Una librería para el aprendizaje Multi-instancia Multi-etiqueta

Álvaro Belmonte, Amelia Zafra, Eva Gibaja, Sebastián Ventura
Departamento de Informática y Análisis Numérico
Universidad de Córdoba
Córdoba, España

***Resumen*—Este artículo presenta una librería para trabajar en la resolución de problemas de clasificación con múltiples instancias y múltiples etiquetas. Se describe el formato de datos, la arquitectura software, así como las diferentes propuestas algorítmicas que incorpora. La librería permite añadir nuevos algoritmos de forma sencilla, facilitando a los investigadores en esta área el desarrollo, prueba y comparación de nuevas propuestas. Además, es libre y de código abierto y está implementada en Java, usando las librerías Weka y Mulan. De este modo, los usuarios que trabajan tanto en el aprendizaje con múltiples instancias como en el aprendizaje con múltiples etiquetas, se encontrarán con un entorno de desarrollo con el que están familiarizados.**

***Index Terms*—multi-instance learning, multi-label learning, software**

I. Introducción

A medida que los problemas de clasificación se vuelven más complejos, encontrar una representación adecuada de la información se convierte en una tarea cada vez más complicada. La experiencia demuestra que una representación precisa, que sea capaz de reflejar todas las relaciones e interacciones existentes en los datos, influye directamente en una resolución más efectiva del problema. En este contexto, el aprendizaje con múltiples instancias y múltiples etiquetas (*multi-instance multi-label learning*, MIML) se presenta como una alternativa prometedora que permite realizar una formulación natural que se adapta a las condiciones particulares de la representación de objetos complejos que suelen tener los problemas reales. Entre los problemas donde ha sido aplicado con éxito destacan la categorización de textos o imágenes [1]–[3], la detección de vídeo y audio [4] o la bioinformática [5], [6].

El aprendizaje MIML introduce una mayor flexibilidad en la representación de objetos, tanto en el espacio de entrada como en el de salida, debido a los dos paradigmas de aprendizaje que combina. Por un lado, la representación basada en el aprendizaje multi-instancia (*multi-instance*, MI) [7] ofrece una alternativa a la representación de instancias simples. En el aprendizaje MI un objeto o patrón, conocido habitualmente como *bolsa*, es representado mediante un conjunto variable de instancias, todas ellas con el mismo número de atributos. De este modo, un objeto puede ser representado con descripciones alternativas [7], con diferentes componentes [2], o bien mostrando su evolución en el tiempo [8]. Por otro lado, el aprendizaje multi-etiqueta (*multi-label*, ML) [9] introduce una mayor flexibilidad, esta vez en el espacio de salida, permitiendo que cada patrón tenga la posibilidad de pertenecer de forma simultánea a varias clases (etiquetas). Por ejemplo, para la resolución del problema de clasificación de imágenes, una imagen podría ser representada mediante múltiples instancias donde cada instancia se correspondería con diferentes regiones de la imagen, pudiendo a su vez tener diferentes etiquetas, tales como nubes, leones, y paisajes.

Actualmente, existen herramientas que permiten resolver problemas utilizando el aprendizaje MI y ML. Así, Weka [10] permite trabajar con problemas MI, y las librerías MEKA [11] y MULAN [12] permiten abordar el aprendizaje ML. No obstante, ninguna de ellas permite trabajar con el aprendizaje MIML. Hasta donde los autores conocemos, los únicos algoritmos públicamente disponibles para resolver problemas MIML han sido desarrollados por el grupo de investigación LAMDA [13]. Las principales limitaciones de estas implementaciones son que: están codificados en Matlab, con lo que es necesario disponer de licencia de este software para poder ejecutarlos y están formados por paquetes independientes, que habitualmente contienen la implementación de un único algoritmo que utiliza un formato de entrada y salida distinto al de otros paquetes, lo que dificulta la tarea de poder realizar un estudio experimental con ellos.

En este contexto, este artículo presenta una librería para aprendizaje MIML basada en las librerías Weka y MULAN, con lo que los investigadores en MI y ML, que empleen las librerías anteriores, estarán familiarizados con su estructura y formato. Entre sus características más relevantes se puede destacar: que utiliza un formato de datos diseñado específicamente para este aprendizaje, que cuenta con un conjunto de algoritmos desarrollados que directamente pueden ser ejecutados, que permite el uso de los algoritmos implementados para aprendizaje MI y ML de Weka y Mulan en un contexto MIML, que facilita el diseño y desarrollo de nuevos modelos que resuelvan problemas de clasificación con una representación MIML, que permite llevar a cabo un estudio experimental utilizando validación cruzada, y finalmente, su uso y ejecución resultan sencillos mediante la configuración de ficheros xml.

El resto del artículo se organiza como sigue. En la sección 2 se describe diferentes algoritmos MIML propuestos en la literatura. La sección 3, muestra la descripción de la librería, considerando el formato de los datos, su funcionalidad, su arquitectura y los principales paquetes y elementos de los que consta. En la sección 4, se describe algunos ejemplos de uso. Finalmente, la sección 5 describe las conclusiones más

relevantes del trabajo realizado.

II. Trabajo Previo

Los algoritmos de clasificación desarrollados para aprendizaje MIML se pueden clasificar en dos enfoques [14]. Por un lado, algoritmos basados en una estrategia de transformación del problema MIML, y por otro lado, algoritmos que abordan el problema MIML directamente.

Debido a que tanto el aprendizaje MI como el aprendizaje ML son parte de MIML, se pueden usar como vía o puente para la transformación y posterior resolución de problemas MIML. Una primera aproximación consiste en aplicar alguna técnica de descomposición de etiquetas que transforme el problema MIML a un problema MI. Después de esto, puede ser utilizado cualquier algoritmo MI para resolver el problema. Otra aproximación consiste en emplear alguna técnica que permita unificar las diferentes instancias de una bolsa en una única instancia, transformando el problema MIML en un problema ML al que se le puede aplicar cualquier algoritmo de ML. En la literatura se pueden encontrar diferentes algoritmos que realizan una transformación del problema, entre ellos encontramos métodos basados en *ensembles* [15], [4], máquinas de vector soporte [5] y métodos basados en redes neuronales [1].

El rendimiento de los algoritmos anteriores puede verse limitado debido a la pérdida de información incurrida durante el proceso de simplificación/transformación. Idealmente, las conexiones entre instancias y etiquetas, así como la correlación entre las propias etiquetas deberían ser tenidas en cuenta. Por este motivo, también se han propuesto técnicas para abordar el problema MIML de forma directa basadas en redes neuronales [16], *ensembles* [6], máquinas de vector soporte [3] o vecinos más cercanos [2]. En [17] puede encontrarse una revisión más exhaustiva de algoritmos publicados en el aprendizaje MIML.

III. Descripción de la librería

En esta sección se comentará el formato de los datos, los algoritmos y funcionalidades que se encuentran implementados, los paquetes de los que se compone y el formato de los ficheros de configuración para ejecutar cualquiera de sus algoritmos de clasificación.

III-A. Formato de los datos

El formato diseñado para la representación de la estructura de un problema MIML está basado en los formatos de datos utilizados por Weka y MULAN. De este modo, un conjunto de datos estará representado mediante dos ficheros:

- Un fichero *xml* cuya función principal es identificar a los atributos del fichero *arff* que representan las etiquetas. De esta manera, las etiquetas no tienen por qué ser necesariamente los últimos atributos que se definan. Además, este fichero permite representar, anidando etiquetas, una jerarquía de clases. A continuación, se muestra un ejemplo con la definición de cuatro etiquetas.

```
<?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>
```

- Un fichero *arff* (*Attribute-Relation File Format*) cuya estructura presenta dos partes, cabecera y datos:
  - La **cabecera** contiene el nombre de la relación junto con una lista de atributos y sus tipos.
    - En la primera línea del fichero se encuentra la sentencia *@relation <relation-name>* que define el nombre del conjunto de datos. Es una cadena de caracteres que, si contiene espacios, debe de ir entrecomillada.
    - A continuación, se definen dos atributos: el primero de tipo nominal que identifica unívocamente a cada bolsa y el segundo atributo, relacional, que contiene los atributos que describen a las instancias. Para definir los atributos se utilizan las sentencias *@attribute <attribute-name><data-type>*. Habrá una línea por atributo:
      - Los tipos de dato numérico se especifican con *numeric*.
      - Si se trata de un conjunto posible de valores nominales debe ir especificado entre llaves y separado por comas. Por ejemplo: $\{valor_1, valor_2, ..., valor_N\}$.
    - Finalmente, se definen los atributos correspondientes a las etiquetas. Estos atributos tienen que ser de tipo binario (nominales con valores 0 y 1).
  - La sección de **datos** comienza con un *@data*. Cada una de las líneas siguientes representará una bolsa. Los atributos deben de estar ordenados según la declaración realizada en la cabecera. El valor de cada atributo se separa por comas y todas las filas deben tener el mismo número de columnas. Los decimales deben de separarse con el punto. El contenido del atributo relacional estará entrecomillado (comillas simples o dobles), separando cada instancia de la bolsa por un $\backslash n$. A continuación, se muestra un ejemplo de fichero *arff* donde cada bolsa está formada por tres atributos de tipo numérico y tiene cuatro etiquetas. La primera bolsa está formada por 3 instancias, y la segunda bolsa por 2 instancias.

```
@relation toy
@attribute id {bag1,bag2}
@attribute bag relational
  @attribute f1 numeric
  @attribute f2 numeric
  @attribute f3 numeric
@end bag
@attribute label1 {0,1}
@attribute label2 {0,1}
@attribute label3 {0,1}
@attribute label4 {0,1}
```

```
@data
bag1,"42,−198,−109\n42,−191,−142\n3,4,6",1,0,0,1
bag2,"12,−98,10\n42,−19,−12",0,1,1,0
```

La librería cuenta con las clases necesarias tanto para representar bolsas individuales (clase *Bag* del paquete *data*) como para representar un conjunto de datos completo (clase *MIMLInstances*). También proporciona medios para guardar un conjunto de datos en sus respectivos ficheros *arff* y *xml* y para obtener diferentes métricas que te permitan estudiarlos en más profundidad (*MIMLStatistics* en *data.statistics*).

III-B. Funcionalidad incluida en la librería

La librería contiene métodos para transformar un conjunto de datos MIML a otro tipo de representación como MI o ML, algoritmos de clasificación MIML y clases para dar soporte al desarrollo de experimentos.

III-B1. Métodos para transformar datos MIML: se han incluido los siguientes métodos:

- Métodos para transformar a MI (formato de datos usado en Weka) [9]:
  - *Binary Relevance Transformation*: transforma un conjunto de datos MIML en tantos conjuntos de datos MI binarios como etiquetas tenga el problema.
  - *Label Powerset Transformation*: transforma un conjunto de datos MIML en uno multiclase en el que cada posible combinación de etiquetas del conjunto de datos original es considerado una clase diferente.
- Métodos para transformar a ML (formato de datos usado en MULAN) [18]:
  - *Arithmetic Transformation*: transforma cada bolsa en una única instancia donde el valor para cada atributo es su valor medio dentro de la bolsa.
  - *Geometric Transformation*: transforma cada bolsa en una única instancia donde el valor para cada atributo es el centro geométrico de sus valores máximo y mínimo dentro de la bolsa.
  - *Min-Max Transformation*: transforma cada bolsa en una única instancia que contiene, para cada atributo, sus valores mínimo y máximo dentro de esa bolsa. Cada instancia está definida por el doble de atributos de los que tenía previamente.

III-B2. Algoritmos de clasificación: se han incluido los siguientes algoritmos:

- *MIMLClassifierMI*: realiza una transformación del problema MIML para obtener un problema MI aplicando una transformación LP o BR. Posteriormente, obtiene un resultado resolviendo el problema con un algoritmo MI que se especifique. Al ser compatible con la librería Weka, en la tabla I se muestra un ejemplo de 15 algoritmos de Weka que podrían ser utilizados directamente este clasificador.
- *MIMLClassifierML*: realiza una transformación del problema MIML para obtener un problema ML aplicando una transformación aritmética, geométrica o min-max. Posteriormente, obtiene un resultado resolviendo el problema con un algoritmo ML que se especifique. Al ser compatible con la librería MULAN, en la tabla I se muestra un ejemplo de 15 algoritmos de MULAN que podrían ser utilizados directamente este clasificador.
- *MIML-kNN* [2]: utiliza los vecinos y referencias más cercanos a una bolsa para estimar las posibles clases a las que pertenece, trabajando directamente sobre datos MIML.

III-B3. Otra funcionalidad: la librería presenta un marco de trabajo para manejar conjuntos de datos MIML, obtener informes, calcular medidas de distancia, realizar experimentos utilizando validación cruzada o indicando los conjuntos de datos de entrenamiento y test, así como las clases e interfaces genéricas y necesarias para desarrollar nuevos clasificadores, que permite que el desarrollo de nuevas propuestas y la realización de un estudio experimental sea una tarea sencilla.

III-C. Arquitectura de la librería

La librería, desarrollada en Java y de código abierto, hace uso de las librerías MULAN y Weka. Se encuentra estructurada en siete paquetes, algunos de ellos organizados en subpaquetes, de acuerdo con su funcionalidad:

- **core**: incluye diferentes funcionalidades que serán utilizadas por otras clases de la librería. Este paquete contiene las clases relacionadas con la configuración de los métodos utilizando ficheros *xml*.
- **core.distance**: contiene diferentes variantes de la distancia de *Hausdorff* para calcular la distancia entre bolsas.
- **data**: incluye clases para trabajar con el formato de datos detallado en la sección III-A. Entre las funcionalidades que soporta se encuentra: cargar en memoria un conjunto de datos, conocer propiedades de los datos como el número de instancias de una bolsa concreta, el número de atributos, el número de bolsas totales, el número de etiquetas, etc. Además, permite acceder tanto a una bolsa particular, como a sus instancias y etiquetas.
- **data.statistics**: incluye clases encargadas de obtener información relevante de los conjuntos de datos MIML. Se considera tanto información referente a MI (atributos por

Tabla I: Algoritmos compatibles

Clasificadores MI (Weka)	Clasificadores ML (MULAN)
CitationKNN	BRkNN
MDD	DMLkNN
MIDD	IBLR
MIBoost	mLkNN
MIEMDD	HOMER
MILR	RAkEL
MINND	EPS
MIOptimalBall	ECC
MIRI	MLStacking
MISMO	BR
MISVM	LP
MITI	PS
MIWrapper	CC
SimpleMI	RPC

bolsa, número medio de instancias por bolsa, número de instancias mínimo y máximo por bolsa, etc.), como a ML (frecuencia de los conjuntos de etiquetas, distribución de etiquetas por bolsa, co-ocurrencias entre etiquetas, etc.).

- **transformation.mimlTOml**: incluye los métodos para transformar un problema MIML en otro ML.
- **transformation.mimlTOmi**: incluye los métodos para transformar un problema MIML en otro MI.
- **mimlclassifier**: incluye las interfaces y las clases abstractas necesarias en el desarrollo de los algoritmos de clasificación para resolver problemas MIML.
- **mimclassifier.mimlTOmi**: incluye los algoritmos de clasificación que resuelven el problema MIML transformando previamente el problema a MI.
- **mimclassifier.mimlTOml**: incluye los algoritmos de clasificación que resuelven el problema MIML transformando previamente el problema a ML.
- **mimclassifier.lazy**: contiene el algoritmo MIMLkNN.
- **report**: incluye clases que permiten generar informes de resultados de los experimentos realizados. La librería incorpora un informe general con principales métricas que se pueden utilizar para evaluar un modelo MIML.
- **tutorial**: incluye ejemplos de uso de las diferentes funcionalidades de la librería: ejecutar y obtener los resultados de un algoritmo de clasificación MIML, transforma un conjunto de datos MIML a ML o MI, etc.
- **tutorial.config**: incluye ejemplos de ficheros de configuración para poder ejecutar y obtener resultados de los algoritmos de clasificación MIML incluidos en la librería.
- **miml**: incluye la clase para ejecutar cualquier clasificador de la librería mediante un fichero de configuración.

Todos los paquetes contienen las interfaces y las clases necesarias para extender su funcionalidad heredando e implementado la interfaz concreta. De este modo, es posible desarrollar de forma sencilla un nuevo método de transformación del conjunto de datos, y/o un nuevo algoritmo de clasificación. La figura 1 muestra un diagrama que representa las diferentes clases de la librería y sus relaciones.

III-D. Ficheros de configuración

Para facilitar la ejecución de los diferentes algoritmos (ver la sección III-B), la librería cuenta con una clase principal *RunAlgorithm*, que junto con un fichero de configuración en formato *xml*, que especifica el clasificador que se va a utilizar, el conjunto de datos y el informe de salida que se desea obtener, ejecuta y obtiene los resultados de cualquier clasificador MIML que se desee emplear. Estos ficheros tienen que seguir el siguiente formato para que la librería, a partir de la clase *ConfigLoader*, sea capaz de leerlos y configurar todo el proceso que se debe de seguir, desde la creación del modelo hasta la generación de los informes con los resultados:

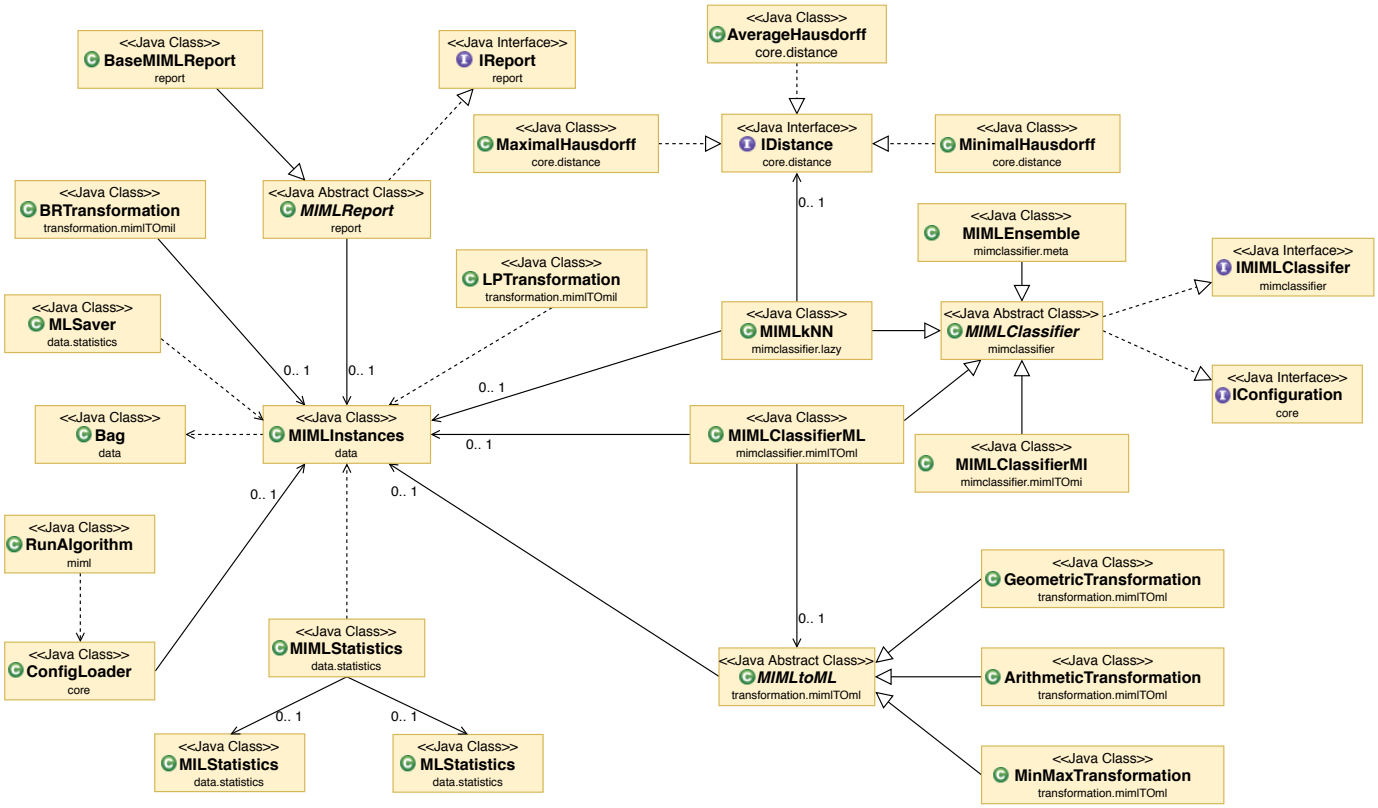

Figura 1: Diagrama de clases de la librería

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

Todos los ficheros deben empezar con el elemento *configuration*. En la etiqueta *classifier*, se especificará el clasificador de la librería que se quiere utilizar. Dentro del clasificador será donde se detalle su configuración: los parámetros específicos que necesite, el método de transformación a utilizar, y/o el clasificador MI o ML que se requiera en las versiones que transforman el problema. La etiqueta *evaluator* detallará el conjunto de datos a utilizar, y si se lleva a cabo una validación cruzada o se especifican directamente los ficheros *train/test*. Finalmente, la etiqueta *report*, indicará el informe que genera el fichero de salida. Esta clase puede ser extendida fácilmente para que los usuarios especifiquen el formato de salida más conveniente.

IV. Ejemplos de uso

La experimentación mediante ficheros *xml* se puede llevar a cabo haciendo uso de la clase ejecutable *RunAlgorithm* pasándole como argumento, a través de la opción *-c*, la ruta del fichero de configuración. A continuación, se mostrarán ejemplos para un conjunto de casos ilustrativos.

IV-A. Ejecución de un algoritmo de clasificación MIML reduciendo el problema a MI

El ejemplo de fichero *xml* para este caso es el siguiente:

```
<configuration>
  <classifier name="mimlclassifier.mimlTOmi.MIMLClassifierMI"
      >
    <multiInstanceClassifier name="weka.classifiers.mi.MISMO">
        <listOptions>-C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V
            -1 -W 1 -K "weka.classifiers.mi.supportVector.
            MIPolyKernel -E 1.0 -C 250007"</listOptions>
        </multiInstanceClassifier>
        <transformMethod name="mulan.classifier.transformation.
            BinaryRelevance"/>
  </classifier>
  <evaluator method="train-test">
    <data>
      <trainFile>data/miml_03_data.arff</trainFile>
      <testFile>data/miml_04_data.arff</testFile>
      <xmlFile>data/miml_03_data.xml</xmlFile>
    </data>
  </evaluator>
  <report name="report.BaseMIMLReport">
      <fileName>results/results.csv</fileName>
    </report>
</configuration>
```

- El clasificador que se desea utilizar debe especificarse en el atributo *name* del elemento *classifier*. Se va a emplear el clasificador *MIMLClassifierMI*. Este clasificador necesita que se especifique en *transformMethod* el método de transformación que se va a utilizar para transformar el problema MIML en un problema MI. En el ejemplo se hace uso del método BR de la librería de Mulan.
- Como configuración de este algoritmo, se debe detallar también en la etiqueta *multiInstanceClassifier* el algoritmo MI que se utilizará una vez realizada la transformación. En este caso se ha utilizado MISMO, un clasificador de la librería Weka para problemas MI. En la etiqueta *listOptions* detallaremos los parámetros que se quieren utilizar en el algoritmo MISMO. Si no se detalla ninguno, se tomarán los valores por defecto establecidos para cada algoritmo.
- El atributo *method* de la etiqueta *evaluator* especifica el proceso de evaluación que se llevará a cabo.
- En el caso de que se vaya a realizar un método *train/test* es necesario especificar en la etiqueta *data*, que pertenece al *evaluator*, la ruta de los ficheros *arff* de entrenamiento y test, así como la del fichero *xml* con las etiquetas.
- Por último, en el elemento *report* se indicará el informe que se desea utilizar, y la configuración de cada informe. En el informe que se ha especificado en el ejemplo, solamente es necesario especificar en *fileName* la ruta donde se quiere guardar el informe de resultados.

IV-B. Ejecución de algoritmo degenerativo utilizando ML

Para la realización de un experimento utilizando ML como vía, un ejemplo de fichero de configuración sería:

```
<configuration>
  <classifier name="mimlclassifier.mimlTOml.MIMLClassifierML"
      >
    <multiLabelClassifier name="mulan.classifier.lazy.MLkNN">
      <parameters>
        <classParameters>int.class</classParameters>
        <classParameters>double.class</classParameters>
        <valueParameters>1</valueParameters>
        <valueParameters>1.0</valueParameters>
      </parameters>
    </multiLabelClassifier>
    <transformMethod name="transformation.mimlTOml.
        ArithmeticTransformation"/>
  </classifier>
  <evaluator method="cross-validation">
    <numFolds>5</numFolds>
    <data>
      <file>data/miml_03_data.arff</file>
      <xmlFile>data/miml_03_data.xml</xmlFile>
    </data>
  </evaluator>
  <report name="report.BaseMIMLReport">
      <fileName>results/results.csv</fileName>
    </report>
</configuration>
```

- En este caso, el elemento *multiInstanceClassifier* es sustituido por *multiLabelClassifier*.
- En caso de que el clasificador ML que se utilice se pueda configurar, los parámetros se deben de especificar de la siguiente forma (para respetar la forma de ejecución que tiene establecida MULAN):
  - En primer lugar se pondrán tantos elementos *classParameters* como parámetros tenga el constructor del clasificador. Su contenido debe de ser el tipo al que

pertenecen dichos parámetros, tienen que estar en el orden en el que se declaren en el constructor.
    - A continuación, se especifican los valores que se quiere que tengan dichos parámetros, indicando cada uno de ellos en un elemento *valueParameters* diferente que, nuevamente, tienen que estar en orden.
- El clasificador que se ha utilizado (*MIMLClassifierML*) se adapta a todos los métodos de transformación disponibles en *transformation.mimlToml*, es necesario especificar en su configuración el elemento *transformMethod*.
- Este experimento se ha configurado para que realice una evaluación basada en validación cruzada, especificado en el atributo *method* la cadena *"cross-validation"*. Es necesario indicar el número de *folds* en el elemento *numFolds* y el fichero que contiene el conjunto de datos. La etiqueta *report* sería similar al ejemplo anterior.

IV-C. Ejecución de algoritmo basado en vecinos más cercanos

Por último, se muestra un ejemplo de fichero *xml* para ejecutar el algoritmo MIMLkNN:

```
<configuration>
  <classifier name="mimlclassifier.lazy.MIMLkNN">
    <nReferences>2</nReferences>
    <nCiters>2</nCiters>
    <metric name="core.distance.AverageHausdorff">
    </metric>
  </classifier>
  <evaluator method="train-test">
    <data>
      <trainFile>data/miml_03_data.arff</trainFile>
      <testFile>data/miml_04_data.arff</testFile>
      <xmlFile>data/miml_03_data.xml</xmlFile>
    </data>
  </evaluator>
  <report name="report.BaseMIMLReport">
      <fileName>results/results.csv</fileName>
    </report>
```

- Con los algoritmos que trabajen directamente con el formato MIML tan solo es necesario especificar en su configuración los parámetros para su ejecución. En este ejemplo, el algoritmo MIML-kNN necesita configurar el número de referencias y citas con los que trabajará, así como la métrica que va a utilizar para medir la distancia entre las bolsas de un *data set*. Las etiquetas *evaluator* y *report*, serían similares a los ejemplos anteriores.

V. Conclusiones

En este trabajo se presenta una librería Java basada en Weka y Mulan para el desarrollo, prueba y comparación de algoritmos dentro del marco de trabajo del aprendizaje MIML. Además de un conjunto de algoritmos MIML, de referencia en la temática, la estructura de la librería permite el desarrollo de nuevas propuestas de forma sencilla. Finalmente, se ha diseñado un formato de datos específico para MIML. La librería es sencilla de utilizar mediante el uso de ficheros de configuración *xml*.

Agradecimientos

Este trabajo está financiado por el proyecto de investigación TIN2017-83445-P del Ministerio de Economía y Competitividad y el Fondo Europeo de Desarrollo Regional.

Referencias

[1] 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.

[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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] T.G. Dietterich, R.H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1):31–71, 1997.

[8] 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.

[9] E. Gibaja and journal=Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery volume=4 number=6 pages=411–444 year=2014 publisher=Wiley Online Library Ventura, S. Multi-label learning: a review of the state of the art and ongoing research.

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[11] 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.

[12] 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.

[13] LAMDA Learning and Mining from DatA. http://lamda.nju.edu.cn/Data.ashx. Accessed: 2018-06-19.

[14] 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.

[15] Z.H. Zhou and M.L. Zhang. Multi-instance multi-label learning with application to scene classification. In *Proceedings of the Advances in neural information processing systems*, pages 1609–1616, 2006.

[16] 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.

[17] F. Herrera, S. Ventura, R. Bello, C. Cornelis, A. Zafra, D. Tarragó, and S. Vluymans. *Multiple Instance Learning. Foundations and Algorithms*. Springer, 230 pages, 2016.

[18] Eibe Frank and Xin Xu. Applying propositional learning algorithms to multi-instance data. pages 1–13, *Computer Science Working Papers. University of Waikato, Department of Computer Science*. 2003.