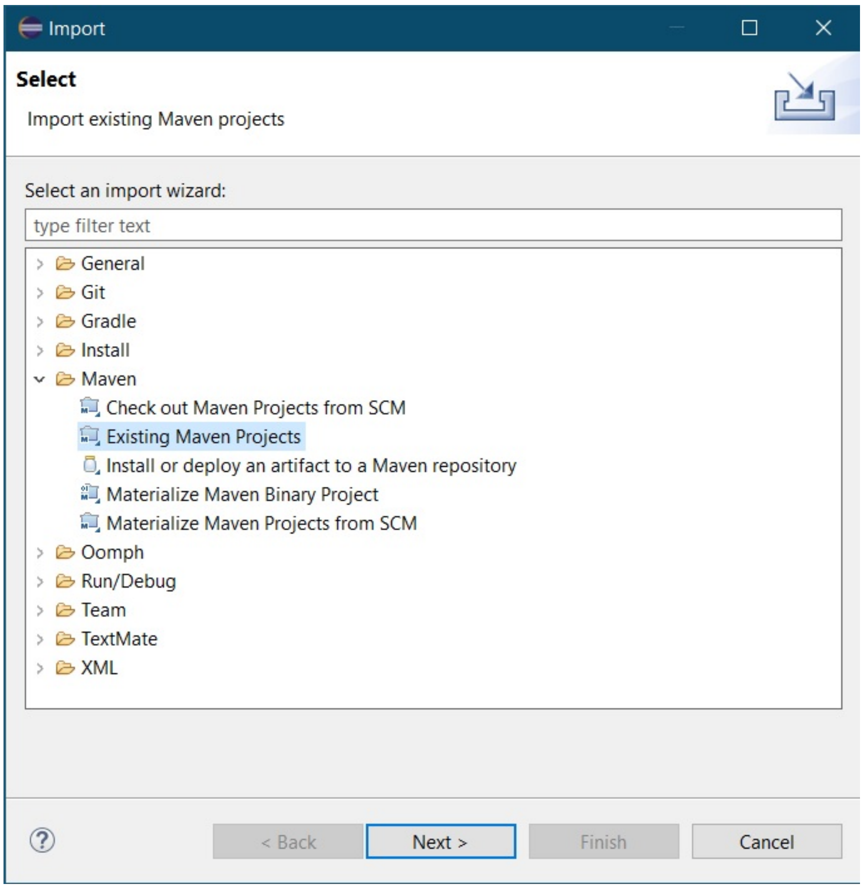
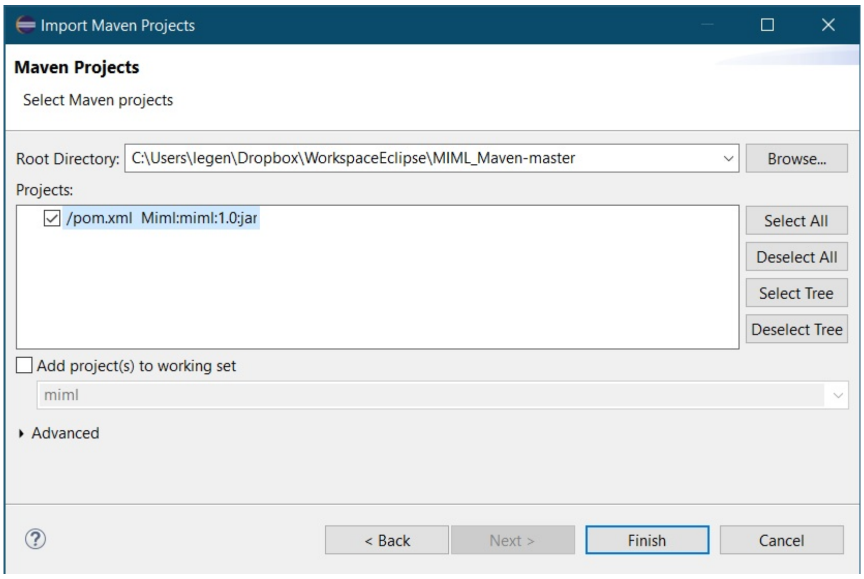

Import
Select
Import existing Maven projects
Select an import wizard:
type filter text
General
Git
Gradle
Install
Maven
Check out Maven Projects from SCM
Existing Maven Projects
Install or deploy an artifact to a Maven repository
Materialize Maven Binary Project
Materialize Maven Projects from SCM
Oomph
Run/Debug
Team
TextMate
XML
< Back
Next >
Finish
Cancel

Figure 3.1: Importing a Maven project in Eclipse IDE

5 The project itself has referenced all the required dependencies. Therefore, it is not necessary to 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 *xml* file path used in the experiment through the option *-c* (see Figure 3.3). Below, an example to execute a configuration file named *MIMLkNN.config* is presented:

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


Import Maven Projects

Maven Projects

Select Maven projects

Root Directory: C:\Users\legen\Dropbox\WorkspaceEclipse\MIML_Maven-master

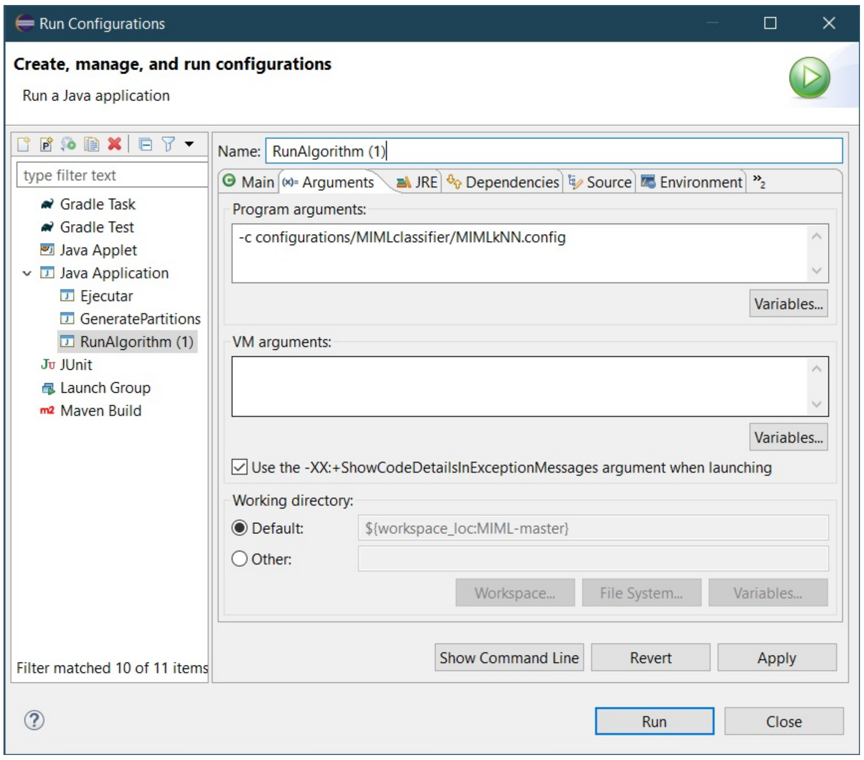
Projects:

☑ /pom.xml Miml:miml:1.0:jar

☐ Add project(s) to working set

miml

Figure 3.2: Selecting a Maven project in Eclipse IDE


Run Configurations

Create, manage, and run configurations

Run a Java application

Name: RunAlgorithm (1)

Program arguments:

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

VM arguments:

[✓] Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching

Working directory:

(●) Default: ${workspace_loc:MIML-master}

( ) Other:

Filter matched 10 of 11 items

Figure 3.3: Configuring the program arguments in Eclipse IDE

CHAPTER

4

MIML library

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

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

2. It includes a set of MIML algorithms. Concretely, 43 algorithms are included in the library. Both methods that transform previously the problem and methods that directly solve the MIML problem are considered.

3. Algorithms can be easily used and executed using *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.

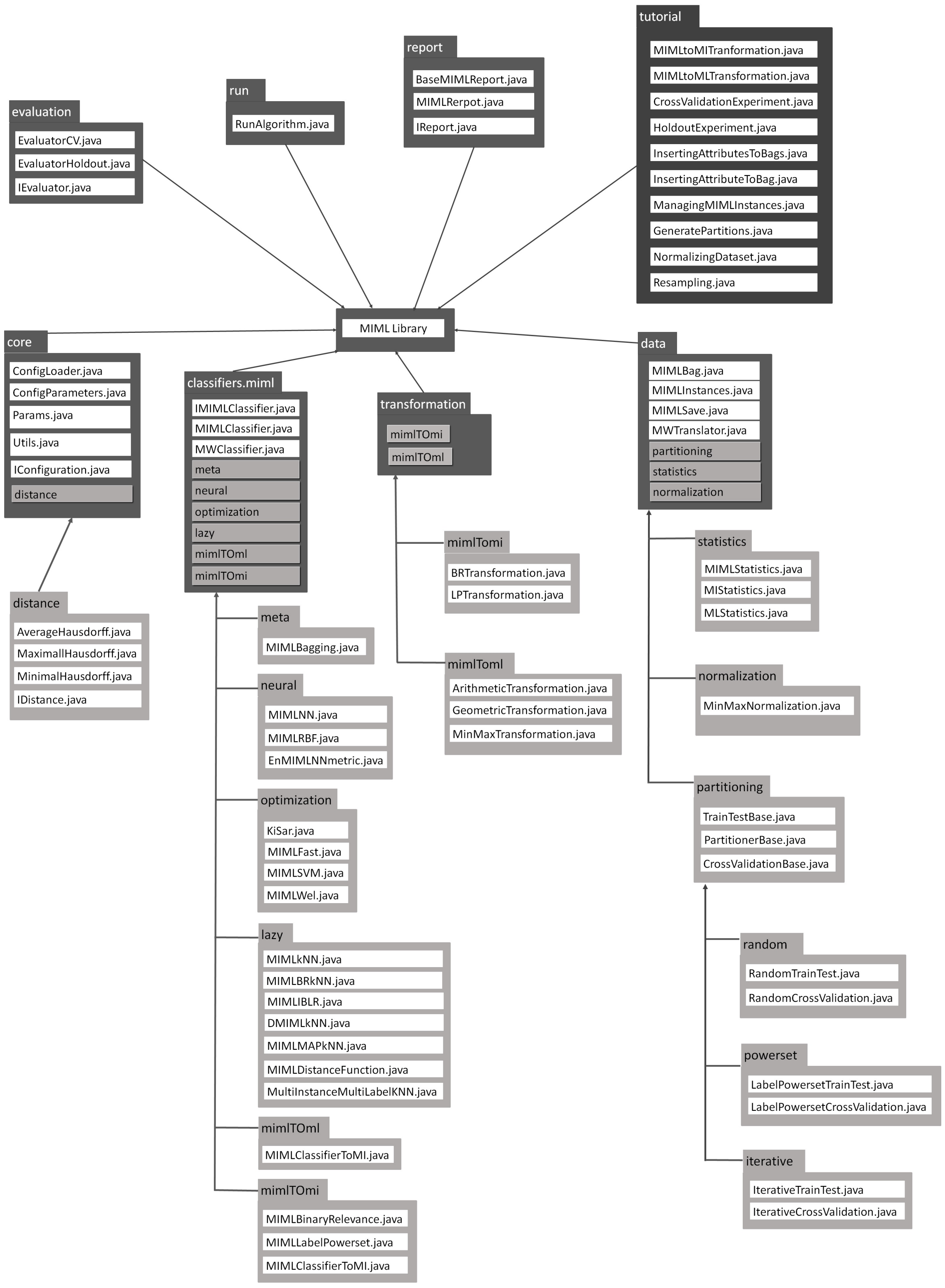


Figure 4.1: Architecture MIML library

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. Figure 4.1 shows a class diagram with the main classes of the library and their relationships. 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 4.2.1. Therefore, classes in this package allow to load a MIML dataset and obtain properties about data such as the number of attributes, the number of bags, the number of labels, etc. Besides, it is possible to access a bag, its instances and labels. Finally, this package includes some preprocessing facilities such as normalization and data partitioning.
- *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 label sets, 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.
- *evaluation.* It contains facilities to run and evaluate a classification algorithm. Particularly it is possible to perform holdout and cross-validation experiments.
- *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 employing a *xml* file.
- *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 previously transforming it to MI problem and then applying any MI classification algorithm. The library considers LP and BR transformations as they are widely used in the multi-label learning framework. Table 4.1 shows 15 Weka MI algorithms that can be used with these transformation methods[1]. Section 4.4.1 includes examples of the execution of each algorithm, showing their configuration file and the results obtained.
- *classifiers.miml.mimlTOml.* It contains classification algorithms that solve a MIML problem by previouslty transforming it into a ML problem and applying any ML classification algorithm. The library considers three transformations widely used in the multi-instance learning environment: arithmetic, geometric or min-max transformation. Table 4.2 shows 15 Mulan ML algorithms that could be used after the transformation. Section 4.4.2 includes examples of the execution of each one of these algorithms, its configuration file and results obtained.

[1]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 4.1 for LP transformation.

Label transformation	**MI classifiers (Weka)**
BR [51]	CitationKNN [43] MDD [50] MIDD [50] MIBoost [45] MILR [37] MIOptimalBall [44] MIRI [52] MISMO [42] MISVM [53] MITI [54] MIWrapper[55] SimpleMI[56]
LP [51]	CitationKNN [43] MIWrapper [55] SimpleMI [56]

Table 4.1: MI Classifiers that can be used to solve the problem transformed to MI problem

Bag transformation	**ML classifiers (Mulan)**
Arithmetic Geometric Min-Max [56]	BR [23] LP [23] RPC [57] CLR [58] BRkNN [59] DMLkNN [60] IBLR [26] MLkNN [25] HOMER [61] RAkEL [27] PS [28] EPS [28] CC [24] ECC [24] MLStacking [51]

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

- *classifiers.miml.lazy.* It includes different kinds of lazy algorithms. MIML-kNN [2] is available. It is an algorithm that directly solves the problem working with the MIML data, without performing any previous transformation of the problem. This algorithm uses the nearest citers and references to a bag to estimate the possible classes to which it belongs. Besides, the library includes specific MIML proposals based on nearest neighbour that have been adapted from ML methods. Thus, MIMLBRkNN is a MIML algorithm adapted from BRkNN [62], DMIMLkNN is adapted from DMLkNN [60], MIMLIBLR is adapted from IBLR_ML [26] and MIMLMAPkNN is adapted from MLkNN [25].These algorithms maintain the ML algorithmic regarding labels but use MI distance between bags. In the section 4.4.3.2 an example of the execution of these algorithms is shown, explaining its configuration file and the results obtained.

- *classifiers.miml.neural.* It includes different kinds of neural networks algorithms. Currently MIMLNN [46], EnMIMLNNmetric [12], and MIMLRBF [63] algorithms are available. They are wrappers over the implementations developed in Matlab by LAMDA members [16] and Zhang and Wang [64]. Section 4.4.3.3 contains an example of the execution of these algorithms explaining its configuration file and the results obtained.

- *classifiers.miml.optimization.* It includes algorithms based on other optimization techniques. Currently MIMLFast [65], KiSar [66], MIMLSVM [47] and MIMLWel [67] are available. They are wrappers over the implementations developed in Matlab by LAMDA members [16]. Section 4.4.3.4 includes example of the execution of these algorithms, explaining its configuration file and the results obtained.

- *classifiers.miml.meta.* It contains MIMLBagging, an adaptation of the traditional bagging strategy of the machine learning [68] that 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 4.4.3.1 an example of the execution of this algorithm is shown, explaining its configuration file and the results obtained.

Approach	**MIML classifiers**
Bagging	MIMLBagging
ANN	MIMLNN [46] EnMIMLNNmetric [12] MIMLRBF [63]
Lazy	MIML-kNN [2] MIMLBRkNN MIMLMAPkNN DMIMLkNN MIMLIBLR
Other	MIMLFast [65] KiSar [66] MIMLSVM [47] MIMLWel [67]

Table 4.3: MIML Classifiers that solve the problem without transformation

4.2 Format of the files used by MIML library

The MIML library needs as input for a classification algorithm a MIML dataset and a configuration of the experiment. This section explains both formats.

4.2.1 Data set format

The format of the MIML data is based on Weka's format for MI learning and 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. It aims to identify those attributes in the *arff* file representing labels. Note that the class attributes do not need to be the last ones 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 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:Figure 4.1 shows a class diagram with the main classes of the library and their relationships.

```
<?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 a comma (,) and all lines must have the same number of attributes. The 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 ($\backslash n$).

    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\n41.9,-191,-142\n35,14.2,6.33",1,0,0,1
 bag2,"11.25,-98,10\n31,40.5,7.85",0,1,1,0
```

The library 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 saving in a file ML and MIML datasets.

The distribution of the library includes the *birds* dataset [69]. 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)[2]. This dataset has been formatted to MIML format specified in this section and Table 4.4 contains a summary of the main features of this dataset (they are described in section 2.4).

dataset	domain	bags	instances	avg. inst/bags	min inst/bag	max inst/bag	attr	labels	card	dens	dist
Birds	audio	257	2062	8.0.23	2	36	38	19	2.066	0.109	120

Table 4.4: Features of the *birds* dataset

4.2.2 Configuration file format

This section explains the format of configuration files and the following sections show the specific configuration files to execute each algorithm considered in the library. These files use an *xml* format, with the following structure:

Listing 4.1: General structure of an *xml* configuration file

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

4.2.2.1 Description of the element classifier

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*:

Listing 4.2: Structure of the element classifier

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

4.2.2.2 Description of the element evaluator

The next element that is a branch direct of root element is `<evaluator>`. This element describes the dataset used for the experiment and different validation methods that can be used that are specified in `<data>` element. Concerning validation methods, both holdout and cross-validation are included, both located in the package *miml.evaluation*. Besides, it is possible to design an own

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

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 the 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 4.2.1. An example configuration file for holdout would be one in Listing 4.3:

Listing 4.3: Example of holdout experiment with train and test files

```xml
<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>
```

It is also possible to specify a single *arff* file and the evaluator will perform data partitioning automatically. This evaluator can use one the following partitioning methods: *random*, *powerset* or *iterative* [70]. If they are not specified, the `<partitionMethod>` will be *random*, the default `<seed>` will be 1 and the default `<percentageTrain>` will be 80.

Listing 4.4: Example of holdout experiment with a single file for partitioning

```xml
<evaluator name="miml.evaluation.EvaluatorHoldout">
   <percentageTrain>65</percentageTrain>
   <partitionMethod>random</partitionMethod>
   <seed>7891</seed>
   <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
   </data>
</evaluator>
```

For the cross-validation evaluator, it is necessary to specify three elements: the number of folds, the path of the dataset in *arff* format, and the *xml* file path corresponding to the dataset. This evaluator runs and valuates an algorithm applying a cross-validation method with random partitioning. It is worth noting that it used *weka.core.Instances.trainCV* and *weka.core.Instances.testCV* so having examples of all labels in the partitioned data is not guaranteed.

Listing 4.5: Example of cross-validation experiment

```xml
<evaluator name="miml.evaluation.EvaluatorCV">
   <seed>127</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>`.

4.2.2.3 Description of the element report

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 4.5 shows the metrics included in the library.

Listing 4.6: Example of configuring a report

```
<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>
```

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 4.5: Evaluation measures included in MIML

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. If the method of validation used is holdout, it has no sense to set a true value the standard deviation parameter.

The library contains a set of configuration files for each algorithm included. These files, located in *configurations* folder, can be used as a template for creating customized configurations. Generally, all configuration files keep the structure specified in this section. In addition, the following sections give specific examples for each algorithm.

4.3 Managing MIML data

4.3.1 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 performing an analysis of imbalance and relationships among labels.

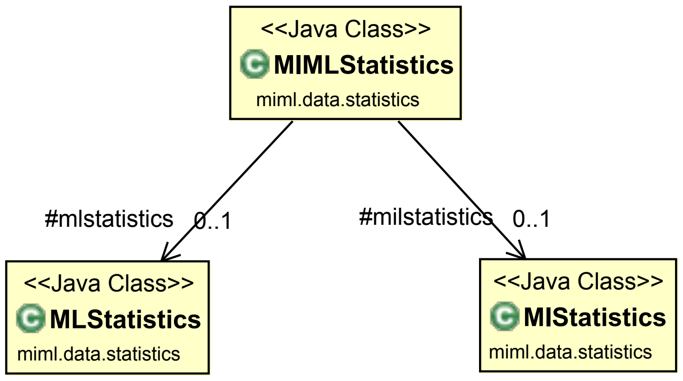


Figure 4.2: Statistics class diagram

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.

4.3.1.1 Information for multi-label data

The MLStatistics class belongs to the *miml.data.statistics* package, and will be responsible for obtaining information about 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 to 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 the proportion of unique label combinations value defined as the proportion of label sets that 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.

4.3.1.2 Information for multi-instance data

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

4.3.1.3 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.

4.3.2 Preprocessing MIML datasets

4.3.2.1 Data normalization

The MIML library includes min-max normalization in the *miml.data.normalization* package. Given an attribute value, x, the new x' normalized value will be calculated as $x' = \frac{x - \min(x)}{\max(x) - \min(x)}$.

4.3.2.2 Data partitioning

Besides the library provides the following methods for data partitioning in *miml.data.partitioning* package:

- *Random partitioning.* Patterns on each fold are randomly chosen, without considering label information except to guarantee the presence of patterns of all labels in the train set. Due to this fact, applied over datasets with a high number of labels (e.g. some subsets of miml protein data), this method may generate folds with uneven number of instances and with some duplicated instances. In these cases, using a lower number of folds (eg. 3 folds) or another kind of partitioning (eg. iterative or powerset) is recommended.
- *Iterative stratification* partitioning [70], which is specific for multi-label learning but considering MIML instances.
- *Label powerset* partitioning [70] which is stratified, specific for ML and based on label powerset transformation but applied to MIML instances.

4.3.3 Transforming MIML data sets

The library contains methods to transform a MIML data set to a MI data set in the Weka library format or to a ML data set in the Mulan library format. The Figure 4.3 shows the library classes that provide the functionality for transforming datasets.

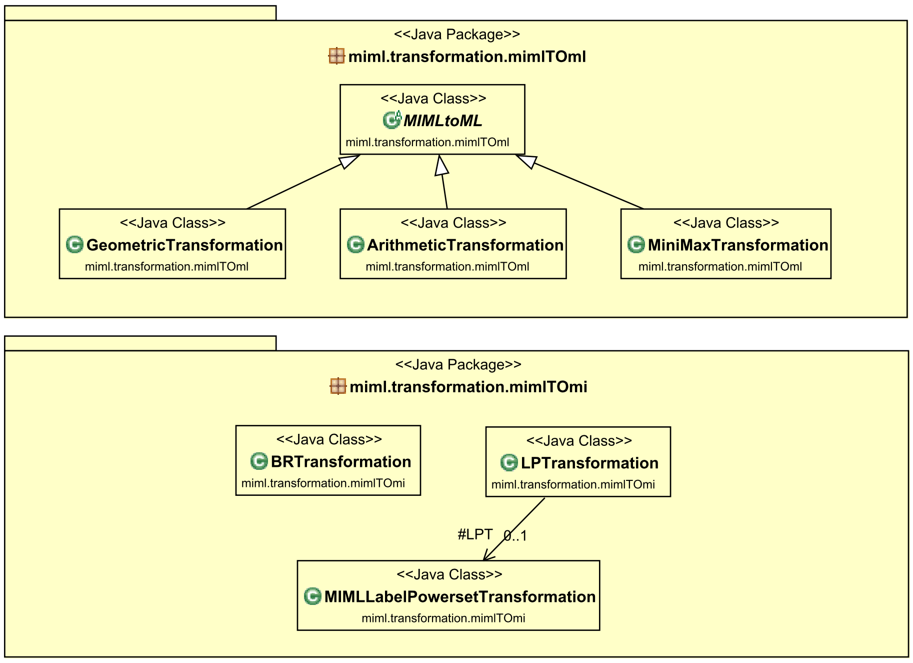

<<Java Package>>
miml.transformation.mimlTOml
<<Java Class>>
MIMLtoML
miml.transformation.mimlTOml
<<Java Class>>
GeometricTransformation
miml.transformation.mimlTOml
<<Java Class>>
ArithmeticTransformation
miml.transformation.mimlTOml
<<Java Class>>
MiniMaxTransformation
miml.transformation.mimlTOml
<<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

4.3.3.1 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. For instance, given the *toy* dataset with 4 binary labels in 4.2.1, the corresponding Binary Relevance transformation is formed by 4 datasets showed in Table 4.6.
- *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. For instance, given the *toy* dataset with 2 bags and 4 binary labels in 4.2.1, the corresponding Label Powerset transformation is formed by 1 dataset showed below.

```
@relation toy_LP_Transformation
@attribute id {bag1,bag2}
@attribute bag relational
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
@end bag
@attribute LP_Class_0 {0110,1001}
@data
bag1,'42,-198,-109\n41.9,-191,-142\n35,14.2,6.33',1001
bag2,'11.25,-98,10\n31,40.5,7.85',0110
```

```
@relation toy_label1

@attribute id {bag1,bag2}
@attribute bag relational
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
@end bag
@attribute label1 {0,1}

@data
bag1,'42,-198,-109\n41.9,-191,-142\n35,14.2,6.33',1
bag2,'11.25,-98,10\n31,40.5,7.85',0
```

```
@relation toy_label2

@attribute id {bag1,bag2}
@attribute bag relational
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
@end bag
@attribute label2 {0,1}

@data
bag1,'42,-198,-109\n41.9,-191,-142\n35,14.2,6.33',0
bag2,'11.25,-98,10\n31,40.5,7.85',1
```

```
@relation toy_label3

@attribute id {bag1,bag2}
@attribute bag relational
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
@end bag
@attribute label3 {0,1}

@data
bag1,'42,-198,-109\n41.9,-191,-142\n35,14.2,6.33',0
bag2,'11.25,-98,10\n31,40.5,7.85',1
```

```
@relation toy_label4

@attribute id {bag1,bag2}
@attribute bag relational
@attribute f1 numeric
@attribute f2 numeric
@attribute f3 numeric
@end bag
@attribute label4 {0,1}

@data
bag1,'42,-198,-109\n41.9,-191,-142\n35,14.2,6.33',1
bag2,'11.25,-98,10\n31,40.5,7.85',0
```

Table 4.6: Binary Relevance transformation of *toy* dataset

4.3.3.2 Methods to transform MIML data to ML data

The library includes three different methods described in [56] 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. In this way, the resulting dataset will be made up of as many instances as bags contained in the original dataset. For instance, given the *toy* dataset with 2 bags and 4 binary labels in 4.2.1, the corresponding arithmetic transformation is formed by 2 simple instances and showed below.

```
@relation toy_arithmetic_transformation

@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}

@data
bag1,39.633333,-124.933333,-81.556667,1,0,0,1
bag2,21.125,-28.75,8.925,0,1,1,0
```

- *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. Note that for a bag with two instances, AT and GT return the same transformed instance. For instance, given the *toy* dataset with 2 bags and 4 binary labels in 4.2.1, the corresponding geometric transformation is formed by 2 simple instances and showed below.

```
@relation toy_geometric_transformation

@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}

@data
bag1,38.5,-91.9,-67.835,1,0,0,1
bag2,21.125,-28.75,8.925,0,1,1,0
```

- *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. For instance, given the *toy* dataset with 2 bags and 4 binary labels in 4.2.1, the corresponding min-max transformation is formed by 2 simple instances and showed below.

```
@relation toy_min_max_transformation

@attribute id {bag1,bag2}
@attribute min_f1 numeric
```

```
@attribute min_f2 numeric
@attribute min_f3 numeric
@attribute max_f1 numeric
@attribute max_f2 numeric
@attribute max_f3 numeric
@attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}

@data
bag1,35,-198,-142,42,14.2,6.33,1,0,0,1
bag2,11.25,-98,7.85,31,40.5,10,0,1,1,0
```

Finally, the MIML library also includes a method to transform a MIML dataset into an ML one by conversion to its propositional format[3]. For instance, given the *toy* dataset with 2 bags and 4 binary labels in 4.2.1, the corresponding propositional transformation is formed by 5 simple instances and is shown below.

```
@relation toy_propositional_transformation

@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}

@data
bag1,42,-198,-109,1,0,0,1
bag1,41.9,-191,-142,1,0,0,1
bag1,35,14.2,6.33,1,0,0,1
bag2,11.25,-98,10,0,1,1,0
bag2,31,40.5,7.85,0,1,1,0
```

4.4 Running MIML classification algorithms included in the library

All algorithms included in the library are executed through 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. Section 3.1 describe how to execute an algorithm given a configuration file. The specific format of a configuration file is specified in section 4.2.2 and examples are shown in the following sections.

Concretely, **43** 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 13 specific algorithms for MIML learning. Moreover, many more combinations can be run considering all possible combinations between algorithms and transformation methods (to ML and to MI) available in the library. More information about algorithms included in the library can be consulted in Section 4.1.

[3]See a detailed description in https://waikato.github.io/weka-wiki/multi_instance_classification

4.4.1 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 4.1 shows the Weka MI algorithms that can be used for each transformation.

In general, algorithms that 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 customized transformation method and the library has the necessary interfaces to facilitate its implementation, the library contains transformation methods (they are detailed in the section 4.3.3.1 and 4.3.3.2). In addition, Table 4.1 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:

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

4.4.1.1 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 4.2.2. 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.

```
<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.

```
<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 miml-1.0.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 4.7.

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	4822
Test_time_ms	2444
Hamming Loss	0.1062753036437247
Subset Accuracy	0.21153846153846154
Macro-averaged Precision	0.46232104324983575
Macro-averaged F-Measure	0.4947605026552395

Table 4.7: Output generated by the CitationKNN report

4.4.1.2 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_MDD.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 miml-1.0.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 4.8).

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)	79415.8
Train_time_ms(std)	15799.96224552451
Test_time_ms(avg)	135.2
Test_time_ms(std)	84.31227668613866
Hamming Loss	0.09929348257521631
Hamming Loss Std	0.0062486261909041165
Subset Accuracy	0.10889894419306186
Subset Accuracy Std	0.045526673605654114
Macro-averaged Precision	0.3513021189336979
Macro-averaged Precision Std	0.07101997896762893
Macro-averaged Precision-BRCR	0.4
Macro-averaged Precision-BRCR Std	0.36800000000000005
Macro-averaged Precision-PAWR	0.6476190476190476
Macro-averaged Precision-PAWR Std	0.14093225623582767
Macro-averaged Precision-PSFL	0.4
Macro-averaged Precision-PSFL Std	0.20800000000000002
...	...

Table 4.8: Output generated by the MDD report

4.4.1.3 MIBoost classifier

This classifier [71] 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/MIML-toMI_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 miml-1.0.jar miml.run.RunAlgorithm -c configurations/toMI/MIMLtoMI_BR_MIBoost.
    config
```

The output generated, showed in Table 4.9, 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)	624.8
Test_time_ms(avg)	181.0
Hamming Loss	0.10956576962768913
Subset Accuracy	0.0
Macro-averaged Precision	0.10526315789473684
Macro-averaged Precision-BRCR	0.2
Macro-averaged Precision-PAWR	0.0
Macro-averaged Precision-PSFL	0.0
...	...

Table 4.9: Output generated by the MIBoost report

4.4.1.4 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 no sense that standard deviation is shown.

This configuration can be run with the command:

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

The output generated and showed in Table 4.10 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	4086
Test_time_ms	799
Hamming Loss	0.10222672064777326
Subset Accuracy	0.0
Macro-averaged Precision	0.10526315789473684
Macro-averaged F-Measure	0.10526315789473684

Table 4.10: Output generated by the MIDD report

4.4.1.5 **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*:

```
<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 no sense that standard deviation is shown.

It is possible run this configuration with the command:

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

The output generated and showed in Table 4.11 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	6430
Test_time_ms	192
Hamming Loss	0.13461538461538464
Subset Accuracy	0.07692307692307693
Macro-averaged Precision	0.35051147682726636
Macro-averaged F-Measure	0.3788037715036473

Table 4.11: Output generated by the MILR report

4.4.1.6 **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/MIML-toMI_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 no sense that standard deviation is shown.

It is possible run this configuration with the following command:

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

The output generated and showed in Table 4.12 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	2621
Test_time_ms	217
Hamming Loss	0.09615384615384613
Subset Accuracy	0.057692307692307696
Macro-averaged Precision	0.35945087176144697
Macro-averaged F-Measure	0.3063138308290663

Table 4.12: Output generated by the MIOptimalBall report

4.4.1.7 **MIRI classifier**

This classifier (Multi Instance Rule Inducer) [54] 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/MIML-toMI_BR_MIRI.config*:

```
<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 no sense that standard deviation is shown.

It is possible run this configuration with the command:

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

The output generated and showed in Table 4.13 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	15255
Test_time_ms	879
Hamming Loss	0.16093117408906882
Subset Accuracy	0.038461538461538464
Macro-averaged Precision	0.20913823019086178
Macro-averaged F-Measure	0.22709640143187557

Table 4.13: Output generated by the MIRI report

4.4.1.8 **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*:

```
<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 no sense that standard deviation is shown.

It is possible run this configuration with the following command:

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

The output generated and showed in Table 4.14 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	10791
Test_time_ms	564
Hamming Loss	0.12550607287449392
Subset Accuracy	0.07692307692307693
Macro-averaged Precision	0.3502810055441634
Macro-averaged F-Measure	0.33382835488098656

Table 4.14: Output generated by the MISMO report

4.4.1.9 **MISVM classifier**

This classifier [53] 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 no sense that standard deviation is shown.

It is possible run this configuration with the following command:

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

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

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	1875217
Test_time_ms	186
Hamming Loss	0.4898785425101216
Subset Accuracy	0.0
Macro-averaged Precision	0.19398817556712294
Macro-averaged F-Measure	0.2481984086139332

Table 4.15: Output generated by the MISVM report

4.4.1.10 MITI classifier

This classifier (Multi instance Tree Inducer) [52] is based a decision tree learned using Blockeel et al.'s algorithm [54]. 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 no sense that standard deviation is shown.

It is possible run this configuration with the following command:

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

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

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	4620
Test_time_ms	228
Hamming Loss	0.15384615384615383
Subset Accuracy	0.057692307692307696
Macro-averaged Precision	0.24422417379481096
Macro-averaged F-Measure	0.26450577838931444

Table 4.16: Output generated by the MITI report

4.4.1.11 **MIWrapper classifier**

MIWrapper It is a simple wrapper method for applying standard propositional learners to multi-instance data [55]. 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 miml-1.0.jar miml.run.RunAlgorithm -c configurations/toMI/
    MIMLtoMI_LP_MIWrapper.config
```

The generated output, located in *results/toMI/LP_MIWrapper*, is showed in Table 4.17.

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)	489
Test_time_ms(avg)	303
Hamming Loss	0.10222672064777326
Subset Accuracy	0.0
Macro-averaged Precision	0.10526315789473684
Macro-averaged F-Measure	0.10526315789473684

Table 4.17: Output generated by the MIWrapper report

4.4.1.12 **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/BR_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 no sense that standard deviation is shown.

It is possible run this configuration with the following command:

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

The generated output, located in *results/toMI/LP_SimpleMI*, is showed in Table 4.18.

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	187
Test_time_ms	330
Hamming Loss	0.10222672064777326
Subset Accuracy	0.0
Macro-averaged Precision	0.10526315789473684
Macro-averaged F-Measure	0.10526315789473684

Table 4.18: Output generated by the SimpleMI report

4.4.2 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 4.2.

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 4.3.3.1 and 4.3.3.2.

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>
```

4.4.2.1 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. It is also very important that the parameters are defined in strict order of appearance in the classifier constructor.

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.

If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

This BinaryRelevance accepts the following parameter:

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

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

```xml
<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"
               listOptions="-K 3 -I"/>
      </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 experiment configuration determines that the classifier specified for binary relevance is the algorithm IBk of Weka's classifiers. IBk algorithm is configured with 3 neighbors (-K=3 option) and weighting neighbors by the inverse of their distance (-I option).

Moreover, holdout is used as the 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 no sense that standard deviation is shown.

It is possible to run it with the following command:

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

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

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	135
Test_time_ms	328
Hamming Loss	0.09716599190283402
Subset Accuracy	0.19230769230769232
Macro-averaged Precision	0.49536340852130323
Macro-averaged F-Measure	0.3976698318312116

Table 4.19: Output generated by the BinaryRelevance report

4.4.2.2 **BRkNN Classifier**

BRkNN [59] 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 multi-label 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/MIML-toML_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">
    <percentageTrain>65</percentageTrain>
    <partitionMethod>random</partitionMethod>
    <seed>712637</seed>
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <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. The partition method will be random and both the the percentage of train and a seed are specified.

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 no sense to set a true the standard deviation.

It is possible to run it with the following command:

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

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

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	64
Test_time_ms	228
Hamming Loss	0.11812865497076015
Subset Accuracy	0.05555555555555555
Macro-averaged Precision	0.47574996563916233
Macro-averaged Precision-BRCR	0.8333333333333334
Macro-averaged Precision-PAWR	0.5714285714285714
Macro-averaged Precision-PSFL	0.3157894736842105
Macro-averaged Precision-RBNU	1.0
Macro-averaged Precision-DEJU	0.0
Macro-averaged Precision-OSFL	0.0
Macro-averaged Precision-HETH	0.8
Macro-averaged Precision-CBCH	0.3333333333333333
...	...

Table 4.20: Output generated by the BRkNN report

4.4.2.3 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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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/MIML-toML_AT_CC.config*:

```xml
<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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	577
Test_time_ms	171
Hamming Loss	0.12449392712550608
Subset Accuracy	0.15384615384615385
Macro-averaged Precision	0.33399122807017545
Macro-averaged F-Measure	0.3020437194273707

Table 4.21: Output generated by the ClassifierChain report

4.4.2.4 **DMLkNN Classifier**

This classifier implementing the Dependent Multi Label k Nearest Neighbours [60] 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 multi-label 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/MIML-toML_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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	116
Test_time_ms	188
Hamming Loss	0.08198380566801622
Subset Accuracy	0.1346153846153846
Macro-averaged Precision	0.47076023391812855
Macro-averaged F-Measure	0.35158200932194744

Table 4.22: Output generated by the DMLkNN report

4.4.2.5 **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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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/MIMLtoML_MMT_PS.config*:

```
<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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	431
Test_time_ms	322
Hamming Loss	0.10425101214574903
Subset Accuracy	0.15384615384615385
Macro-averaged Precision	0.22761904761904764
Macro-averaged F-Measure	0.22372589075935703

Table 4.23: Output generated by the PrunedSets report

4.4.2.6 EnsembleOfClassifierChains Classifier

It is an implementation of an 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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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">
    <percentageTrain>70</percentageTrain>
    <partitionMethod>powerset</partitionMethod>
    <seed>5</seed>
    <data>
```

```
      <trainFile>data/miml_birds.arff</trainFile>
      <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 with *powerset* stratification. The percentage of train will be 70 and the seed 5. 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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	388
Test_time_ms	933
Hamming Loss	0.10053981106612687
Subset Accuracy	0.20512820512820512
Macro-averaged Precision	0.5564745196324143
Macro-averaged F-Measure	0.5105007695626227

Table 4.24: Output generated by the EnsembleOfClassifierChains report

4.4.2.7 **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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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 a 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"
               listOptions="-K 5 -F"/>
      </parameters>
    </multiLabelClassifier>
    <transformationMethod name="miml.transformation.mimlTOml.ArithmeticTransformation"/>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <percentageTrain>75</percentageTrain>
    <partitionMethod>iterative</partitionMethod>
    <seed>25</seed>
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <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 the experiment determines that it is used holdout with *birds* dataset as validation method. The partitioning method is *iterative*, with 75 as percentage of train and 25 as seed. 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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	124
Test_time_ms	228
Hamming Loss	0.11460101867572156
Subset Accuracy	0.14516129032258066
Macro-averaged Precision	0.20906432748538012
Macro-averaged F-Measure	0.14680640996430472

Table 4.25: Output generated by the EnsembleOfPrunedSets report

4.4.2.8 **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 [61]. 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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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*:

```
<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
                " listOptions="-M 3 -B"/>
          </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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	471
Test_time_ms	186
Hamming Loss	0.1548582995951417
Subset Accuracy	0.07692307692307693
Macro-averaged Precision	0.3449372265161738
Macro-averaged F-Measure	0.3416423908234097

Table 4.26: Output generated by the HOMER report

4.4.2.9 **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 multi-label 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 miml-1.0.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 4.27.

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)	496.6
Test_time_ms(avg)	83.2
Hamming Loss	0.16364213701675
Subset Accuracy	0.0234539969834087487
Macro-averaged Precision	0.2655703968238594
Macro-averaged F-Measure	0.2814683457185845

Table 4.27: Output generated by the IBLR_ML report

4.4.2.10 **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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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.trees.J48"/>
     </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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	241
Test_time_ms	210
Hamming Loss	0.1538461538461538
Subset Accuracy	0.11538461538461539
Macro-averaged Precision	0.20664160401002504
Macro-averaged F-Measure	0.2067232859476627

Table 4.28: Output generated by the LabelPowerset report

4.4.2.11 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 multi-label 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*:

```
<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 miml-1.0.jar miml.run.RunAlgorithm -c configurations/toML/MIMLtoML_GT_MLkNN.
    config
```

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

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)	52.4
Test_time_ms(avg)	65.6
Hamming Loss	0.09417321584504248
Subset Accuracy	0.15573152337858218
Macro-averaged Precision	0.3862819928609402
Macro-averaged F-Measure	0.27417292167017526

Table 4.29: Output generated by the MLkNN report

4.4.2.12 MultiLabelStacking Classifier

Implementation of the *2BR* or Multi-Label stacking method [51]. 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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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/MIML-toML_GT_MLStacking.config*:

```xml
<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"
            listOptions="-B -C 0.5"/>
        <parameter class="weka.classifiers.Classifier" value="weka.classifiers.trees.LMT"
            listOptions="-B -I 5"/>
      </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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	3488
Test_time_ms	181
Hamming Loss	0.09514170040485827
Subset Accuracy	0.1346153846153846
Macro-averaged Precision	0.3451754385964912
Macro-averaged F-Measure	0.3169019996227127

Table 4.30: Output generated by the MultiLabelStacking report

4.4.2.13 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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

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

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	1850
Test_time_ms	328
Hamming Loss	0.1174089068825911
Subset Accuracy	0.09615384615384616
Macro-averaged Precision	0.378587494376968
Macro-averaged F-Measure	0.3516529023454231

Table 4.31: Output generated by the RAkEL report

4.4.2.14 **Pairwise Classifier**

Implementation of the Ranking by Pairwise Comparisons (RPC) [57] 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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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/MIML-toML_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>
  </report>
</configuration>
```

The configuration of experiment determines that it is used cross validation with 5 folds as validation method and all measures will be shown in the output file.

It is possible to run it with the following command:

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

The generated output, located in *results/toML/MMT_RPC*, is showed in Table 4.32.

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)	258.6
Test_time_ms(avg)	26101.4
Average Precision	0.6402053375289556
Coverage	5.356938159879336
OneError	0.34969834087481144
Ranking Loss	0.15999635065961254

Table 4.32: Output generated by the Pairwise report

4.4.2.15 **CalibratedLabelRanking classifier**

Implementation of the Calibrated Label Ranking (CLR) [58] 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 multi-label classifier constructor. If the parameter is a *weka.classifiers.Classifier*, it is also possible to specify a specific configuration for the single-label classifier by means of an attribute `listOptions`. This attribute will be a string with a valid available option according to its specifications in the Weka library. If `listOptions` is not set, the default configuration will be used.

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/MIML-toML_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 no sense to set a true value the standard deviation parameter.

It is possible to run it with the following command:

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

The generated output, located in *results/toML/MMT_CLR*, is showed in Table 4.33.

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	240
Test_time_ms	548
Hamming Loss	0.09514170040485834
Subset Accuracy	0.23076923076923078
Macro-averaged Precision	0.4971177944862155
Macro-averaged F-Measure	0.44446377026233747

Table 4.33: Output generated by the CalibratedLabelRanking report

4.4.3 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 different implementations of this type of algorithms: Multi-Instance MultiLabel k-nearest neighbour [72], an ensemble algorithm using bagging [68], wrappers for ML neighbour based methods and wrappers for neural network based methods.

4.4.3.1 Ensemble methods

4.4.3.1.1 MIMLBagging Algorithm

This algorithm is an adaptation of the traditional bagging strategy of the machine learning [68]. 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/MIMLBagging*:

```
<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>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 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 previosly 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 four 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 miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/
    MIMLBagging.config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLBagging.csv*, is showed in Table 4.34.

Algorithm	MIMLBagging
Dataset	miml_birds.arff
Configuration File	MIMLClassifierEnsemble.config
Train_time_ms	2273
Test_time_ms	2372
Hamming Loss	0.08198380566801619
Subset Accuracy	0.21153846153846154
Macro-averaged Precision	0.49808612440191385
Macro-averaged F-Measure	0.40434419381787806

Table 4.34: Output generated by the MIMLBagging report

4.4.3.2 **Neighbour based methods**

4.4.3.2.1 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.

```xml
<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 no 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 miml-1.0.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 4.35.

Algorithm	MIMLkNN
Dataset	miml_birds_random_80train.arff
Configuration File	MIMLkNN.config
Train_time_ms	539
Test_time_ms	528
Hamming Loss	0.078947368421052
Subset Accuracy	0.19230769230769232
Macro-averaged Precision	0.5798245614035087
Macro-averaged F-Measure	0.38389505231610493

Table 4.35: Output generated by the MIMLkNN report

4.4.3.2.2 MIMLBRkNN Algorithm

MIMLBRkNN is the adaptation to the MIML framework of the BRkNN [62] ML algorithm. MIMLBRkNN maintains the treatment of labels of BRkNN but uses a MI measure of distance between bags.

This classifier accepts the following parameters:

- *numOfNeighbours*: number of neighbours.
- *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 MIMLBRkNN algorithm is shown.

```
<configuration>
  <classifier name="miml.classifiers.miml.lazy.MIMLBRkNN">
    <numOfNeighbours>5</numOfNeighbours>
    <extension>NONE</extension>
    <metric name="miml.core.distance.MinimalHausdorff">
    </metric>
  </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/BRkNN_wrapper.csv</fileName>
  </report>

</configuration>
```

In this case, it is necessary to use the `<numOFNeighbours>` 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 cross-validation with 5 folds as validation method.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLBRkNN.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLBRkNN.csv*, is shown in Table 4.36.

Algorithm	MIMLBRkNN
Dataset	miml_birds.arff
Configuration File	MIMLBRkNN.config
Train_time_ms(avg)	0.8
Test_time_ms(avg)	376.0
Hamming Loss	0.08950940700166707
Subset Accuracy	0.14773755656108598
Example-Based Precision	0.5849296128707895
Example-Based Recall	0.4403770739064856
...	...

Table 4.36: Output generated by the MIMLBRkNN report

4.4.3.2.3 MIMLMAPkNN Algorithm

MIMLMAPkNN is the adaptation to the MIML framework of the MLkNN [25] ML algorithm. MIMLMAPkNN maintains the treatment of labels of MLkNN but uses a MI measure of distance between bags.

This classifier accepts the following parameters:

- *numOfNeighbours*: number of neighbours.
- *smooth*: the smooth factor.
- *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 MIMLMAPkNN algorithm is shown.

```
<configuration>
  <classifier name="miml.classifiers.miml.lazy.MIMLMAPkNN">
    <numOfNeighbours>5</numOfNeighbours>
    <smooth>1.0</smooth>

    <metric name="miml.core.distance.AverageHausdorff">
    </metric>
  </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/MIMLMAPkNN.csv</fileName>
  </report>

</configuration>
```

In this case, it is necessary to use the `<numOFNeighbours>`, `<smooth>` 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 cross-validation with 5 folds as validation method.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLMAPkNN
    .config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLMAPkNN.csv*, is shown in Table 4.37.

Algorithm	MIMLMAPkNN
Dataset	miml_birds.arff
Configuration File	MIMLMAPkNN.config
Train_time_ms(avg)	1044.6
Test_time_ms(avg)	287.4
Hamming Loss	0.08129713423831071
Subset Accuracy	0.21787330316742085
Example-Based Precision	0.58497988939165415
Example-Based Recall	0.42320764203117145
...	...

Table 4.37: Output generated by the MIMLMAPkNN report

4.4.3.2.4 DMIMLkNN Algorithm

DMIMLkNN is the adaptation to the MIML framework of the DMLkNN [60] (Dependent Multi-Label k Nearest Neighbours) ML algorithm. DMIMLkNN maintains the treatment of labels of DMLkNN but uses a MI measure of distance between bags.

This classifier accepts the following parameters:

- *numOfNeighbours*: number of neighbours.
- *smooth*: the smooth factor.
- *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 DMIMLkNN algorithm is shown.

```
<configuration>
  <classifier name="miml.classifiers.miml.lazy.DMIMLkNN">

    <numOfNeighbours>5</numOfNeighbours>
    <smooth>1.0</smooth>
    <metric name="miml.core.distance.MinimalHausdorff">
    </metric>
  </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/DMLkNN_wrapper.csv</fileName>
  </report>

</configuration>
```

In this case, it is necessary to use the `<numOFNeighbours>`, `<smooth>` 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 cross-validation with 5 folds as validation method.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/DMIMLkNN.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/DMIMLkNN.csv*, is shown in Table 4.38.

Algorithm	DMIMLkNN
Dataset	miml_birds.arff
Configuration File	DMIMLkNN.config
Train_time_ms (avg)	966.2
Test_time_ms (avg)	364.6
Hamming Loss	0.09279193458760024
Subset Accuracy	0.10837104072398189
Example-Based Precision	0.4581322272498743
Example-Based Recall	0.2705869783810961
...	...

Table 4.38: Output generated by the DMIMLkNN report

4.4.3.2.5 MIMLIBLR Algorithm

MIMLIBLR is the adaptation to the MIML framework of the IBLR_ML [26] ML algorithm. MIMLIBLR maintains the treatment of labels of IBLR_ML but uses a MI measure of distance between bags.

This classifier accepts the following parameters:

- *numOfNeighbours*: number of neighbours.
- *addFeatures*: if it is false IBLR_ML is used. If it is true, IBLR_ML+ is used.
- *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 MIMLIBLR algorithm is shown.

```
<configuration>
<classifier name="miml.classifiers.miml.lazy.MIMLIBLR">
   <numOfNeighbours>5</numOfNeighbours>
   <addFeatures>true</addFeatures>
   <metric name="miml.core.distance.MinimalHausdorff">
   </metric>
 </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/MIMLIBLR.csv</fileName>
 </report>

</configuration>
```

In this case, it is necessary to use the `<numOFNeighbours>`, `<addFeatures>` 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 cross-validation with 5 folds as validation method.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLIBLR.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLIBLR.csv*, is shown in Table 4.39.

Algorithm	MIMLIBLR
Dataset	miml_birds.arff
Configuration File	MIMLIBLR.config
Train_time_ms (avg)	1191.6
Test_time_ms (av)	364.4
Hamming Loss	0.1113995395729142
Subset Accuracy	0.09720965309200605
Example-Based Precision	0.4709351432880845
Example-Based Recall	0.41476370035193566
...	...

Table 4.39: Output generated by the MIMLIBLR report

4.4.3.3 Neural Network based methods

In this section, it is shown a set of algorithms based on neural networks for MIML. Concretely, it has been developed wrappers for MIMLNN [46] and EnMIMLNNmetric [12] algorithms available at web of LAMDA members [16] and MIMLRBF [25] available at Zhangs's personal website [64].

To run these algorithms, it is necessary to install the MATLAB Runtime that it is available and free for all operative systems `https://es.mathworks.com/products/compiler/matlab-runtime.html`. It should be selected *R2021a (9.10)* option.

4.4.3.3.1 MIMLNN Algorithm

This is a wrapper for including the MIMLNN [46] algorithm available at lamda group web `http://www.lamda.nju.edu.cn/code_MIML.ashx`.

This classifier accepts the following parameters:

- *ratio*: the number of clusters is set to ratio x numberOfTrainingBags.
- *lamda*: the regularization parameter used to compute matrix inverse.
- *seed*: the seed for kmedoids clustering.

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

```xml
<configuration>
<classifier name="miml.classifiers.miml.neural.MIMLNN">
    <ratio>1.0</ratio>
    <lambda>0.4</lambda>
        <seed>1</seed>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
      <percentageTrain>80</percentageTrain>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/MIMLNN.csv</fileName>
  </report>
</configuration>
```

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

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout with 80% for training data.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLNN.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLNN.csv*, is shown in Table 4.40.

Algorithm	MIMLNN
Dataset	miml_birds.arff
Configuration File	MIMLNN.config
Train_time_ms	929
Test_time_ms	885
Hamming Loss	0.23942208462332298
Subset Accuracy	0.0392156862745098
Example-Based Precision	0.19946524064171123
Example-Based Recall	0.25882352941176473
...	...

Table 4.40: Output generated by the MIMLNN report

4.4.3.3.2 MIMLRBF Algorithm

This is a wrapper for including the MIMLRBF [63] algorithm available at Zhang's web page http://palm.seu.edu.cn/zhangml/.

This classifier accepts the following parameters:

- *ratio*: the number of clusters is set to ratio x numberOfTrainingBags.
- *mu*: the regularization parameter used to compute matrix inverse.
- *seed*: the seed for kmedoids clustering.

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

```xml
<configuration>
<classifier name="miml.classifiers.miml.neural.MIMLRBF">
    <ratio>0.1</ratio>
    <mu>0.6</mu>
        <seed>1</seed>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
      <percentageTrain>80</percentageTrain>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/MIMLRBF.csv</fileName>
  </report>
</configuration>
```

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

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout with 80% for training data.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLRBF.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLRBF.csv*, is shown in Table 4.41.

Algorithm	MIMLRBF
Dataset	miml_birds.arff
Configuration File	MIMLRBF.config
Train_time_ms	1586
Test_time_ms	870
Hamming Loss	0.12590299277605785
Subset Accuracy	0.0
Example-Based Precision	0.08169934640522876
Example-Based Recall	0.06699346405228758
...	...

Table 4.41: Output generated by the MIMLRBF report

4.4.3.3.3 EnMIMLNNmetric Algorithm

This is a wrapper for including the EnMIMLNNmetric [12] algorithm available at lamda group web `http://www.lamda.nju.edu.cn/code_EnMIMLNNmetric.ashx`.

This classifier accepts the following parameters:

- *ratio*: the fraction parameter of EnMIMLNNmetric.
- *mu*: the scaling factor of EnMIMLNNmetric.
- *seed*: the seed for kmedoids clustering.

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

```
<configuration>
<classifier name="miml.classifiers.miml.neural.EnMIMLNNmetric">
    <ratio>1.0</ratio>
    <mu>0.8</mu>
        <seed>1</seed>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
      <percentageTrain>80</percentageTrain>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/EnMIMLNNmetric.csv</fileName>
  </report>
</configuration>
```

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

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout with 80% for training data.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/
    EnMIMLNNmetric.config
```

The output generated by this configuration, located in *results/MIMLClassifier/EnMIMLNN-metric.csv*, is shown in Table 4.42.

Algorithm	EnMIMLNNmetric
Dataset	miml_birds.arff
Configuration File	EnMIMLNNmetric.config
Train_time_ms	1931
Test_time_ms	1022
Hamming Loss	0.21981424148606815
Subset Accuracy	0.0392156862745098
Example-Based Precision	0.20410830999066293
Example-Based Recall	0.23921568627450981
...	...

Table 4.42: Output generated by the EnMIMLNNmetric report

4.4.3.4 Other optimization based methods

In this section, it is shown a set of algorithms based on optimization for MIML. Concretely, it have been developed wrappers for MIMLFast [65], KiSar [66], MIMLSVM [47] and MIMLWel [67] algorithms.

To run these algorithms, it is necessary to install the MATLAB Runtime that it is available and free for all operative systems `https://es.mathworks.com/products/compiler/matlab-runtime.html`. It should be selected *R2021a (9.10)* option. Currently, due to the specific libraries that use, KiSar and MIMLSVM can only be run with Windows 64 bits. Concretely KiSar needs the Matlab version of Liblinear and MIMLSVM needs Libsvm and the packaging has been carried out with Windows 64 bits.

4.4.3.4.1 MIMLFast Algorithm

This is a wrapper for including the MIMLFast [65] algorithm available at lamda group web site `http://www.lamda.nju.edu.cn/code_MIMLfast.ashx`.

This classifier accepts the following parameters:

- *normUp*: the norm of each vector.
- *maxiter*: the number of iterations.
- *stepSize*: the step size of SGD (stochastic gradient descent).
- *numSub*: the number of sub concepts.

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

```
<configuration>
  <classifier name="miml.classifiers.miml.optimization.MIMLFast">
    <normUp>10</normUp>
    <maxiter>10</maxiter>
        <stepSize>0.005</stepSize>
        <numSub>5</numSub>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
      <percentageTrain>80</percentageTrain>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/MIMLFast.csv</fileName>
  </report>
</configuration>
```

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

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout with 80% for training data experiment.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLFast.
   config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLFast.csv*, is shown in Table 4.43.

Algorithm	MIMLFast
Dataset	miml_birds.arff
Configuration File	MIMLFast.config
Train_time_ms	892
Test_time_ms	717
Hamming Loss	0.5376676986584106
Subset Accuracy	0.0
Example-Based Precision	0.08153948448066095
Example-Based Recall	0.32222222222222224
...	...

Table 4.43: Output generated by the MIMLFast report

4.4.3.4.2 MIMLWel Algorithm

This is a wrapper for including the MIMLWel [67] algorithm available at lamda group web site `http://www.lamda.nju.edu.cn/code_MIMLWEL.ashx`.

This classifier accepts the following parameters:

- C: Controls the empirical loss on labeled data.

- *m*: Controls the difference between the learned training targets and the original input training targets.
- *beta*: Controls the similarity between training bags and their prototypes.
- *iteration*: Iteration number.
- *epsilon*: Value for epsilon.
- *ratio*: The number of centroids of the i-th class is set to be $ratio * T_i$, where T_i is the number of train bags with label i.
- *mu*: The ratio used to determine the standard deviation of the Gaussian activation function.

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

```
<configuration>
  <classifier name="miml.classifiers.miml.optimization.MIMLWel">
    <C>50</C>
    <m>1</m>
    <beta>2</beta>
    <iteration>20</iteration>
    <epsilon>1e-3</epsilon>
    <ratio>0.1</ratio>
    <mu>1.0</mu>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
      <percentageTrain>80</percentageTrain>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/MIMLWel.csv</fileName>
  </report>
</configuration>
```

In this case, it is necessary to use the `<C>`, `<m>`, `<beta>`, `<iteration>`, `<epsilon>`, `<ratio>`, and `<mu>` elements to configure the different parameters of algorithm (they have been commented previously).

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout with 80% for training data experiment.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLWel.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLWel.csv*, is shown in Table 4.44.

Algorithm	MIMLWel
Dataset	miml_birds.arff
Configuration File	MIMLWel.config
Train_time_ms	34771
Test_time_ms	875
Hamming Loss	0.22187822497420007
Subset Accuracy	0.0392156862745098
Example-Based Precision	0.3014113337642749
Example-Based Recall	0.34934640522875826
...	...

Table 4.44: Output generated by the MIMLWel report

4.4.3.4.3 KiSar Algorithm

This is a wrapper for including the KiSar [66] algorithm available at lamda group web site http://www.lamda.nju.edu.cn/code_KISAR.ashx.

This classifier accepts the following parameters:

- C: Parameter set for liblinear.
- *iteration*: Maximum number of optimization iterations.
- *epsilon*: The epsilon parameter for the algorithm.
- K: Maximum number of prototypes for k_means clustering.
- *relationMethod*: Method used to build relation matrix.
  - 1. No cooccurrences.
  - 2. All labels are related.
  - 3. Labels *i,j* cooccur if their cooccurrence values are greater than the mean of all values in the cooccurrence matrix (including main diagonal).
  - 4. Labels *i,j* coocur if their coocurrence values are greater than the mean of the coocurrence values of all labels (excluding main diagonal).
  - 5. Labels *i,j* coocur if $prob(i,j) >= min(prob(i), prob(j)) * 0.1$ (10 percent).

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

```
<configuration>
  <classifier name="miml.classifiers.miml.optimization.KiSar">
            <C>500</C>
            <iteration>20</iteration>
            <epsilon>1e-3</epsilon>
            <K>1000</K>
            <relationMethod>4</relationMethod>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
      <percentageTrain>80</percentageTrain>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/KiSar.csv</fileName>
  </report>
</configuration>
```

In this case, it is necessary to use the `<C>`, `<iteration>`, `<epsilon>`, `<K>`, and `<relationMethod>` elements to configure the different parameters of algorithm (they have been commented previously).

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout with 80% for training data experiment.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/KiSar.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/KiSar.csv*, is shown in Table 4.45.

Algorithm	KiSar
Dataset	miml_birds.arff
Configuration File	KiSar.config
Train_time_ms	5388
Test_time_ms	964
Hamming Loss	0.1382868937048504
Subset Accuracy	0.0392156862745098
Example-Based Precision	0.22549019607843138
Example-Based Recall	0.1281045751633987
...	...

Table 4.45: Output generated by the KiSar report

4.4.3.4.4 MIMLSVM Algorithm

This is a wrapper for including the MIMLSVM [47] algorithm available at lamda group web site `http://www.lamda.nju.edu.cn/code_MIMLBoost%20and%20MIMLSVM.ashx`.

This classifier accepts the following parameters:

- *type*: The type of svm used in training, which can take the value of "RBF", "Poly" or "Linear".
- *para*: A string that gives the corresponding parameters used for the svm:
  - If type is "RBF", para gives the value of gamma (i.e. para="1") where the kernel is $exp(-Gamma * |x(i) - x(j)|^2)$.
  - If type is "Poly", then para gives the value of gamma, coefficient, and degree respectively, where the kernel is $(gamma* < x(i), x(j) > +coefficient)^{degree}$. Values in the string are delimited by blank spaces (i.e. para="1, 0, 1").
  - If type is "Linear", then para is an empty string, where the kernel is <x(i),x(j)> (i.e. para ="").
- *cost*: The cost parameter used for the base svm classifier.
- *h*: Whether to use the shrinking heuristics, 0 or 1 (default 1).
- *ratio*: Parameter k is set to be 20% of the number of training bags.
- *seed*: Seed for kmedoids clustering.

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

```xml
<configuration>
  <classifier name="miml.classifiers.miml.optimization.MIMLSVM">
      <type>RBF</type>
      <para>0.2</para>
      <cost>1</cost>
      <h>1</h>
      <ratio>0.2</ratio>
      <seed>1</seed>
  </classifier>

  <evaluator name="miml.evaluation.EvaluatorHoldout">
    <data>
      <trainFile>data/miml_birds.arff</trainFile>
      <xmlFile>data/miml_birds.xml</xmlFile>
      <percentageTrain>80</percentageTrain>
    </data>
  </evaluator>

  <report name="miml.report.BaseMIMLReport">
    <fileName>results/MIMLClassifier/MIMLSVM.csv</fileName>
  </report>
</configuration>
```

In this case, it is necessary to use the `<type>`, `<para>`, `<cost>`, `<h>`, `<ratio>`, and `<seed>` elements to configure the different parameters of algorithm (they have been commented previously).

The rest of configuration elements are defined similarly to the rest of algorithms. It has been defined a holdout with 80% for training data experiment.

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

```
$ java -cp miml-1.0.jar miml.run.RunAlgorithm -c configurations/MIMLClassifier/MIMLSVM.
    config
```

The output generated by this configuration, located in *results/MIMLClassifier/MIMLFast.csv*, is shown in Table 4.46.

Algorithm	MIMLSVM
Dataset	miml_birds.arff
Configuration File	MIMLSVM.config
Train_time_ms	1460
Test_time_ms	1062
Hamming Loss	0.16615067079463366
Subset Accuracy	0.0392156862745098
Example-Based Precision	0.23529411764705882
Example-Based Recall	0.18039215686274512
...	...

Table 4.46: Output generated by the MIMLSVM report

4.5 Developing a new MIML classification 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.

4.5.1 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.

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

public class MIMLkNN {
}
```

4.5.2 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()*.

```
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() {
```

```java
	}

	@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 3.1 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.

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] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] Lin Dong. A comparison of multi-instance learning algorithms. Master's thesis, The University of Waikato, 2006.

[57] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172:1897–1916, 2008.

[58] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza mencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133 – 153, 2008.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] M.L. Zhang and Z.J. Wang. Mimlrbf: Rbf neural networks for multi-instance multi-label learning. *Neurocomputing*, 72(16-18):3951–3956, 2009.

[64] Min-Ling Zhang. Personal web of min-ling zhang. http://palm.seu.edu.cn/zhangml/, 2021. Accessed: 2020-11-17.

[65] Sheng-Jun Huang, Wei Gao, and Zhi-Hua Zhou. Fast multi-instance multi-label learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(11):2614–2627, 2019.

[66] Yu-Feng Li, Ju-Hua Hu, Yuang Jiang, and Zhi-Hua Zhou. Towards discovering what patterns trigger what labels. In *26th AAAI Conference on Artificial Intelligence*, Toronto, Canada, 2012.

[67] Shu-Jun Yang, Yuan Jiang, and Zhi-Hua Zhou. Multi-instance multi-label learning with weak label. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence (IJCAI'13)*, pages 1862–1868, Beijing, China, 2013.

[68] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[69] 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.

[70] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. On the stratification of multi-label data. In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 145–158, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[71] 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.

[72] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.