Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Diseño de algoritmos de remuestreo multi-etiqueta con modelos generativos profundos

Alumno: Miguel Ángel Dávila Romero

Tutor: Francisco Charte Ojeda

María José del Jesus Díaz

Dpto: Informática

UNIVERSIDAD DE JAÉN

D./Dª Francisco Charte Ojeda y D./Dª María José del Jesus Díaz, tutor(es) del Trabajo Fin de Grado titulado: **Diseño de algoritmos de remuestreo multi-etiqueta con modelos generativos profundos**, que presenta Miguel Ángel Dávila Romero, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2023

El estudiante

DAVILA ROMERO MIGUEL ANGEL - 77691852Z

Firmado digitalmente por DAVILA ROMERO MIGUEL ANGEL - 77691852Z
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-77691852Z, givenName=MIGUEL ANGEL, sn=DAVILA ROMERO, cn=DAVILA ROMERO MIGUEL ANGEL - 77691852Z
Fecha: 2023.06.30 19:43:29 +02'00'

Miguel Ángel Dávila Romero

Los tutores

CHARTE OJEDA FRANCISCO - 25996290B

Firmado digitalmente por CHARTE OJEDA FRANCISCO - 25996290B
Fecha: 2023.06.30 20:42:18 +02'00'

Francisco Charte Ojeda

DEL JESUS DIAZ MARIA JOSE - 26485651R

Firmado digitalmente por DEL JESUS DIAZ MARIA JOSE - 26485651R
Fecha: 2023.06.30 20:04:16 +02'00'

María José del Jesus Díaz

Agradecimientos

Me gustaría comenzar esta memoria agradeciendo a Francisco Charte Ojeda y a María José del Jesus Díaz, a quienes he tenido la suerte de conocer primero como profesores, y, posteriormente, como tutores de este trabajo: el más importante, hasta ahora, de mi vida académica. Desde septiembre de hace ya dos años, hasta hoy, he estado en las mejores manos.

Como en todo lo que haga, también debo agradecer enormemente a mi familia: a mis padres, Sebastián y María del Carmen, y a mis hermanos, Charo y Manuel. Ellos me han apoyado y se han interesado por mi trabajo, a pesar de la poca claridad que era capaz de ofrecerles al explicarles el contenido del mismo.

De igual manera, quiero agradecer a mis amigos, quienes, esparcidos en este momento por toda Europa, también me han ayudado a relajarme y despejarme cuando era importante hacerlo. Por supuesto, agradezco también a mi novia, María Isabel, a la que podría incluir tanto en este párrafo como en el anterior, y con quien he compartido la experiencia de llevar a cabo el trabajo de fin de grado, cada uno del suyo.

Igualmente, he compartido este proceso con mis compañeros de clase, a quienes he ido conociendo a lo largo de estos cuatro años, y que ahora considero amigos. También ellos me han ayudado en este proceso.

Por último, quiero agradecer su labor a todos los profesores que me han impartido clase a lo largo de estos cuatro años de grado. Todos ellos me han guiado hasta aquí, ayudándome cuando lo necesitaba, y les doy las gracias por ello.

Tabla de contenidos

Lista de figuras

Lista de tablas

Lista de algoritmos

Capítulo 1

INTRODUCCIÓN

En este primer capítulo se presenta el contenido de este trabajo de fin de grado, en el cual se propone aplicar los nuevos **modelos generativos profundos** (actualmente famosos por la generación de imágenes y texto) a **datos tabulares multietiqueta**, como técnica de remuestreo.

Se explicarán aspectos relativos al trabajo, de modo que, una vez finalizado el capítulo, se puedan presentar los antecedentes que preceden al mismo.

1.1. Fundamentos

En este primer apartado se introducirá una serie de conceptos necesarios para entender el problema a abordar y las soluciones propuestas. Estos conceptos están relacionados con el campo de la **inteligencia artificial** (IA) y en concreto, del **aprendizaje automático** o *machine learning* (ML).

Para llevar a cabo esta breve revisión, se ha seguido la estructura de la introducción de la tesis *Nuevos métodos híbridos de computación flexible para clasificación multietiqueta* [1].

1.1.1. KDD (*Knowledge Discovery in Databases*)

El volumen de datos generados diariamente en el mundo es inmenso. La irrupción de Internet como el pilar de la sociedad en la que vivimos trajo consigo un **flujo de**

datos incesante, que se presenta en distintos formatos: texto, imagen, vídeo, audio, etc. Teniendo en cuenta la abundancia de recursos disponibles en la web (redes sociales, *blogs*, revistas, plataformas de *streaming*, entre otros muchos), la cantidad de datos existentes es prácticamente infinita.

Resulta, por tanto, inevitable la necesidad de usar esos datos para obtener conocimiento. No obstante, es impensable que una o varias personas sean capaces de trabajar con tal cantidad de datos. En este contexto surge el concepto de KDD (*Knowledge Discovery in Databases*) o, en español, **descubrimiento de conocimiento en bases de datos** [2].

El KDD es un procedimiento que consta de diferentes etapas, entre las cuales se puede avanzar o retroceder. Dichas etapas son las siguientes:

1. **Comprensión del dominio:** llevada a cabo en colaboración con expertos en la materia, en esta fase se estudia el problema a resolver, los objetivos a conseguir, el tipo de datos a usar o integrar, etc.

2. **Recolección de datos:** en este paso se recopilan los datos, procedentes de una fuente de origen. Esta fuente puede ser desde un conjunto de sensores (por ejemplo, de tráfico o de calidad del aire) hasta una base de datos ya existente (por ejemplo, las calificaciones de los estudiantes de un grado).

3. **Preprocesamiento:** en esta etapa [3] se introducen cambios en el conjunto de datos o *dataset*, con el fin de eliminar deficiencias de los datos, para así optimizar el funcionamiento de los algoritmos de minería de datos. En este proceso, el cual puede acaparar entre el 80 y el 90 % del proceso de KDD, encontramos varias subetapas:

    - **Integración:** en esta fase, se opera sobre los datos para garantizar la homogeneidad de los mismos. Por ejemplo, expresando los atributos que representan magnitudes con las mismas unidades.
    - **Limpieza:** este proceso [4] consiste, como su nombre indica, en limpiar los datos para conseguir un mejor funcionamiento de los algoritmos, tratando los **valores perdidos** [5], que son aquellos que no se encuentran disponibles; el **ruido** [6], esto es, valores erróneos que se han introducido en la base de datos (por errores en el proceso de medida, por ejemplo); y los **valores extremos** u *outliers* [7], término que designa aquellos valores de casos muy aislados que, aunque no son ruido, pueden confundir a los algoritmos. Hay varias maneras de limpiar estos datos: se pueden imputar,

introduciendo valores como la media, la moda o incluso una predicción llevada a cabo por otro modelo; o bien eliminarse.

- **Transformación:** se trata de operar sobre los datos para conseguir un formato que mejore el funcionamiento del algoritmo de minería de datos. Por ejemplo, se incluyen en esta subetapa la **discretización** [8], que consiste en convertir valores cuantitativos a cualitativos, o la **normalización** [9], cuyo objetivo es llevar todos los valores a una escala común.

4. **Reducción de dimensionalidad:** en esta etapa se intenta reducir el volumen de los datos del conjunto. Para ello, se pueden seleccionar los **atributos** que más nos interesen [10], eliminando aquellos irrelevantes (poca varianza) o redundantes (por ejemplo, que son combinación lineal de otros). También se pueden seleccionar las **instancias** del conjunto de datos más representativas [11]. Se trata de una fase **opcional** dentro del proceso de KDD, puesto que, en muchos casos, se opera con el conjunto de datos original, sin necesidad de modificar atributos o instancias.

5. **Minería:** se trata del análisis de los datos propiamente dicho. Se extenderá este concepto en el siguiente apartado.

6. **Evaluación:** consiste en determinar si los resultados obtenidos han sido lo suficientemente buenos, pudiendo volver a etapas previas del proceso para mejorarlos. Se explorará más a fondo esta fase en la sección 2.3.

1.1.2. Minería de datos

La **minería de datos** o *data mining* [2] es la etapa más significativa del proceso de KDD. Tanto es así que, comúnmente, se emplea este término para referirse a todo el proceso.

Como se ha mencionado con anterioridad, la minería de datos es la fase en la cual se emplean algoritmos para analizar los datos (clasificarlos, agruparlos, etc). A continuación se detallan las características de esta etapa más a fondo.

Enfoques del aprendizaje automático

Podemos distinguir diversos tipos de algoritmos de minería de datos, atendiendo a dos criterios:

- Según la información que precisen los algoritmos:
  - **Aprendizaje supervisado:** los datos han debido ser previamente etiquetados, normalmente por expertos en la materia. Los algoritmos [12] buscan entonces, para los datos del conjunto, patrones que relacionen los atributos con sus etiquetas.
  - **Aprendizaje no supervisado:** los datos no tienen por qué estar etiquetados, por lo que los algoritmos [13] detectan patrones atendiendo únicamente a los atributos, su estructura, distribución, relaciones entre ellos, etc.
  - **Aprendizaje semi-supervisado:** este tipo de aprendizaje [14] es una combinación de los anteriores. Dentro del conjunto hay datos que están etiquetados, a partir de los cuales se induce el modelo de clasificación, y posteriormente se refina el modelo con las muestras no etiquetadas.
  - **Aprendizaje por refuerzo:** en este caso [15], no existe el concepto de etiqueta de un dato, sino que los sistemas aprenden según una recompensa que se les otorga en función de sus acciones.
- Según los objetivos de los algoritmos:
  - **Minería de datos descriptiva:** su objetivo es describir la estructura de los datos y sus relaciones. Se suele asociar al aprendizaje no supervisado, aunque también existen modelos descriptivos supervisados, como los de descubrimiento de subgrupos y otros relacionados con ellos.
  - **Minería de datos predictiva:** su objetivo es generar un modelo para predecir un valor de salida categórico (clasificación) o numérico (regresión). Se suele asociar al aprendizaje supervisado.
  - **Minería de datos generativa:** aunque en la literatura aún no se reconoce como un enfoque independiente, las tareas de minería de datos cuyo objetivo es producir nuevos datos sintéticos, por ejemplo la generación de imágenes [16] o texto [17], no se integran en ninguno de los dos tipos anteriores. Es por ese motivo, y debido a que este trabajo se centra en dicho enfoque, que se ha decidido enmarcar los algoritmos generativos dentro de un nuevo tipo de minería de datos.

Tareas de la minería de datos

Los algoritmos de minería de datos pueden programarse para llevar a cabo distintas tareas, como:

- **Clasificación:** consiste en asignar a los datos una etiqueta categórica (habitualmente discreta y sin orden, aunque existen otros tipos de clasificación [18]). Nos centraremos en esta tarea en el apartado 2.1.
- **Regresión:** se trata de predecir una o varias salidas numéricas continuas, no categóricas [19].
- **Tratamiento de series temporales:** consiste en el análisis de datos que tienen una relación cronológica [20].
- **Agrupamiento:** su objetivo es agrupar datos no etiquetados según el grado de similitud [21]. Es una tarea útil para definir conjuntos de instancias que tienen aspectos en común, detectar *outliers*, o mejorar las fronteras entre grupos tras la clasificación.
- **Asociación:** consiste en descubrir relaciones (asociaciones) entre datos [22], muy útiles para sistemas de recomendación, o para eliminar atributos innecesarios, con una consecuente reducción de la dimensionalidad. Es una tarea frecuente del aprendizaje no supervisado.
- **Optimización:** se trata de la búsqueda de la mejor solución posible para una función objetivo. Dicha solución no necesariamente será la óptima, pues se deberán tener en cuenta restricciones como el tiempo o los recursos disponibles. Para ello, se usan heurísticas [23] y algoritmos como los genéticos [24].

Técnicas de minería de datos

Las tareas mencionadas pueden ser llevadas a cabo empleando distintas **técnicas**. En algunos casos, estas generan modelos de representación del conocimiento, que posteriormente se utilizan para llevar a cabo las tareas mencionadas anteriormente. En otros casos, no se genera ningún modelo. Algunas técnicas de la minería de datos se recogen a continuación.

- **Árboles:** los algoritmos inducen árboles [25] con el fin de resolver tareas como clasificación, regresión y descubrimiento de reglas.
- **Máquinas de Vectores Soporte (SVM):** estos algoritmos [26] se basan en la maximización de la distancia de las clases de datos a una frontera, generando modelos que separan las instancias del conjunto de datos. Se emplea para clasificación binaria y multiclase, así como para regresión.

- **Vecinos cercanos (*kNN*):** no se construye un modelo, sino que "los datos son el modelo" [27]. Es por ello que se enmarca dentro del llamado aprendizaje basado en instancias [28], también conocido como aprendizaje perezoso o *lazy learning*, ya que, hasta que no llega una nueva instancia al modelo, no se lleva a cabo el proceso de inferencia. Se emplean para clasificación, determinando los datos conocidos más cercanos al de entrada para procesar un patrón.

- **Minería de datos frecuentes:** este enfoque [29] consiste en determinar la frecuencia de los elementos de una base de datos a fin de descubrir reglas de asociación, obtener correlaciones, agrupar datos o clasificar.

- **Computación flexible:** este área [30] agrupa los métodos basados en la imitación de procesos de la naturaleza, como por ejemplo:

  - **Técnicas estadísticas:** basadas en el análisis de probabilidades e inferencia bayesiana [31].
  - **Redes neuronales artificiales (*ANN*):** estos modelos [32] basan su funcionamiento en la estructura del cerebro humano. Se emplean para gran variedad de problemas: de clasificación, regresión, reducción de dimensionalidad, agrupamiento, generación de datos sintéticos, etc.
  - **Técnicas evolutivas y bioinspiradas:** este enfoque [33], que incluye algoritmos como los genéticos [24] o los de colonias de hormigas [34], se emplea sobre todo para optimización, y también clasificación, regresión y agrupamiento.
  - **Lógica difusa:** este tipo de lógica [35] trata de modular la cognición humana. No hay una división nítida entre grupos de datos, sino grados de pertenencia con mayor o menor fuerza.

Los nombrados son solo algunos de los procedimientos abarcados por la minería de datos. Estos, y otros muchos, se recogen en el libro *Data Mining and Knowledge Discovery Handbook* [36].

1.1.3. Clasificación multietiqueta

Como se ha indicado anteriormente, la **clasificación** es una tarea dentro de la minería de datos que consiste en **predecir** una etiqueta categórica a partir de los valores de los atributos de entrada de una instancia.

Dentro de la tarea de clasificación se puede distinguir la **clasificación multietiqueta**. En ella, el objetivo es predecir no solo una variable categórica de salida, sino varias, siendo cada una de ellas binaria y dependiente tanto de los atributos de entrada como del resto de etiquetas.

La clasificación multietiqueta es una tarea muy investigada, en la que aún quedan problemas por resolver. En el apartado 2.2 se profundiza en este concepto.

1.1.4. Desbalanceo

Ahora que se ha introducido brevemente la clasificación, se puede presentar uno de los principales problemas al que debe enfrentarse: el **desbalanceo**.

La tarea de clasificación requiere de ejemplos etiquetados que permitan la predicción de la clase (o las etiquetas) para nuevas instancias. Cuando los modelos destinados a esta tarea llevan a cabo su función, lo hacen detectando **patrones** en los datos de entrada, que tienen un efecto en la salida. Por ese motivo, para que no se dé más importancia a unos valores de salida que a otros, el conjunto de datos de ejemplos empleados por el clasificador debe estar **balanceado**, es decir, que tenga el mismo número de instancias para cada una de las posibles clases.

Por tanto, el problema del desbalanceo surge cuando hay más ejemplos de una clase que del resto, y, por tanto, el clasificador en cuestión le da más importancia. Se trata de un problema que aún **no tiene una solución definitiva**, y se explicará en mayor profundidad en el apartado 2.4.

1.2. Motivación

Ya se han introducido conceptos muy importantes dentro de la minería de datos, como el de **clasificación** y, más en concreto, la **clasificación multietiqueta**. Se trata de una herramienta muy potente en el análisis de datos, pudiendo ser de ayuda, por ejemplo, en la detección y tratamiento de enfermedades, o en la automatización de la conducción.

No obstante, para que la clasificación sea tan **eficaz** como su potencial indica, debemos hacer frente a diversos problemas. Uno de los más importantes es el men-

cionado **desbalanceo**, que además se acentúa en los MLD. Explicaremos a fondo el problema del desbalanceo en el apartado 2.4.

En este trabajo de fin de grado se propondrá un **algoritmo** que, mediante la adaptación de los nuevos modelos generativos empleados hasta ahora en imágenes, sea capaz de reducir el impacto del desbalanceo en clasificación multietiqueta.

1.3. Estructura de la memoria

La memoria de este trabajo de fin de grado se organiza de la siguiente manera:

- En el en Capítulo 1 Introducción se revisan conceptos básicos importantes para el desarrollo del trabajo, y se presenta la motivación del mismo. De igual modo, se incluye el glosario de términos empleados en la memoria, así como el cronograma a seguir en la realización del proyecto.

- En el Capítulo 2 Antecedentes se define formalmente la tarea de clasificación, específicamente aquella realizada sobre datos multietiqueta, y se explica cómo se lleva a cabo la evaluación de dicha tarea. También se detalla el problema del desbalanceo en clasificación y las técnicas que permiten abordarlo. Se ahonda en el método del remuestreo y, en concreto, del sobremuestreo, centrándonos en las técnicas de sobremuestreo multietiqueta. Además, se hace un repaso de los nuevos modelos generativos profundos, incluyendo una tabla comparativa entre los mismos que permita establecer cuál de ellos funciona mejor.

- En el Capítulo 3 Objetivos se explica qué se pretende conseguir con el desarrollo de este trabajo de fin de grado.

- En el Capítulo 4 Materiales y métodos se propone el algoritmo a desarrollar, y se plantea el proceso de experimentación para estimar su calidad.

- En el Capítulo 5 Resultados se recogen los resultados de la experimentación, incluyéndose un análisis de los mismos.

- En el Capítulo 6 Conclusiones, por último, se resume el trabajo, se reflexiona sobre la realización del mismo, y se plantean posibles líneas de trabajo relacionadas.

1.4. Glosario de términos

A continuación se recoge un listado de términos que se repiten a lo largo del trabajo, con su traducción al inglés y las siglas empleadas para referirse a ellos en el texto. El listado está ordenado alfabéticamente según la sigla.

Siglas	Inglés	Español
AAE	Adversarial autoencoder	Autoencoder antagónico
ANN	Artificial neural network	Red neuronal artificial
ARFF	Attribute-relation file format	Formato de fichero atributo-relación
BR	Binary relevance	Descomposición binaria de etiquetas
CCR	Combined cleaning and resampling	Combinación de limpieza y remuestreo
CNN	Condensed nearest neighbors	Vecinos más cercanos condensados
DM	Diffusion model	Modelo de difusión
ENN	Edited nearest neighbors	Vecinos más cercanos editados
GAN	Generative adversarial network	Red antagónica generativa
GMM	Gaussian mixture model	Modelo gaussiano mixto
IA	Artificial intelligence	Inteligencia artificial
KDD	Knowledge discovery in databases	Descubrimiento de conocimiento en bases de datos
kNN	K nearest neighbors	K vecinos más cercanos
LP	Label Powerset	Conjunto de todos los conjuntos de etiquetas
ML	Machine learning	Aprendizaje automático
MLC	Multilabel classification	Clasificación multietiqueta
MLD	Multilabel dataset	Conjunto de datos multietiqueta
MLP	Multilayer Perceptron	Perceptrón multicapa
NCR	Neighborhood cleaning rule	Regla de limpieza de vecinos
NF	Normalizing flows	Flujos normalizados
OSS	Open Source Software	Software de código abierto
RBF	Restricted Boltzmann Machine	Máquina de Boltzmann restringida
RkNN	Reverse k nearest neighbors	K vecinos más cercanos reversos
ROS	Random oversampling	Sobremuestreo aleatorio
RUS	Random undersampling	Bajomuestreo aleatorio
SVM	Support vector machine	Máquina de vectores soporte
VAE	Variational autoencoder	Autoencoder variacional
XML	Extensible markup language	Lenguaje de marcas extensible

Tabla. 1.1: Glosario de términos

1.5. Cronograma del proyecto

La estructura temporal del proyecto se refleja en el diagrama de Gantt de la Figura 1.1.

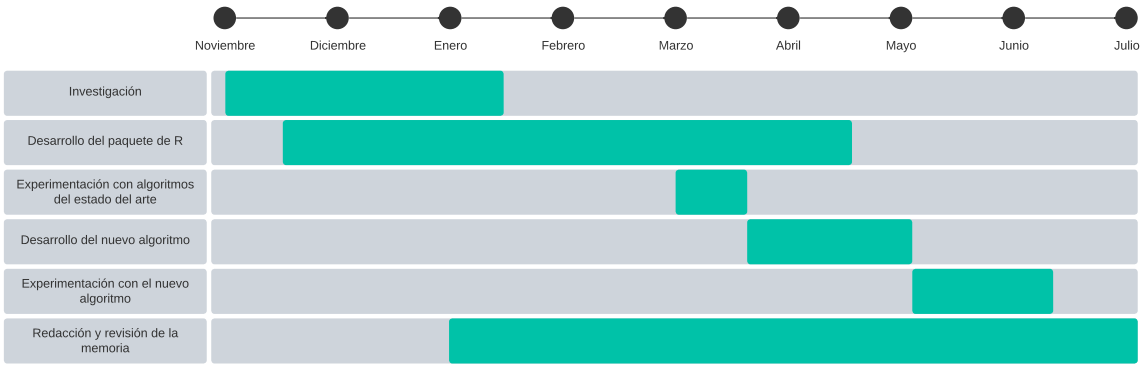

Figura 1.1: Diagrama de Gantt del proyecto

Capítulo 2

ANTECEDENTES

En este segundo capítulo, se profundiza en la tarea de clasificación; en concreto, la clasificación multietiqueta. Tras ello, se explica el problema que se intenta resolver, que es el desbalanceo (en concreto, el desbalanceo multietiqueta), los métodos propuestos con ese fin, y los modelos en los que se basa la propuesta que se presenta en este TFG.

2.1. Clasificación

Como se ha adelantado anteriormente, la clasificación es una de las tareas de la minería de datos, la cual consiste en **asignar** a una instancia, en función de los valores de sus atributos, una **etiqueta**.

Como es obvio, se relaciona con la minería de datos **predictiva** y con el aprendizaje **supervisado**, pues se está aprendiendo, a partir de unos ejemplos etiquetados, a predecir las etiquetas de ejemplos nuevos.

Podemos definir el problema de clasificación de la siguiente manera: partimos de un conjunto de datos D, con $D \subseteq X \times Y$, donde X es el conjunto de posibles valores para los distintos atributos de entrada, e Y es el conjunto de posibles valores de salida. El objetivo de la clasificación es determinar una función $C : X \to Y$, capaz de predecir para cada instancia $x \in X$ del conjunto de datos D el valor de salida $y \in Y$ que le corresponde.

El tipo de clasificación más básico es aquel en el que se utilizan los atributos de una instancia para predecir **un solo atributo de salida**. Por ejemplo, detectar si un

correo es o no *spam* según las palabras que lo componen, predecir el tipo de una flor según sus medidas, etc. En estos dos ejemplos se puede observar una diferencia, y es que en el caso del *spam* la variable a predecir puede tomar solo **dos valores**: sí o no, verdadero o falso. Siguiendo la notación anterior, en estos casos ocurre que $|Y| = 2$. A este tipo de clasificación se la conoce como **clasificación binaria**. Por otro lado, si queremos predecir el tipo de una flor, puede haber más de dos valores posibles y, por tanto, $|Y| > 2$. Esta es la llamada **clasificación multiclase**, en la cual, aunque predecimos una sola variable, esta puede tomar más de dos valores.

La **calidad** de una tarea de clasificación debe ser determinada, con el fin de estimar el funcionamiento del clasificador evaluado, y así compararlo con otros para seleccionar el mejor. Para llevar a cabo esta evaluación, en la literatura se han propuesto diferentes **métricas**, algunas de las cuales se describen en la sección 2.3.

En resumen, tal y como se ha explicado, podemos entender el proceso de clasificación como la predicción, para un ejemplo, de un valor de salida discreto o clase, de entre un conjunto de dos o más valores posibles, pudiendo así distinguir entre clasificación binaria o multiclase.

Pero, ¿qué ocurre cuando se quiere predecir **más de una variable**? Por ejemplo, las emociones que transmite una pieza musical. No estamos limitados a una sola emoción: podemos sentirnos, a la vez, relajados y felices, por ejemplo. Tenemos, por tanto, varios atributos de salida.

En este caso, las variables a predecir se denominan etiquetas, y el procedimiento se conoce como **clasificación multietiqueta**.

2.2. Clasificación multietiqueta

La **clasificación multietiqueta** (*Multilabel classification* o MLC) es un problema muy presente en la minería de datos, pues es normal que un mismo objeto pueda poseer varios atributos que se desean conocer. Por ejemplo, un texto puede pertenecer a varios géneros a la vez: puede ser narrativo o expositivo, y a la vez puede ser objetivo o subjetivo, o periodístico, o literario, etc. Son muchas las variables que se pueden predecir.

Cada una de las etiquetas puede estar presente en una instancia o no. Es decir, las etiquetas son **binarias**: pueden tener solo dos valores. El conjunto de los valores de todas las etiquetas de una instancia concreta constituye su *labelset* o **conjunto de**

etiquetas, representado por la letra L. Para un número de etiquetas $|L|$, el número de *labelsets* posibles al combinar los valores de las diferentes etiquetas es igual a $2^{|L|}$.

Por tanto, el problema de MLC trata de encontrar una función $C : X \rightarrow Y$, donde $Y \subseteq powerset(L)$. Es decir, para cada instancia, el objetivo de la MLC es predecir su *labelset*.

La Figura 2.1 ilustra la diferencia entre la clasificación tradicional (binaria y multiclase) y la MLC. Como se puede observar, en clasificación binaria una instancia puede ser de un color o de otro (dos valores posibles para un solo atributo). Por otro lado, en clasificación multiclase una instancia puede ser de 3 colores diferentes (más de dos valores posibles para un solo atributo). Por último, en MLC, un mismo objeto puede pertenecer o no a varias etiquetas (dos valores posibles para varios atributos).

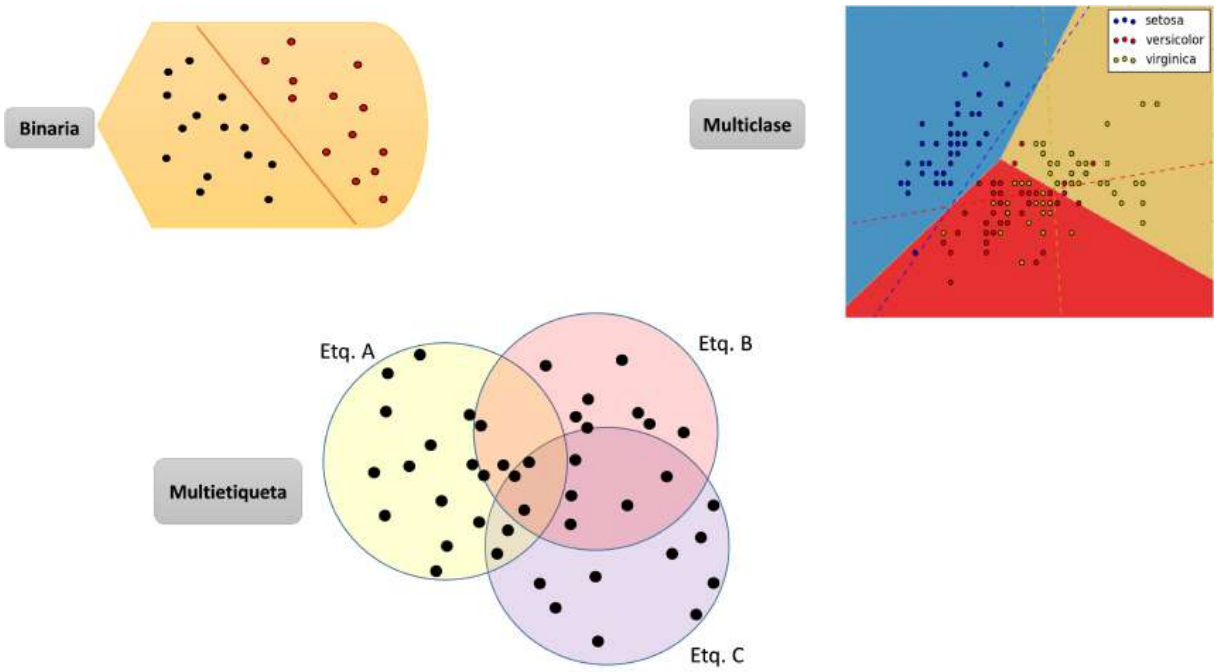

Figura 2.1: Diferencia entre tipos de clasificación. Fuente: [1].

La naturaleza de los conjuntos de datos multietiqueta (*multilabel datasets* o MLD) permite su **transformación** en conjuntos de datos binarios o multiclase, mediante dos procedimientos conocidos como *Binary Relevance* y *Label powerset*.

2.2.1. *Binary Relevance* (BR)

La transformación de un conjunto de datos multietiqueta en binario se conoce como *Binary Relevance* o BR [37]. En realidad, para un MLD con $|L|$ etiquetas, se obtienen

$|L|$ conjuntos de datos binarios distintos, donde en cada uno de ellos, la clase a predecir es una de las etiquetas. Se puede predecir cada etiqueta de forma individual, y posteriormente unir dichas predicciones.

Una ventaja de esta transformación es que, a no ser que el número de etiquetas sea demasiado grande, el proceso de clasificación tiene una **baja complejidad computacional**.

No obstante, esta transformación trae consigo una **pérdida de información**, pues, al hacer la predicción de cada etiqueta independiente, no se están teniendo en cuenta las relaciones entre etiquetas, que son una fuente de información importante en los MLD. Debido a este problema, han surgido varias propuestas [38] que tratan de solucionar las carencias de BR.

En la Figura 2.2 se incluye una ilustración que explica de manera gráfica esta transformación.

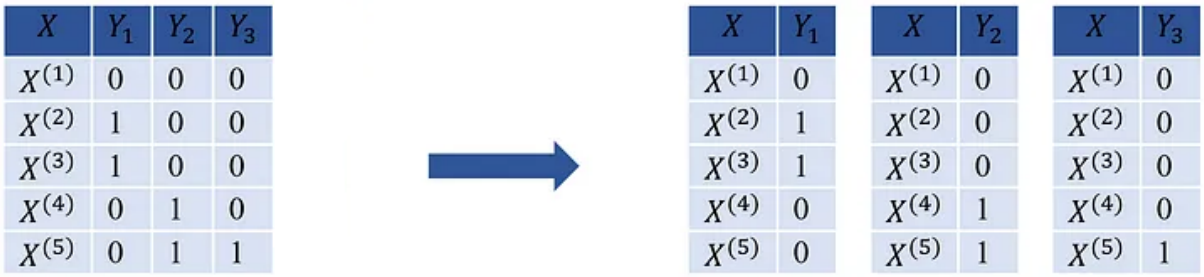

Figura 2.2: Transformación BR. Fuente: [39].

2.2.2. *Label Powerset* (*LP*)

Este método de transformación [37] consiste en tomar cada conjunto de etiquetas (*labelset*) del conjunto de datos como una clase distinta. De este modo, el resultado es un conjunto de datos multiclase. El proceso *LP* se ilustra en la Figura 2.3.

A diferencia de BR, con esta transformación sí que se tienen en cuenta las **dependencias entre etiquetas**. No obstante, hay un evidente problema, y es que si nos encontramos ante un conjunto de datos con un número de etiquetas elevado, el número de *labelsets* posibles es **demasiado grande**, afectando de manera negativa al aprendizaje. De hecho, se puede llegar al extremo de que cada una de las instancias del MLD tenga un *labelset* distinto.

Aunque muy útiles, las transformaciones de los MLD no son la única opción posible para llevar a cabo MLC. De igual modo que se pueden adaptar los datos, también

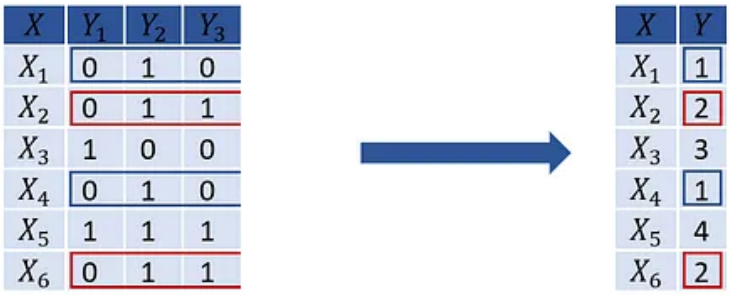

Figura 2.3: Transformación LP. Fuente: [39].

se han **adaptado clasificadores** para que sean capaces de llevar a cabo esta tarea. Algunas adaptaciones de clasificadores tradicionales se revisan en el mencionado libro *Multilabel Classification: Problem Analysis, Metrics and Techniques* [40].

2.2.3. Métricas de caracterización de MLD

Como se ha explicado, la principal diferencia entre los MLD y los conjuntos de datos tradicionales es el hecho de que una instancia pueda pertenecer a varias etiquetas. Por tanto, la mayor parte de las métricas específicas de estos tipos de datos van a reflejar este fenómeno.

La métrica más sencilla es la **cardinalidad de etiquetas** ($Card$), que devuelve el número medio de etiquetas que presenta cada instancia del MLD. En la Ecuación 2.1 se recoge la fórmula para calcularla.

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

Un dataset con $Card$ cercana a 1 implicará que los datos no tienen una naturaleza multietiqueta tan pronunciada, ajustándose más a un conjunto de datos multiclase, y que, por tanto, los métodos de transformación presentados anteriormente serán efectivos para la clasificación. Cuanto mayor sea la cardinalidad, mayor será la dificultad a la hora de clasificar.

No obstante, $Card$ es una medida **con dimensión**. En concreto, su unidad es el número de etiquetas. Por tanto, el número de etiquetas de un MLD influirá en su valor, siendo mayor en conjuntos de datos con muchas etiquetas, y, por tanto, no será preciso comparar MLD distintos usando esta medida.

Es por eso que surge la medida de **densidad de etiquetas** de un MLD ($Dens$), que sí que es **adimensional**, y se calcula dividiendo la cardinalidad del MLD entre el número de etiquetas del mismo, como se muestra en la Ecuación 2.2.

$$Dens(D) = \frac{1}{|L|}\frac{1}{|D|}\sum_{i=1}^{|D|}|Y_i| \tag{2.2}$$

Por tanto, un MLD con una mayor densidad indica que las etiquetas están bien representadas en cada instancia (con independencia del número total de las mismas), mientras que una densidad baja implica que las etiquetas están muy dispersas entre las instancias.

Por otro lado, existe la **diversidad de etiquetas** de un MLD (Div), la cual mide el número de *labelsets* distintos presentes en el mismo. El número máximo de *labelsets* será el mínimo entre el número de instancias del MLD, y $2^{|L|}$. Por tanto, cuanto más se aproxime Div a ese máximo, más irregular es la distribución entre las etiquetas (menos correlación entre las mismas). En la Ecuación 2.3 se muestra la fórmula de esta métrica, que puede normalizarse dividiendo entre el número de instancias.

$$Div(D) = \sum_{i=1}^{|D|} distinct(y_i) \tag{2.3}$$

Estas son las métricas más básicas y extendidas que permiten describir un MLD. En el libro *Multilabel Classification: Problem Analysis, Metrics and Techniques* [40] se explican estas y otras en mayor profundidad.

Las presentadas son métricas destinadas a describir aspectos a tener en cuenta sobre un MLD. Sin embargo, también son necesarias métricas para determinar la bondad del funcionamiento de los modelos empleados para MLC. Debido a la peculiaridad de los MLD, se requieren **adaptaciones**, o incluso **métricas específicas**, para llevar a cabo la evaluación de un clasificador, como se explica en el siguiente apartado.

2.3. Evaluación

Ya se han repasado algunas de las técnicas dentro de la minería de datos. No obstante, al llevar a cabo un proyecto, se debe incluir un proceso de **experimentación**

con el fin comparar los modelos generados por dichas técnicas y seleccionar la mejor opción. Este proceso constituye la etapa de **evaluación** [41].

Para estimar el rendimiento de un modelo de minería de datos, se debe afrontar el llamado **problema de generalización** [41]. La idea detrás de dicho problema es la siguiente: el objetivo de un modelo, entrenado con un conjunto de datos, es ser capaz de aplicar el conocimiento adquirido con datos nuevos, no vistos aún, por lo que se debe simular este hecho en la evaluación. En otras palabras, si se estimara la calidad del funcionamiento de un modelo con los datos con los que este fue entrenado, los resultados estarán **sesgados**, pues puede que el modelo se haya ajustado en exceso a los datos de entrenamiento. Este fenómeno se conoce como **sobreaprendizaje** [42].

Para afrontar el problema de generalización, se evalúa el rendimiento del modelo con datos **distintos** de aquellos con los que fue entrenado. La manera más sencilla de llevar a cabo esta tarea es particionar el conjunto de datos de manera aleatoria, dando lugar a un subconjunto de **entrenamiento** y otro de **test**, siendo el primero empleado para entrenar al modelo, y el segundo para evaluarlo. Este enfoque se conoce como *hold-out* [43], y es útil en conjuntos de datos muy grandes, en los que los subconjuntos generados son suficientemente representativos.

No obstante, *hold-out* tiene un problema, presente sobre todo en conjuntos de datos con pocas instancias, y es que es posible que, al particionar, los conjuntos generados no sigan la distribución del conjunto de datos original. Dicho de otra forma, sería posible que en uno de los conjuntos coincidieran instancias con ruido, o con valores extremos, que **falsearan los resultados**. Frente a este problema, surgen otros enfoques, como la **validación cruzada** o *cross-validation* [44], que consiste en llevar a cabo un número determinado de particionamientos y evaluaciones, agregando los resultados. De este modo, la probabilidad de que los resultados no reflejen el comportamiento real del modelo se disminuye en gran medida. Existe una variante del método de validación cruzada, llamado *leave-one-out* [45], que en cada iteración separa una única instancia para test y mantiene el resto del conjunto para entrenamiento. Dicha variante puede resultar interesante en conjuntos de datos pequeños, pero es **computacionalmente inviable** en grandes conjuntos de datos.

En definitiva, los enfoques explicados son formas diferentes de estimar el comportamiento del modelo en datos nuevos, no utilizados para su entrenamiento. Los resultados mostrados al usar *hold-out* pueden estar muy condicionados por las características de la partición llevada a cabo, mientras que métodos como la validación cruzada pueden reducir ese sesgo.

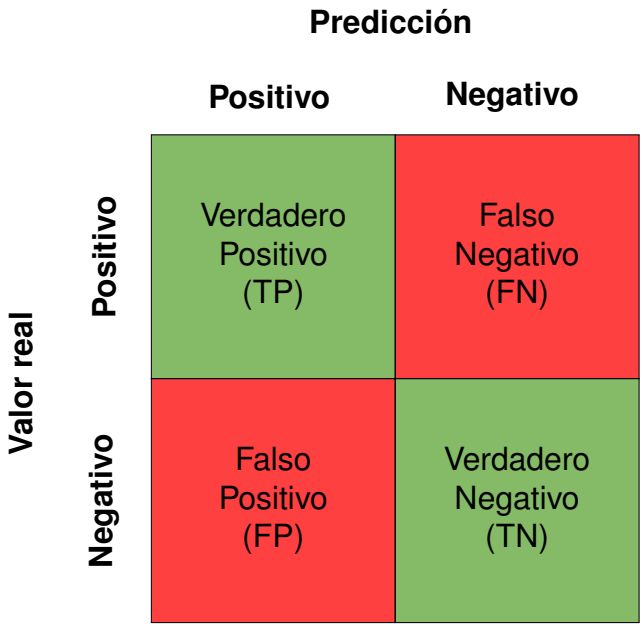

Figura 2.4: Matriz de confusión en clasificación binaria.

Para estimar la calidad de un modelo, se emplean **métricas de calidad**, las cuales son capaces de cuantificar la bondad de una tarea de minería de datos. Existen distintas métricas, que se usarán en función de la tarea que se desee evaluar.

2.3.1. Métricas de calidad para clasificación tradicional

Como se ha indicado anteriormente, se han propuesto muchas métricas con el fin de estimar la calidad de un modelo de clasificación, gran parte de ellas basadas en el **número de aciertos y fallos**, desglosados en verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Los dos primeros son aciertos, y los dos últimos son fallos. Estos conceptos pueden observarse fácilmente en una **matriz de confusión**, como la que se muestra en la Figura 2.4.

Como se puede apreciar, esta matriz representa un proceso de clasificación binaria, en la que una instancia puede ser clasificada como positiva o negativa. No obstante, esta representación es extensible a clasificación multiclase, pues simplemente cambiará la dimensión de la matriz, que será igual al número de clases posibles. De este modo, al igual que en la matriz binaria, cuando la predicción coincide con el valor real (lo cual ocurre en la diagonal principal de la matriz), podemos contarlo como un acierto del clasificador, mientras que el resto de casos son fallos. Las métricas se calcularán de manera individual para cada clase, y serán agregadas posteriormente.

La métrica de evaluación más sencilla es el **ratio de acierto** o $Accuracy$, que se calcula según la Ecuación 2.4, y mide la proporción de instancias sobre las cuales la clasificación ha sido correcta, es decir, coincidente con el valor real.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.4}$$

No obstante, el ratio de acierto es una medida **poco fiable** del rendimiento de un clasificador, pues dependerá de la distribución de la clase. Por ejemplo, si hay pocos ejemplos de una de las clases, un clasificador que siempre devolviera la clase mayoritaria como predicción tendría un valor de $Accuracy$ alto. Este problema se conoce como **desbalanceo**, y se explica en la sección 2.4.

Frente a los problemas de $Accuracy$ surgen otras métricas como la **precisión** y la **cobertura**, o $Precision$ y $Recall$. La primera determina el ratio de ejemplos que, de entre los que se han clasificado como positivos, eran realmente positivos. Por otro lado, la segunda mide el ratio de ejemplos que, de entre los que eran realmente positivos, se han clasificado como tal. Las fórmulas para calcular estas dos métricas se recogen en la Ecuación 2.5.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + TN} \tag{2.5}$$

Ambas están relacionadas entre sí, e interesa que sean lo más altas posibles. Para unificarlas en una única métrica surgió $F\text{-}Measure$, que es la **media armónica** entre $Precision$ y $Recall$, y que se calcula según la expresión que se muestra en la Ecuación 2.6.

$$F\text{-}Measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.6}$$

Estas son solo las métricas más básicas para evaluar el funcionamiento de un clasificador. Hay muchas otras [46], como el área bajo la curva ROC, conocida como AUC, o $Kappa$, métricas más complejas, pero que son menos sensibles a la distribución de las clases [47].

2.3.2. Métricas de calidad para MLC

A diferencia de la clasificación tradicional, la MLC es más **difícil de evaluar**, pues la predicción de un conjunto de etiquetas para una instancia, a diferencia de la predicción de una clase en clasificación tradicional, no es simplemente correcta o incorrecta, pues puede haber etiquetas cuya presencia se haya predicho correctamente y otras que no, haciendo difícil estimar qué clasificación es mejor que otra.

Es por ello que existen diversos enfoques a la hora de diseñar métricas de evaluación, pudiendo distinguir entre:

- **Métricas basadas en instancias** [48] **y métricas basadas en etiquetas** [49]. Las primeras reciben la predicción de un conjunto de etiquetas por cada instancia y deben especificar, para cada una de esas predicciones, una medida de evaluación, agregándolas todas finalmente. Por su parte, las segundas determinan métricas a nivel de cada etiqueta y, posteriormente llevan a cabo una agregación de tipo *macro* o *micro*. La agregación *macro* consiste en calcular una medida por separado para cada etiqueta, y después agregar dichos valores. Por otro lado, la agregación *micro* consiste en acumular los resultados de cada etiqueta y, finalmente, calcular la métrica. En las ecuaciones 2.7 y 2.8 se recogen fórmulas genéricas para calcular métricas con sendos tipos de agregación.

$$MacroMet = \frac{1}{|L|}\sum_{l=1}^{|L|} evalMet(TP_l, FP_l, TN_l, FN_l) \tag{2.7}$$

$$MicroMet = evalMet(\sum_{l=1}^{|L|} TP_l, \sum_{l=1}^{|L|} FP_l, \sum_{l=1}^{|L|} TN_l, \sum_{l=1}^{|L|} FN_l) \tag{2.8}$$

- **Métricas basadas en bipartición y métricas basadas en ranking**. Estos dos tipos de métricas surgen de las dos posibles maneras en las que se puede presentar la salida de un clasificador. Por una parte, puede directamente especificar qué etiquetas están presentes en una muestra y cuáles no. En ese caso, se puede simplemente contar el número de aciertos y fallos, aplicando métricas de bipartición. Por otro lado, la salida del clasificador también puede ser un ranking, en el que a cada etiqueta se le asocie un grado de relevancia, en función de su probabilidad de presencia. Sobre este tipo de salida, pueden aplicarse métricas basadas en ranking. No obstante, también un ranking puede convertirse en bipartición de manera sencilla, usando un umbral para el grado de relevancia. Así, también podrán aplicarse métricas basadas en bipartición.

Combinando los tipos anteriores surge una taxonomía en la que se enmarcan métricas ya vistas, como $Accuracy$, $Precision$, $Recall$ y $F\text{-}Measure$, las cuales son basadas en bipartición, y, según cómo se agreguen, pueden tener un enfoque basado en ejemplos o basado en etiquetas. Otra métrica bastante usada en MLC, la cual es basada en ejemplos, y en bipartición, es $Hamming\ Loss$ (HL), la cual es, probablemente, la más usada en la literatura, y cuya fórmula se muestra en la Ecuación 2.9, donde Y_i es el conjunto de etiquetas real de la instancia i, Z_i es el predicho por el clasificador, y el operador Δ contabiliza el número de diferencias entre ambos.

$$HammingLoss = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{|Y_i \Delta Z_i|}{|L|} \tag{2.9}$$

Como se puede observar, se contabilizan las **diferencias** entre el conjunto de etiquetas real y el predicho; es decir, los fallos. Esta métrica es, por tanto, la complementaria de $Accuracy$, la cual contabiliza los aciertos. Por tanto, nos interesa que el valor de $Hamming\ Loss$ sea lo menor posible.

Como caso más extremo, dentro de las métricas basadas en instancias y bipartición, tenemos la métrica $Subset\ Accuracy$, la cual únicamente contabiliza como acierto aquellos casos en los que el conjunto de etiquetas predicho es **exactamente igual que el real**, es decir, las predicciones de presencia o ausencia de todas las etiquetas han sido acertadas. La Ecuación 2.10 recoge la fórmula para el cálculo de esta métrica.

$$SubsetAccuracy = \frac{1}{|D|}\sum_{i=1}^{|D|}[[Y_i = Z_i]] \tag{2.10}$$

Por otro lado están las métricas basadas en **ranking**, las cuales, como se ha explicado, se pueden utilizar cuando la salida del clasificador no es binaria, sino una ordenación de las etiquetas, o un conjunto de probabilidades de pertenencia para cada una de ellas. Una de las más importantes es $Ranking\ Loss$ (RL), basada en instancias, la cual realiza un conteo, para cada par de etiquetas, de aquellos casos en los que una etiqueta que no está presente en el conjunto de etiquetas real queda, en el ranking, por delante de otra que sí que está presente. Estos casos se consideran errores, por lo que $Ranking\ Loss$ es una métrica a minimizar. En la Ecuación 2.11 se muestra la fórmula para calcular esta métrica, donde Y_i es el conjunto de etiquetas presentes en la instancia i.

$$RL = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{1}{|Y_i||\overline{Y_i}|}|y_a, y_b : rank(x_i, y_a) > rank(x_i, y_b), (y_a, y_b) \in Y_i \times \overline{Y_i}| \quad (2.11)$$

Otra métrica basada en ranking es $One\ Error$. Esta métrica contabiliza el número de veces que la primera etiqueta forma parte del conjunto de etiquetas real. En la Ecuación 2.12 se recoge la fórmula para calcularla.

$$OneError = \frac{1}{|D|}\sum_{i=1}^{|D|}[[\underset{y \in Z_i}{argmax}(rank(x_i, y)) \notin Y_i]] \quad (2.12)$$

Estas son solo algunas de las métricas que se pueden emplear para cuantificar la bondad del proceso de MLC. En la literatura [50] se han propuesto otras muchas, cada una con características distintas, por lo que, en un trabajo de experimentación, conviene incluir varias.

Una vez se ha explicado con mayor profundidad la tarea de la clasificación multi-etiqueta, se puede también entrar en más detalle sobre uno de sus principales problemas, que dificultan el desempeño de los algoritmos de clasificación. Se trata, tal y como se ha adelantado con anterioridad, del desbalanceo.

2.4. Desbalanceo en clasificación

Como se ha explicado en el apartado 1.1, el entrenamiento de los modelos de minería de datos se lleva a cabo con un conjunto de datos o *dataset*, consistente en una colección de instancias o muestras con una serie de valores para distintos atributos, y un valor para una variable a predecir (o varios valores para varias variables, en el caso de multietiqueta).

Para la tarea concreta de clasificación (pensando de momento en la tradicional), la variable a predecir solo puede tomar una serie **finita** de valores. Por tanto, existirán varias muestras en un mismo conjunto de datos que pertenezcan a la misma clase.

Se dice que un conjunto de datos está **balanceado** cuando el número de instancias que pertenecen a cada una de las clases es el mismo. En la práctica, esto es muy **improbable**, y se considera que un conjunto de datos está balanceado si la distribución de las clases es **aproximadamente uniforme**. Por tanto, un conjunto de datos **desba-**

lanceado es aquel en el que hay diferencias **muy notorias** entre clases que abarcan muchas instancias (las llamadas clases mayoritarias) y clases a las que pertenecen muy pocas muestras (conocidas como clases minoritarias).

Como se ha reconocido en la literatura [51], el mal comportamiento de los clasificadores frente a conjuntos de datos desbalanceados es **inherente** al diseño de los mismos, pues el objetivo es minimizar el **error global** de la predicción. Esto conlleva un evidente beneficio a favor de las clases mayoritarias, pues una buena clasificación de las mismas implica una ratio de acierto mayor, mientras que si se clasifican mal las muestras pertenecientes a clases minoritarias, el ratio de acierto no se ve tan afectado.

Llevando esto a un extremo, en un conjunto de datos que estuviera excesivamente desbalanceado, un clasificador que siempre asignara la clase mayoritaria tendría un buen rendimiento si nos basáramos en el ratio de acierto.

Como agravante, la clase minoritaria es habitualmente el concepto **más importante** a ser aprendido. Por ejemplo, en un conjunto de datos con el historial médico de diversos pacientes, es muy probable que haya más ejemplos de pacientes sanos que de pacientes enfermos. Esto es un problema, teniendo en cuenta que el aprendizaje debe ir enfocado a reconocer la enfermedad.

El desbalanceo puede darse en cualquier tarea de clasificación, por lo que se puede agrupar de igual manera, distinguiendo así entre desbalanceo binario, multiclase y multietiqueta (entre otros, pues existen más tipos de clasificación, los cuales no son relevantes para este trabajo).

2.4.1. Desbalanceo en clasificación tradicional

Dentro de la clasificación tradicional se enmarcan la clasificación binaria y multiclase. El problema de desbalanceo es muy parecido en ambos casos, aunque cada uno tiene características propias que se exponen a continuación.

Desbalanceo binario

Es el caso más sencillo y ocurre cuando, de las dos clases posibles a las que puede pertenecer una instancia, una de ellas es **mayoritaria**. Es decir, un conjunto de datos binario está desbalanceado cuando el número de instancias pertenecientes a una clase es superior al número de instancias pertenecientes a la otra.

La Figura 2.5 muestra de manera visual un conjunto de datos binario desbalanceado. Como se puede observar, hay más instancias de la clase de la izquierda que de la clase de la derecha.

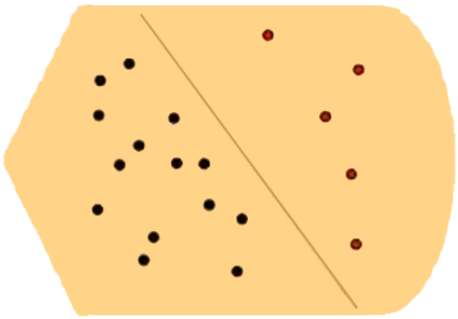

Figura 2.5: Conjunto de datos binario desbalanceado. Elaboración propia a partir de [1].

Desbalanceo multiclase

El desbalanceo multiclase es parecido al desbalanceo binario, solo que en este caso no hay una clase mayoritaria y otra minoritaria, sino que hay **varias** de cada tipo. Es decir, en un conjunto de datos en el que la clase puede tomar, digamos, 10 valores, puede que tres de ellos tengan muchas más instancias que los demás, dos tengan un número de instancias extremadamente bajo, y los cinco restantes sí que se mantengan en un número de instancias más o menos similar.

De nuevo, podemos visualizar el problema de forma gráfica fijándonos en la Figura 2.6. A la vista de la misma, podemos concluir que hay una clase mayoritaria, que es la azul, mientras que las clases roja y amarilla tienen muchos menos ejemplos (son clases minoritarias).

En varias publicaciones [52, 53] se llega a la conclusión de que el problema de desbalanceo es **más difícil** de resolver en conjuntos de datos multiclase que en conjuntos de datos binarios, debido a la posible presencia de múltiples clases minoritarias.

Como ya se ha indicado, el desbalanceo es un problema que afecta negativamente al proceso de clasificación. Resulta, por tanto, interesante, ser capaces de conocer **cómo de desbalanceado** está un conjunto de datos como parte del análisis exploratorio del mismo. Para ello, existen distintas métricas, algunas de las cuales se presentan a continuación.

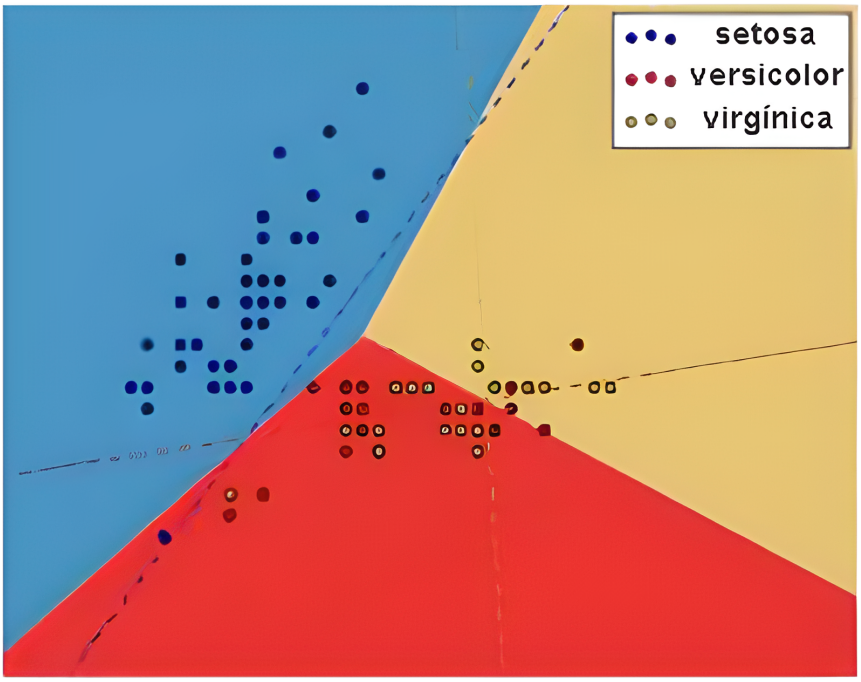

Figura 2.6: Conjunto de datos multiclase desbalanceado. Elaboración propia a partir de [1].

Métricas para medir el desbalanceo en clasificación tradicional

Antes de pasar a un caso peculiar de desbalanceo, como lo es el que se da en conjuntos de datos multietiqueta, y que, por tanto, se diferencia de los dos que se acaban de ver, resulta interesante repasar qué métricas pueden utilizarse para conocer cómo de desbalanceado está un conjunto de datos tradicional.

La métrica más común es el **ratio de desbalanceo** (*Imbalance Ratio* o IR) [54]. Como se muestra en la Ecuación 2.13, esta métrica se calcula de manera sencilla dividiendo el número de instancias de clases mayoritarias entre el número de instancias con clases minoritarias.

$$IR = \frac{N_{\text{maj}}}{N_{\text{mín}}} \tag{2.13}$$

Por tanto, el IR será mayor cuanta mayor sea la diferencia (relativa al tamaño del conjunto de datos) entre el número de muestras pertenecientes a clases mayoritarias y minoritarias, lo cual implica un mayor desbalanceo del conjunto de datos.

No obstante, esta métrica presenta algunos problemas, pues originalmente fue diseñada para conjuntos de datos binarios. Por tanto, no se distingue entre situaciones en las que hay muchas clases mayoritarias o muchas minoritarias, solo importa la diferencia de instancias entre ambos grupos. Sin embargo, resulta interesante reflejar dicha información en nuestra métrica, puesto que los conjuntos de datos con muchas clases minoritarias son más difíciles de abordar.

Es por eso que surgen otras propuestas, como el **grado de desbalanceo** (*Imbalance Degree* o ID) [55]. Este tiene en cuenta la distribución empírica de los datos según la clase, concretamente su distancia respecto a una distribución **uniforme**, que es la que tendría el conjunto de datos si estuviera balanceado.

La fórmula para el cálculo de esta métrica se recoge en la Ecuación 2.14, donde m es el número de clases minoritarias, ζ es la distribución empírica de los datos, $\mathbf{e}$ es una distribución uniforme, d_Δ es una métrica de distancia, y $\boldsymbol{l}_m$ es la distribución con m clases minoritarias que más lejos está de $\mathbf{e}$.

$$ID(\zeta) = \frac{d_\Delta(\zeta, \mathbf{e})}{d_\Delta\left(\boldsymbol{l}_m, \mathbf{e}\right)} + (m-1) \tag{2.14}$$

De este modo, cuanto menor sea el valor de ID, más se aproximará la distribución de las instancias por clase a una uniforme y, por tanto, más balanceado estará el conjunto de datos en cuestión.

Por otro lado, existe la métrica *likelihood-ratio imbalance degree* ($LRID$) [56], la cual pretende resolver algunos problemas introducidos por ID, relacionados sobre todo con el segundo término de la ecuación, que hace que un conjunto de datos menos desbalanceado que otro, pero que tiene más clases minoritarias, tenga un grado de desbalanceo más alto. En definitiva, solo se habla de clases mayoritarias y minoritarias, pero no se tiene en cuenta que el límite entre estas dos denominaciones, a veces, es muy **difuso**. Por su parte, $LRID$ propone basar el grado de desbalanceo, en lugar de en una función de distancia, en la técnica de **inferencia estadística** del *likelihood-ratio* (LR), y eliminar el mencionado segundo término de la ecuación.

La Ecuación 2.15 muestra la fórmula para calcular el $LRID$ de un conjunto de datos, donde se realiza una sumatoria por clase del LR, siendo N el número total de muestras del conjunto de datos, C el número de clases, y n_c es el número de muestras de la clase c. El resto de la nomenclatura se mantiene con respecto a la Ecuación 2.14.

$$LRID = -2\sum_{c=1}^{C} n_c \ln \frac{\mathbf{e}_c}{\zeta_c} = -2\sum_{c=1}^{C} n_c \ln \frac{N}{Cn_c} \tag{2.15}$$

Al igual que con ID, un valor de $LRID$ bajo para un conjunto de datos implica un nivel de desbalanceo bajo.

Se ha visto que tanto ID como $LRID$ se basan en la distribución de las clases en el conjunto de datos. No obstante, este no es el único aspecto importante si lo que queremos reflejar con nuestra métrica es cómo de difícil va a ser clasificar las instancias del mismo. Es por esto que surge una nueva propuesta, el $AdjustedIR$ [57], en la que también se tiene en cuenta la **dimensionalidad** del conjunto de datos.

Suponiendo dos conjuntos de datos iguales, el valor de las métricas presentadas hasta ahora también será igual. Si ahora se añadieran atributos muy discriminativos a uno de los conjuntos de datos, este pasaría a ser más **fácil de clasificar** gracias a dichos atributos. No obstante, el valor de las métricas anteriores se mantendría igual para los dos conjuntos de datos.

Por eso, $AdjustedIR$ tiene también en cuenta los atributos de un conjunto de datos y su poder discriminativo respecto a la clase, por lo que, de alguna manera, se está midiendo cómo de difícil será clasificar las instancias del conjunto de datos. En la Ecuación 2.16 se recoge la fórmula de $AdjustedIR$, donde p^* es el número de atributos discriminativos del conjunto de datos, determinados mediante el test de correlación de *Pearson* [58], y λ es un parámetro para controlar la penalización aplicada sobre el IR.

$$AdjustedIR = IR - \lambda \cdot \log(p^*) \tag{2.16}$$

Por tanto, cuanto mayor es el valor de $AdjustedIR$, más difícil de clasificar serán las instancias del conjuntos de datos.

Una vez presentado el problema de desbalanceo en clasificación tradicional, podemos centrarnos en el problema a resolver en este trabajo: el desbalanceo multietiqueta.

2.4.2. Desbalanceo multietiqueta

El problema del desbalanceo también puede aparecer en conjuntos de datos multietiqueta. De hecho, **es de lo más habitual**, debido a la gran cantidad de etiquetas con las que cuentan los mismos.

Es, por tanto, muy probable que una o varias etiquetas tengan una presencia bastante inferior a la de las demás. El problema que hace que el desbalanceo multietiqueta sea más difícil de abordar que los anteriores es la posible presencia de varias etiquetas en una instancia.

Al poder una instancia presentar varias etiquetas, existe la posibilidad de que, en el *labelset* de la misma, estén presentes tanto etiquetas minoritarias como mayoritarias. Esta situación es muy peculiar y, como veremos, incluso se han propuesto métricas que determinen en qué medida esto ocurre.

En la Figura 2.7 se puede visualizar el mencionado problema. Como se puede apreciar, existen pocas instancias que presenten la etiqueta C, y muchas de ellas también presentan la etiqueta A, que es mayoritaria.

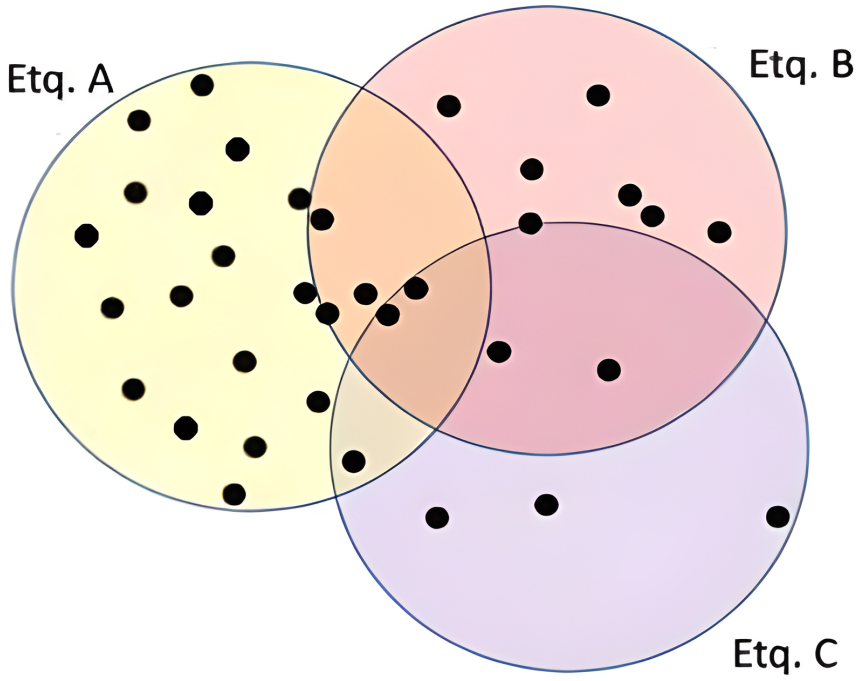

Figura 2.7: Conjunto de datos multietiqueta desbalanceado. Elaboración propia a partir de [1].

Al ser el desbalanceo multietiqueta distinto del desbalanceo tradicional, es de esperar que cuente con sus propias métricas, que intenten representar numéricamente cómo de desbalanceado está un MLD. Algunas de estas métricas se recogen a continuación.

Métricas para medir el desbalanceo en clasificación multietiqueta

Una de las métricas más sencillas para medir el desbalanceo multietiqueta es $IRlbl$ [59], o $IRperLabel$, que mide el IR de cada una de las etiquetas, empleando para ello la fórmula que se muestra en la Ecuación 2.17. En ella, se puede observar que para cada etiqueta y se divide el número de instancias pertenecientes a la etiqueta con más ejemplos entre el número de instancias pertenecientes a y. De este modo, $IRlbl = 1$ para la etiqueta mayoritaria, y crece cuanto menos frecuente es la etiqueta.

$$IRlbl(y) = \frac{\underset{y'=Y_1}{\overset{Y_{|Y|}}{\operatorname{argmax}}} \left(\sum_{i=1}^{|D|} h\left(y', Y_i\right) \right)}{\sum_{i=1}^{|D|} h\left(y, Y_i\right)}, \quad h\left(y, Y_i\right) = \begin{cases} 1 & y \in Y_i \\ 0 & y \notin Y_i \end{cases} \tag{2.17}$$

Por tanto, cuanto mayor sea el valor de $IRlbl$, mayor será el desbalanceo si tenemos en cuenta esa etiqueta, es decir, mayor será la diferencia entre el número de instancias que presentan esa etiqueta y las que no.

A partir de $IRlbl$ se puede obtener una métrica que se refiera al conjunto de datos **al completo**, y no a cada una de sus etiquetas individualmente. Esta métrica es $MeanIR$ [59], y se calcula, como se aprecia en la Ecuación 2.18, como la media de los $IRlbl$ de cada etiqueta.

$$MeanIR = \frac{1}{\mid Y \mid} \sum_{y=Y_1}^{Y_{|Y|}} (IRlbl(y)) \tag{2.18}$$

Al ser una agregación de los valores de $IRlbl$ de cada etiqueta, se puede interpretar el resultado de la misma manera, solo que de forma global. Por tanto, cuanto mayor sea el valor de $MeanIR$, mayor será el desbalanceo en el MLD.

Por otro lado, se puede obtener el coeficiente de variación del $IRlbl$, o $CVIR$ [59]. Este coeficiente es un **estadístico** que se obtiene dividiendo la varianza entre la media. Por tanto, como se observa en la Ecuación 2.19, debemos calcular la varianza del $IRlbl$, o $IRlbl\sigma$, entre $MeanIR$.

$$CVIR = \frac{IRlbl\sigma}{MeanIR}, \quad IRlbl\sigma = \sqrt{\sum_{y=Y_1}^{Y_{|Y|}} \frac{(IRlbl - MeanIR)^2}{\mid Y \mid -1}} \tag{2.19}$$

Con esta medida podemos cuantificar la **dispersión** de los valores de $IRlbl$, es decir, cómo de diferente es el desbalanceo entre etiquetas. Por tanto, cuanto mayor sea este valor más desbalanceado estará el conjunto de datos en cuestión.

Por último, existe la métrica $SCUMBLE$ [60], la cual permite cuantificar la frecuencia con la que se da una situación problemática ya mencionada: aquella en la que las etiquetas minoritarias siempre aparecen acompañadas de una o más etiquetas mayoritarias.

La fórmula para calcular esta métrica, que se muestra en la Ecuación 2.20, está basada en el índice de Atkinson [61], una medida para calcular el grado de desigualdad entre los ingresos de los individuos de una población. En el caso de $SCUMBLE$, cada instancia D_i será una población, cada etiqueta l presente en dicha instancia será un individuo, y los ingresos serán en realidad el $IRlbl$ de cada etiqueta.

$$SCUMBLE = \frac{1}{\mid D \mid} \sum_{i=1}^{|D|} [1 - \frac{1}{\overline{IRlbl_i}} (\prod_{l=1}^{|L|} IRlbl_{il})^{\frac{1}{|L|}}] \tag{2.20}$$

La idea es que un $SCUMBLE$ mayor implica que en el conjunto de datos aparecen con más frecuencia muestras que pertenecen a la vez a una o varias etiquetas con un nivel de desbalanceo muy distinto. Por tanto, nos interesa que el valor de esta métrica sea mínimo para una mejor clasificación.

Una vez se ha explicado el problema de desbalanceo, y se han presentado métricas para cuantificarlo, se puede hacer un repaso de los métodos propuestos en la literatura para tratar dicho problema.

2.5. Métodos para abordar el desbalanceo

Al ser una problemática tan importante en el proceso de minería de datos, existen distintas propuestas que pretenden resolver o, al menos, paliar el efecto negativo del desbalanceo en la clasificación. En la literatura se distinguen **tres enfoques** en los que pueden incluirse las diversas propuestas [62]. Estos enfoques son la modificación de algoritmos, el aprendizaje sensible al coste y el remuestreo de los datos.

2.5.1. Modificación de algoritmos

La modificación de algoritmos consiste en **adaptar el funcionamiento de un modelo**, de modo que pueda hacer frente de una mejor manera al problema del desbalanceo. En función del modelo, se incluirán distintas mejoras.

La mayoría de clasificadores adaptados son basados en redes neuronales o en SVM [63]. De igual modo, también se han propuesto algunas adaptaciones de modelos tradicionales para datos desbalanceados. Por ejemplo, un esquema basado en kNN, llamado SNN [64] que incluye un sistema de pesos para disminuir el efecto del desbalanceo. También se ha propuesto una SVM ondícula, en inglés *wavelet* SVM o WSVM [65], en la cual se lleva a cabo una selección de características para paliar el efecto del desbalanceo en el entrenamiento del modelo. Otra propuesta de SVM combina un modelo de este tipo con probabilidades bayesianas, dando lugar a NBSVM [66] (*Near Bayesian* SVM). Por último, también se han comparado varias redes neuronales con función de base radial (RBFN) con conjuntos de datos desbalanceados [67].

2.5.2. Aprendizaje sensible al coste

Otro enfoque es el aprendizaje sensible al coste [68]. Este se caracteriza por asignar un valor de coste $C(i, j)$ cuando se clasifica de manera errónea la clase i como j. Estos pesos pueden ser dados por expertos o ser aprendidos por el modelo.

Para el caso concreto de conjuntos de datos desbalanceados, es interesante, en lugar de asignar costes a pares de clases, asignarlos a tipos de clases, distinguiendo entre clases mayoritarias y minoritarias, de modo que tengamos los costes $C(+, -)$ y $C(-, +)$.

Así, podemos se puede diciendo que el aprendizaje sensible al coste en el contexto de los conjuntos de datos desbalanceados consiste en **asignar un mayor peso a las clases minoritarias**, forzando así al modelo a prestar más atención a estos ejemplos y, por tanto, a mejorar su rendimiento en las clases menos representadas. En cierto modo se podría decir que se trata de una modificación de los algoritmos, pero al ser un enfoque más general y, por tanto, independiente del algoritmo en el que se aplique, **se considera como una técnica aparte**.

2.5.3. Remuestreo

El remuestreo es el método más genérico frente al desbalanceo en un conjunto de datos. En este caso, en lugar de modificar los modelos de clasificación, **se modifican los datos** antes del proceso de entrenamiento.

En concreto, el remuestreo consiste en **añadir o eliminar instancias** para alterar la distribución del conjunto de datos y así balancearlo, de modo que el sesgo que el desbalanceo introducía en el clasificador deje de existir. Ahora bien, no todo son ventajas, pues por el camino puede **perderse información**, o **introducirse ruido** en el conjunto de datos.

Se pueden distinguir dos tipos principales de remuestreo: el **bajomuestreo** (*undersampling*) y el **sobremuestreo** (*oversampling*). También pueden combinarse ambos tipos para dar lugar a **enfoques híbridos**.

Bajomuestreo

El **bajomuestreo** consiste en balancear el conjunto de datos **eliminando instancias de la clase mayoritaria**. Su principal problema reside en la pérdida de información que tiene lugar al eliminar instancias del conjunto de datos, aunque haciéndolo se esté balanceando.

Existen varias formas de llevar a cabo bajomuestreo, que se diferencian entre sí según la lógica seguida para la **elección de las instancias a eliminar** [69].

El método más básico consiste en seleccionar las instancias de la clase mayoritaria de manera aleatoria. Este método se conoce como **bajomuestreo aleatorio** o *random undersampling* o RUS [70].

Por otro lado, existen métodos que eligen qué instancias mantener en el conjunto de datos.

Se han propuesto técnicas de **bajomuestreo cercano al fallo**, en inglés *near-miss* o NM *undersampling* [71], es decir, que eliminan aquellas instancias que, según algún criterio, confunden más al modelo. El criterio puede variar, pero siempre se tiene en cuenta la distancia entre una instancia de la clase mayoritaria y una o varias instancias de la clase minoritaria. Cuando dicha distancia sea mínima, se eliminará la instancia de la clase mayoritaria.

Otro enfoque es el **bajomuestreo de vecinos condensados**, o *condensed nearest neighbors* (CNN) *undersampling* [72]. Este algoritmo mantiene en un "almacén" (*store*) las instancias pertenecientes a la clase minoritaria y, de manera iterativa, se añaden aquellas instancias de la clase mayoritaria que no son clasificadas de manera correcta teniendo en cuenta las instancias del almacén. Al terminar el proceso, el almacén generado es el nuevo conjunto de datos. De este modo, se han eliminado del conjunto de datos aquellas instancias que son redundantes en el proceso de aprendizaje, y que **introducen sesgo en el clasificador**.

También existen métodos que eligen qué instancias eliminar del conjunto de datos.

El método *Tomek Links* o TL [73] consiste en encontrar pares de instancias que son vecinos más cercanos, tal que una pertenece a la clase mayoritaria y la otra a la minoritaria. Estas parejas en conflicto son conocidas como *tomek links*. Para cada uno de los *tomek links*, se eliminará la instancia de la clase mayoritaria. Este método no es muy útil por sí solo, y suele ser combinado con CNN.

Otro método que sigue este mismo enfoque es la **regla de los vecinos más cercanos editada** (*edited nearest neighbors rule* o ENN) [74]. El funcionamiento de este algoritmo consiste en localizar aquellas instancias mal clasificadas con un modelo kNN con $k = 1$ y eliminar la propia instancia, si pertenecía a la clase mayoritaria, o su vecino, si pertenecía a la clase minoritaria.

Por último, existen combinaciones de los dos enfoques anteriores, en los que se eligen tanto instancias que se eliminan como instancias que se mantienen en el conjunto de datos.

En esta línea, podemos mencionar OSS (*one-sided selection*) [75], que combina TL con CNN. En concreto, primero se aplica CNN sobre el conjunto de datos y, posteriormente, sobre las instancias incluidas en el almacén, se buscan *tomek links* y se eliminan las instancias de los mismos que pertenecen a la clase mayoritaria.

Para terminar, también siguiendo el enfoque combinatorio, se encuentra la **regla de limpieza de vecinos** o *neighborhood cleaning rule* (NCR) [76], que combina CNN y ENN, aplicando primero CNN y, posteriormente, ENN sobre el almacén generado, al igual que el método anterior.

Todos estos métodos tratan de **reducir la pérdida de información introducida por la eliminación de instancias**, intentando que las instancias eliminadas sean redundantes, teniendo en cuenta sus vecinos. Otra manera de hacer esto es usar *clustering* sobre la clase mayoritaria [77], ajustando el número de grupos al número de

instancias de la clase minoritaria, y manteniendo los medioides de los grupos generados. De este modo, las instancias de la clase mayoritaria que se mantienen en el conjunto de datos son las más representativas, y, por tanto, la pérdida de información es menor.

Sobremuestreo

El **sobremuestreo** consiste en alterar la distribución del conjunto de datos añadiendo instancias de la clase minoritaria al mismo. El resultado es el mismo que en el bajomuestreo: un conjunto de datos más balanceado, solo que en este caso, sin necesidad de eliminar instancias, por lo que no se pierde información.

No obstante, no todo son ventajas, pues el sobremuestreo puede traer consigo inconvenientes, como la **introducción de ruido** en el conjunto de datos, al añadir instancias al mismo.

Al igual que con el bajomuestreo, uno de los algoritmos más básicos es el **sobremuestreo aleatorio** (*random oversampling* o ROS) [78], que consiste en, de manera aleatoria, elegir instancias de la clase mayoritaria que se duplicarán, pudiendo una misma instancia aparecer varias veces en el conjunto de datos. Al hecho de que se dupliquen instancias se le conoce como **sobremuestreo con remplazamiento**.

Estas técnicas, en efecto, disminuyen el desbalanceo en el conjunto de datos, pero lo hacen a expensas de introducir información redundante en el mismo. Por esta razón, surgen algoritmos capaces de generar instancias **sintéticas**, con valores distintos a los del conjunto de datos original, pero que siguen la distribución de los mismos. Uno de los más conocidos es SMOTE [79]. Este algoritmo genera nuevas instancias usando los valores de los atributos de instancias vecinas.

Tanto SMOTE como otras técnicas de sobremuestreo se recogen y explican en el apartado 2.6.

Enfoques híbridos

Debido a las ventajas e inconvenientes que pueden tener el sobremuestreo y el bajomuestreo para problemas determinados, varios autores han propuesto algoritmos que combinan ambos métodos para solucionar un problema concreto.

Por ejemplo, existe un algoritmo que combina RUS y SMOTE, para la alerta temprana de riesgos extremos en los mercados financieros [80]. También se han propuesto varias combinaciones entre CNN y SMOTE [81].

Por otro lado, existe un algoritmo [82] que lleva a cabo bajomuestreo mediante *clustering* sobre la clase mayoritaria con un *Gaussian Mixture Model* (modelo generativo que se introduce en el apartado 2.7), para posteriormente realizar sobremuestreo sobre la clase minoritaria con SMOTE.

Siguiendo el enfoque anterior, merece la pena destacar una interesante idea [83], que propone llevar a cabo *clustering* sobre la clase mayoritaria para, posteriormente, generar una instancia sintética con SMOTE que represente el medioide perfecto de cada grupo. De este modo, cada grupo queda representado por una instancia sintética que minimiza la pérdida de información asociadada al *clustering*, pues el medioide representa de la manera más fiel posible a todos los miembros del grupo.

Una vez hemos repasado los tipos de remuestreo, debemos escoger uno para el algoritmo a proponer en el trabajo. Para ello, me remito al estado del arte [84], donde, tras un extensivo estudio, ha sido demostrado que el sobremuestreo ofrece, en general, mejores resultados que el bajomuestreo. Por ello, en el siguiente apartado se profundiza en distintas técnicas de sobremuestreo, tanto para conjuntos de datos tradicionales como para MLD.

2.6. Técnicas para sobremuestreo

Debido a que, en este trabajo, se va a proponer un algoritmo de sobremuestreo, este apartado se dedica exclusivamente a revisar otros algoritmos ya propuestos con el mismo fin, tanto para clasificación tradicional como, de manera más específica, para MLC.

2.6.1. Sobremuestreo en conjuntos de datos tradicionales

Son muchas las técnicas propuestas para llevar a cabo sobremuestreo en conjuntos de datos binarios y multiclase. En este apartado se revisarán algunas de ellas.

SMOTE

Este algoritmo [79] es, probablemente, el más famoso entre las técnicas de sobremuestreo tradicional. Se han llevado a cabo múltiples variaciones sobre el mismo que mejoran ciertos aspectos, algunas de las cuales serán revisadas en este apartado.

El algoritmo SMOTE (*Synthetic Minority Oversampling Technique*) básico propone un sobremuestreo en el que no se usan muestras duplicadas (como es el caso de ROS), sino que se crean instancias sintéticas que pertenecen a la clase minoritaria.

El punto clave en la generación de instancias sintéticas es el valor de los atributos. La clase va a ser la minoritaria, pero se deben asignar a los atributos de la muestra unos valores que mantengan la distribución del conjunto de datos original.

La primera versión del algoritmo propone calcular los valores de los atributos de las instancias sintéticas a partir de los valores de las **instancias vecinas**, siendo el número de vecinos un parámetro a ajustar. En esta primera versión, el cálculo de vecinos se lleva a cabo teniendo en cuenta la **distancia euclídea** [85]. El pseudocódigo de SMOTE se recoge en el Algoritmo 1.

Algoritmo 1: Pseudocódigo del algoritmo SMOTE. Fuente: [79]

Input: <Dataset> D, <Porcentaje> P, <Entero> k
Output: Dataset preprocesado
if $P < 100\,\%$ **then**
    $aleatorizarInstanciasMinoritarias(D, P)$
    $T \leftarrow P/100 \cdot N$
    $N \leftarrow 100$
end
$N \leftarrow aproximar(N/100)$; ▷Se consideran múltiplos de 100
for $i \leftarrow 1 \rightarrow T$ **do**
    $vecinos \leftarrow calcularKVecinosMasCercanos(D, k)$
    $D \leftarrow D + generarInstancias(N, i, vecinos)$
end

A pesar de ofrecer buenos resultados, SMOTE presenta algunos problemas. Para empezar, como se aprecia en el Algoritmo 2, la función que genera las instancias sinteticas **no considera atributos categóricos**, sino que únicamente funciona con atributos numéricos. Otro problema es que no tiene en cuenta que no todas las instancias minoritarias son igual de difíciles de clasificar, pues aquellas cercanas al límite de decisión son mal clasificadas con más frecuencia. También se ha achacado a SMOTE la **sensibilidad al ruido**, o el hecho de ignorar los vecinos mayoritarios de las instancias minoritarias.

Algoritmo 2: Generación de instancias sintéticas en SMOTE. Fuente: [79]

```
1:  function GenerarInstancias(N, i, vecinos)
2:      instanciasSinteticas ← listaVacia() while N ≠ 0 do
3:          nuevaInstancia ← newInstancia ;                ▷Nueva instancia vacía
4:          nn ← vecinoAleatorio(vecinos)
5:          for each atributo in D do
6:              dif ← D[nn].atributo − D[i].atributo
7:              gap ← numeroAleatorio(0, 1)
8:              nuevaInstancia.atributo ← D[i].atributo + gap · dif
                instanciasSinteticas ← instanciasSinteticas + nuevaInstancia
9:          end
10:         N ← N − 1
11:     end
12:     return instanciasSinteticas
```

Es debido a estas debilidades que, tras su publicación, muchas otras variantes de SMOTE fueron propuestas para solucionar dichos problemas.

Variaciones de SMOTE

Como se ha indicado, las técnicas basadas en SMOTE constituyen gran parte de los algoritmos de sobremuestreo. Algunas de ellas se mencionan en este apartado.

En el mismo artículo en el que se presentó SMOTE [79], también se propusieron extensiones **capaces de trabajar con atributos categóricos**. En concreto, SMOTE-NC (SMOTE *Nominal Continuous*), usa la mediana de los valores nominales para el cálculo de distancias y, por tanto, para la elección de los vecinos. Por otro lado, SMOTE-N, en lugar de usar la mediana de los atributos categóricos, usa una medida de distancia distinta de la euclídea para los atributos nominales en el cálculo de los vecinos. Esta medida es VDM (*Value difference Metric*) [86].

Otra mejora sobre SMOTE es Borderline-SMOTE [87]. La idea expuesta por los autores es que SMOTE crea instancias sintéticas basándose en todas las instancias minoritarias del conjunto de datos. De este modo, aquellas instancias que están en el límite (*borderline*) de decisión, es decir, las que son más frecuentemente mal clasificadas, y que determinan el buen funcionamiento del clasificador, **no son representadas en el sobremuestreo**. Por tanto, Borderline-SMOTE propone encontrar esas instancias "peligrosas" mediante vecinos más cercanos (buscando aquellas instancias minoritarias cuyos vecinos son, en su mayoría, instancias mayoritarias) y realizar SMOTE sobre ellas. Una primera variante genera las instancias sintéticas usando los valores de los vecinos minoritarios, y una segunda tiene también en cuenta a los vecinos mayoritarios.

Por otro lado está Safe-level-SMOTE [88], que recoge la estela de la segunda versión de Borderline-SMOTE y, al igual que dicha versión, también tiene en cuenta las instancias de la clase mayoritaria que son vecinas de la instancia a la que se está aplicando SMOTE. La diferencia es que, en este caso, se controla la influencia de dichas instancias en el cálculo de los atributos sintéticos mediante pesos, de modo que se mantiene un "nivel seguro".

Otros ejemplos son DBSMOTE [89], que combina SMOTE con agrupamiento, o SMOTE-IPF [90], que lleva a cabo un filtrado de los valores de los atributos, lo cual permite reducir el negativo impacto del ruido en el sobremuestreo.

Técnica de sobremuestreo con rechazo

Otro problema de SMOTE es la posibilidad de que las instancias sintéticas generadas sean **ruido** [91]. Esto ocurre debido a que SMOTE es sensible al número de vecinos elegidos para cada instancia, lo cual hace posible situaciones como la que se observa en la Figura 2.8. En ella, se representan las instancias mayortarias con círculos y las minoritarias con triángulos. Como se aprecia en la figura, puede ocurrir que, entre los vecinos minoritarios más cercanos de algunas instancias minoritarias (por ejemplo. B y E), se encuentren instancias demasiado lejanas, como A y D. De este modo, las instancias sintéticas generadas (C y F) no son más que ruido.

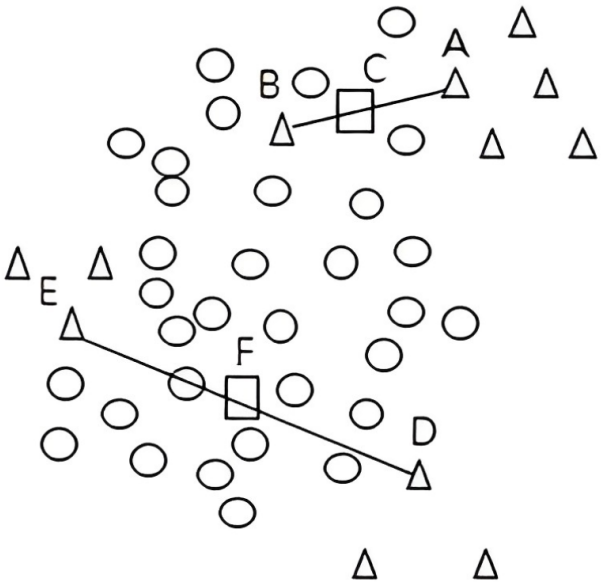

Figura 2.8: Posible situación indeseable en SMOTE. Fuente: [91].

Teniendo en cuenta este posible inconveniente, se propone un algoritmo, en cierto modo también derivado de SMOTE, en el que la generación de datos sintéticos se

lleva a cabo **teniendo en cuenta su localización**. De este modo, si una instancia generada se considera ruido, esta será rechazada. Para ello, a cada instancia sintética se le asigna un nivel de rechazo, en función de la proporción de vecinos de la misma que pertenecen a la clase mayoritaria. Si el nivel de rechazo es superior a un umbral establecido, entonces la instancia queda rechazada.

Combinar limpieza con remuestreo

En realidad, el desbalanceo, entendido exclusivamente como la excesiva diferencia entre el número de instancias pertenecientes a la clase mayoritaria y minoritaria, no es el motivo del mal funcionamiento de los clasificadores. Por ejemplo, puede existir un conjunto de datos desbalanceado, pero en el que las instancias pertenecientes a las diferentes clases sean **linealmente separables**. En este caso, aunque exista desbalanceo, el clasificador funcionará bien. Por tanto, se puede concluir que son **otros aspectos**, como datos con ruido, distribuciones complicadas o un número de instancias demasiado bajo, los que propician un mal rendimiento en la clasificación.

Es por este motivo que se propone el algoritmo CCR [92] (*combined cleaning and resampling*), cuyo objetivo es **limpiar el conjunto de datos** para que el proceso de generación de instancias sintéticas se lleve a cabo en zonas menos seguras, forzando a los clasificadores a prestar más atención a ejemplos más difíciles de aprender.

En el Algoritmo 3 se recoge el pseudocódigo de CCR. Como se puede apreciar, se debe determinar un parámetro energía, que regulará el radio de expansión de la esfera generada alrededor de cada instancia minoritaria. La energía de cada instancia dependerá del parámetro inicial, así como del número de instancias pertenecientes a la clase mayoritaria que hay en los alrededores de la misma.

En la Figura 2.9 se visualiza el funcionamiento del algoritmo. Partiendo de un conjunto de datos compuesto por instancias de una clase minoritaria (cuadrados) y de una mayoritaria (triángulos), lo primero que se hace es crear esferas alrededor de los cuadrados con un tamaño proporcional al número de triángulos que los rodean. Después, se desplazan los triángulos que quedan dentro de alguna esfera hacia los bordes de la misma. Por último, se seleccionan puntos aleatorios dentro de la esfera para combinar con la instancia central, para generar instancias sintéticas.

La idea es que se generen más instancias sintéticas en las esferas con radios más pequeños, de modo que las clases minoritarias queden mejor representadas en las zonas inseguras, y así los clasificadores presten más atención a estos casos.

Algoritmo 3: Pseudocódigo del algoritmo CCR. Fuente: [92]

Input: <Dataset> D, <Entero> $energia$
Output: Dataset preprocesado

for ***each*** *instancia minoritaria* m_i ***in*** D **do**
  $e_i \leftarrow energia$; ▷Energía restante
  $r_i \leftarrow 0$
  while $e_i > 0$ **do**
    $\Delta r \leftarrow \frac{e_i}{numeroMayoritarias(m_i, r_i)}$
    if $numeroMayoritarias(m_i, r_i + \Delta r) > numeroMayoritarias(m_i, r_i)$ **then**
      $\Delta r \leftarrow$ distancia a la instancia mayoritaria más cercana que no esté dentro de r_i
    end
    $r_i \leftarrow r_i + \Delta r$
    $e_i \leftarrow e_i - \Delta r \cdot numeroMayoritarias(m_i, r_i)$
  end
  for ***each*** *instancia mayoritaria* M_j ***dentro de*** r_i *de* m_i **do**
    $d \leftarrow ||M_j - m_j||_1$
    $t_j \leftarrow t_j + \frac{r_i - d}{d} \cdot (M_j - m_i)$; ▷Traslación de M_j
  end
end
$aplicarTraslacionesAcumuladas(M)$
$G \leftarrow |M| - |m|$; ▷Número de muestras sintéticas a generar
for ***each*** *instancia minoritaria* m_i ***in*** D **do**
  $g_i \leftarrow \frac{{r_i}^{-1}}{\sum {r_k}^{-1}} \cdot G$; ▷Proporción de G para m_i
  for $n = 1 \rightarrow g_i$ **do**
    $p \leftarrow$ punto aleatorio dentro de una esfera con radio r_i
    $generarPuntoSintetico(m_i, p)$
  end
end

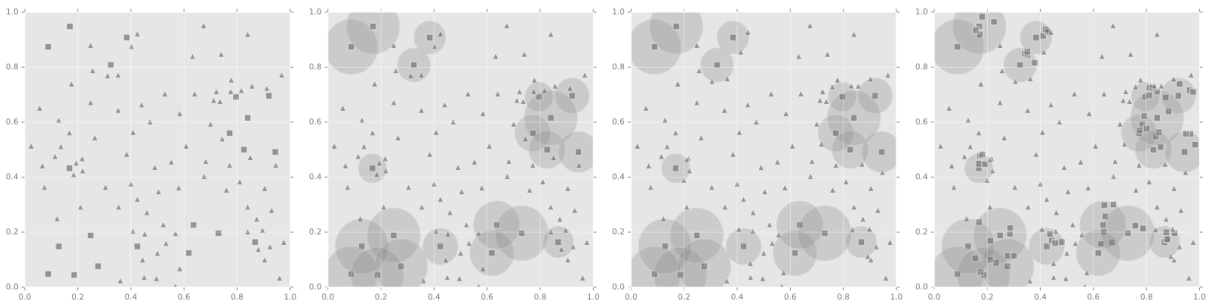

Figura 2.9: Funcionamiento del algoritmo CCR. Fuente: [92].

Estas son solo algunas de las muchas técnicas de sobremuestreo disponibles en la literatura. En [93] se puede encontrar una revisión y estudio experimental de este tipo de técnicas.

2.6.2. Técnicas para sobremuestreo en conjuntos de datos multietiqueta

Ya se han repasado algunas técnicas de sobremuestreo en conjuntos de datos tradicionales. No obstante, el interés de este trabajo se centra en el aprendizaje multi-etiqueta.

Es por eso que, a continuación, se presentan algunos de los algoritmos de sobremuestreo diseñados para MLD [94]. Estos y otros algoritmos se han implementado en un **paquete de R** llamado `mldr.resampling` [95], que he desarrollado en paralelo con la escritura de este trabajo, y que servirá para poner en contexto el rendimiento del algoritmo propuesto en el mismo.

En la Tabla 2.1 se recogen los algoritmos que se van a explicar en este apartado, incluyendo una breve descripción, antes de ahondar más en su funcionamiento.

Algoritmo	Descripción	Referencia
`LPROS`	Sobremuestreo aleatorio de *labelsets*	[96]
`MLROS`	Sobremuestreo aleatorio de una etiqueta minoritaria	[96]
`MLRkNNOS`	Sobremuestreo basado en vecinos más cercanos reversos	[97]
`MLSMOTE`	Sobremuestreo sintético basado en similitud de vecinos	[98]
`MLSOL`	Sobremuestreo basado en desbalanceo local	[99]
`REMEDIAL`	Desacoplamiento de etiquetas minoritarias y mayoritarias	[100]
`RHwRSMT`	Aplicación de REMEDIAL y SMOTE	[101]

Tabla. 2.1: Algoritmos de sobremuestreo multietiqueta del estado del arte

LPROS

El algoritmo LPROS [96] **detecta conjuntos de etiquetas minoritarios y duplica instancias**, hasta alcanzar un porcentaje determinado de los mismos para reducir el desbalanceo.

La ventaja de este algoritmo es su sencillez, pero tiene un claro inconveniente, y es que es posible que un conjunto de etiquetas minoritario surja de la combinación de etiquetas minoritarias con etiquetas mayoritarias. Por tanto, al duplicar instancias con dichos *labelsets* se está aumentando la frecuencia de etiquetas minoritarias, lo cual ayuda a paliar el desbalanceo, pero también la de etiquetas mayoritarias. Podemos decir, por tanto, que la reducción del desbalanceo está limitada por el grado de $SCUMBLE$ del conjunto de datos.

El pseudocódigo de LPROS se recoge en el Algoritmo 4.

Algoritmo 4: Pseudocódigo del algoritmo LPROS. Fuente: [96]

Input: <Dataset> D, <Porcentaje> P
Output: MLD preprocesado
$muestrasAGenerar \leftarrow |D|/100 \cdot P$; ▷Incremento de un P%
▷Agrupar muestras según su labelset
for $i = 1 \rightarrow |labelsets|$ **do**
    $labelSetBag_i \leftarrow muestrasConLabelset(i)$
end
▷Calcular el número medio de muestras por labelset
$tamMedio \leftarrow 1/|labelsets| \cdot \sum_{i=1}^{|labelsets|} |labelSetBag_i|$
▷Obtener muestras con labelsets minoritarios
for ***each*** $labelSetBag_i$ ***in*** $labelSetBag$ **do**
    if $|labelSetBag_i| > tamMedio$ **then**
        $minBag_i \leftarrow labelSetBag_i$
    end
end
$incMedio \leftarrow muestrasAGenerar/|minBag|$
$minBag \leftarrow OrdenarDeMayorAMenor(minBag)$
▷Calcular # de instancias a agregar y añadirlas
for ***each*** $minBag_i$ ***in*** $minBag$ **do**
    $rBag_i \leftarrow min(tamMedio - |minBag_i|, incMedio)$
    $resto \leftarrow incMedio - rBag_i$
    $distribuirEntreBags_{j>i}(resto)$ **for** $n = 1 \rightarrow rBag_i$ **do**
        $x \leftarrow random(1, |minBag_i|)$
        $clonarMuestra(minBag_i, x)$
    end
end

MLROS

El algoritmo MLROS [96] tiene un enfoque similar a LPROS, solo que, en su caso, **lo que se detectan son las etiquetas minoritarias** (no los *labelsets*). Se obtienen las instancias que presentan dichas etiquetas minoritarias, y se duplican de manera aleatoria hasta alcanzar el porcentaje deseado.

De nuevo, el problema es que una instancia que tiene una etiqueta minoritaria puede tener a su vez una o varias etiquetas mayoritarias. Por tanto, al duplicarla, la reducción del desbalanceo no es tan efectiva como podríamos esperar.

El pseudocódigo de MLROS se recoge en el Algoritmo 5.

MLRkNNOS

Existe también un algoritmo basado en vecinos más cercanos “reversos”, en inglés *reverse nearest neighbors* o RkNN [97], un concepto que hace referencia a aquellas instancias que tienen a una instancia determinada como vecina. Puesto que sus auto-

Algoritmo 5: Pseudocódigo del algoritmo MLROS. Fuente: [96]

Input: <Dataset> D, <Porcentaje> P
Output: MLD preprocesado

$muestrasAGenerar \leftarrow |D|/100 \cdot P$; ▷Incremento de un P%
$L \leftarrow etiquetasEnDataset(D)$; ▷Conjunto completo de etiquetas
$MeanIR \leftarrow calcularMeanIR(D, L)$
for ***each*** $etiqueta$ ***in*** L **do**
    $IRLbl_{etiqueta} \leftarrow calcularIRporEtiqueta(D, etiqueta)$
    if $IRLbl_{etiqueta} > MeanIR$ **then**
        $minBag_{i++} \leftarrow Bag_{etiqueta}$
    end
end
while $muestrasAGenerar > 0$ **do**
    ▷Clonar una muestra aleatoria de cada paquete minoritario
    for ***each*** $minBag_i$ ***in*** $minBag$ **do**
        $x \leftarrow random(1, |minBag_i|)$
        $clonarMuestra(|minBag_i|, x)$
        if $IRLbl_{minBag_i} <= MeanIR$ **then**
            $minBag \rightarrow minBag_i$; ▷Excluir del proceso de clonado
        end
        $--muestrasAGenerar$
    end
end

res no le otorgaron un nombre, en esta memoria será llamado MLRkNNOS (*multilabel reverse k nearest neighbors oversampling*).

Hasta ahora, para cada instancia el enfoque había residido en sus vecinos. No obstante, en este algoritmo, para cada instancia se tendrán en cuenta aquellas que la tienen como uno de sus vecinos más cercanos.

La idea es que, mediante el método kNN, el vecindario de una instancia se ve restringido a un número fijo, de modo que no se están reflejando aspectos como la densidad o distribución del MLD en esas zonas. Por su parte, el enfoque RkNN **permite que el número de vecinos sea variable**, representando mejor el vecindario de las instancias.

En el Algoritmo 6 se recoge el pseudocódigo del mencionado algoritmo. Como se puede observar, se aplica un enfoque BR, pues se lleva a cabo un proceso de sobremuestreo para cada etiqueta de manera independiente. Este proceso consiste en generar instancias sintéticas hasta igualar el número de instancias mayoritarias y minoritarias respecto a una etiqueta. Dicho de otra forma, el objetivo del algoritmo es que, para cada etiqueta, haya el mismo número de instancias en el MLD que la tengan presente y que no (normalmente, la clase mayoritaria en una etiqueta suele ser 0 o no presente, y la minoritaria, 1 o presente).

Los atributos de las instancias sintéticas se generarán combinando instancias minoritarias con alguno de sus vecinos reversos, y la clase de la etiqueta que estamos balanceando, será la minoritaria. No obstante, el resto de etiquetas se predicen, más que calcularse, pues para cada etiqueta se entrena un modelo SVM capaz de predecir la clase de la misma. Así, la clase de las etiquetas desconocidas de cada instancia sintética generada será predicha por el correspondiente modelo.

Este enfoque puede ser preciso pero, en MLD con una alta dimensionalidad, el entrenamiento de los modelos sería **muy poco eficiente**.

Algoritmo 6: Pseudocódigo del algoritmo basado en RkNN. Fuente: Elaboración propia a partir de [97]

Input: <Dataset> D, <Entero> k
Output: MLD preprocesado
for ***each*** *etiqueta* l ***in*** L **do**
    $Mayoritaria_l \leftarrow instanciasCon(l = claseMayoritaria)$
    $Minoritaria_l \leftarrow instanciasCon(l = claseMinoritaria)$
    ▷`Diferencia entre instancias con clase mayoritaria y minoritaria`
    $N_l \leftarrow |Mayoritaria_l| - |Minoritaria_l|$
    for ***each*** *instancia* i ***in*** $Minoritaria_l$ **do**
        $vecinos_i \leftarrow RkNN(Minoritaria_l, k)$; ▷`Solo vecinos minoritarios`
        $MinoriaAumentada \leftarrow vecinos_i \; con \; |vecinos_i| > 1$
    end
    ▷`Generación de atributos de instancias sintéticas`
    $S_l \leftarrow \emptyset$
    repeat N_l **times**
        $q \leftarrow aleatoria(MinoriaAumentada)$
        $r \leftarrow aleatoria(vecinos_q) \; con \; r \neq q$
        $atributosSinteticos \leftarrow q + (q - r) * random(0, 1) * distanciaEuclidea(q, r)$
        ▷`Valor para la etiqueta` l `minoritario, resto sin determinar`
        $instanciaSintetica \leftarrow atributosSinteticos \; con \; l = valorMinoritario$
        $S_l \leftarrow S_l + instanciaSintetica$
    end
end
▷`Entrenamiento de modelos SVM para cada etiqueta`
for ***each*** *etiqueta* l ***in*** L **do**
    $O_l \leftarrow Mayoritaria_l + Minoritaria_l + S_l$
    $M_l \leftarrow entrenarSVM(O_l)$
end
▷`Predicción de etiquetas con clasificadores`
for ***each*** *instancia* i ***in*** S **do**
    $Predecir \; etiquetas \; desconocidas \; de \; i \; con \; SVM \; de \; cada \; etiqueta$
end

MLSMOTE

El algoritmo MLSMOTE [98] es una adaptación para datos multietiqueta del algoritmo SMOTE [79], que fue diseñado para clasificación tradicional. En ambos casos, el objetivo es **añadir instancias sintéticas al conjunto de datos original**, para reducir

el desbalanceo. Para ello, dichas instancias deberán pertenecer a la clase minoritaria (o, en el caso de MLSMOTE, a etiquetas minoritarias).

El proceso consiste en obtener, para cada una de las etiquetas minoritarias, las muestras pertenecientes a la misma y, para cada una de ellas, encontrar las k muestras más cercanas (siendo k un parámetro a definir). Una vez se han calculado los vecinos, se generará una nueva instancia agregando los valores de cada uno de sus atributos y etiquetas, y esta se añadirá al conjunto de datos original.

El pseudocódigo de MLSMOTE se recoge en el Algoritmo 7.

Algoritmo 7: Pseudocódigo del algoritmo MLSMOTE. Fuente: [98]

Input: <Dataset> D, <Entero> k
Output: MLD preprocesado

$L \leftarrow etiquetasEnDataset(D)$; ▷`Conjunto completo de etiquetas`
$MeanIR \leftarrow calcularMeanIR(D)$
for ***each*** $etiqueta$ ***in*** L **do**
    $IRLbl_{etiqueta} \leftarrow calcularIRporEtiqueta(D, etiqueta)$
    if $IRLbl_{etiqueta} > MeanIR$ **then**
        ▷`Paquete de instancias con etiqueta minoritaria`
        $minBag \leftarrow instanciasConEtiqueta(etiqueta)$
        for ***each*** $instancia$ ***in*** $minBag$ **do**
            $distancias \leftarrow calcularDistancias(instancia, minBag)$
            $ordenarDeMenorAMayor(distancias)$
            ▷`Selección del conjunto de vecinos cercanos`
            $vecinos \leftarrow obtenerPrimerosElementos(distancias, k)$
            $vecinoRef \leftarrow obtenerVecinoAleatorio(vecinos)$
            ▷`Generación del conjunto de atributos y etiquetas`
            $instanciaSintetica \leftarrow nuevaInstancia(instancia, vecinoRef, vecinos)$
            $D \leftarrow D + instanciaSintetica$
        end
    end
end

El cálculo de las distancias a los vecinos puede llevarse a cabo con distintas métricas. En la implementación original, para atributos numéricos se usaba la distancia euclídea [85], y entre atributos categóricos se recurría a la VDM (*Value difference Metric*) [86]. El algoritmo 8 muestra la función encargada de producir nuevas instancias sintéticas.

MLSOL

Los algoritmos mencionados anteriormente se centraban en el desbalanceo global del conjunto de datos, aunque medido con distintas métricas.

Algoritmo 8: Generación de instancias sintéticas en MLSMOTE. Fuente: [98]

function NuevaInstancia($instancia, vecinoRef, vecinos$)
  $instanciaSintetica \leftarrow$ **new** $Instancia$; ▷Nueva instancia vacía
  ▷Asignación del conjunto de atributos
  for ***each*** $atributo$ ***in*** $instanciaSintetica$ **do**
    if $tipo(atributo)$ es $numerico$ **then**
      $diferencia \leftarrow vecinoRef.atributo - instancia.atributo$
      $offset \leftarrow diferencia \cdot aleatorioEntre(0,1)$
      $valor \leftarrow instancia.atributo + offset$
    else
      $valor \leftarrow valorMasFrecuente(vecinos, atributo)$
    end
    $instanciaSintetica.atributo \leftarrow valor$
  end
  ▷Asignación del conjunto de etiquetas
  $contadorEtiquetas \leftarrow contar(instancia.etiquetas)$
  $contadorEtiquetas+ \leftarrow contar(vecinos.etiquetas)$
  $etiquetas \leftarrow contadorEtiquetas > (k+1)/2$
  $instanciaSintetica.etiquetas \leftarrow etiquetas$
  return $instanciaSintetica$

No obstante, se ha planteado la idea [99] de que en MLC, al igual que en clasificación binaria y multiclase [102], la principal razón por la que un clasificador tiene dificultades para reconocer las etiquetas minoritarias no es el nivel de desbalanceo global, sino el **vecindario local de las instancias minoritarias**.

Este hecho se ilustra en la Figura 2.10. En ella, se observa un MLD con tres etiquetas, representadas por la forma (triángulo o círculo), el color (verde o rojo), y el borde (con o sin él). Tanto en (a) como en (b), los MLD, el nivel de desbalanceo es el mismo. No obstante, (b) aparenta ser más difícil de clasificar, debido a la presencia de "subconceptos" (por ejemplo, en el caso de los triángulos, estos no están todos juntos en la misma zona), así como el solapamiento que se da entre círculos rojos y verdes, con y sin borde.

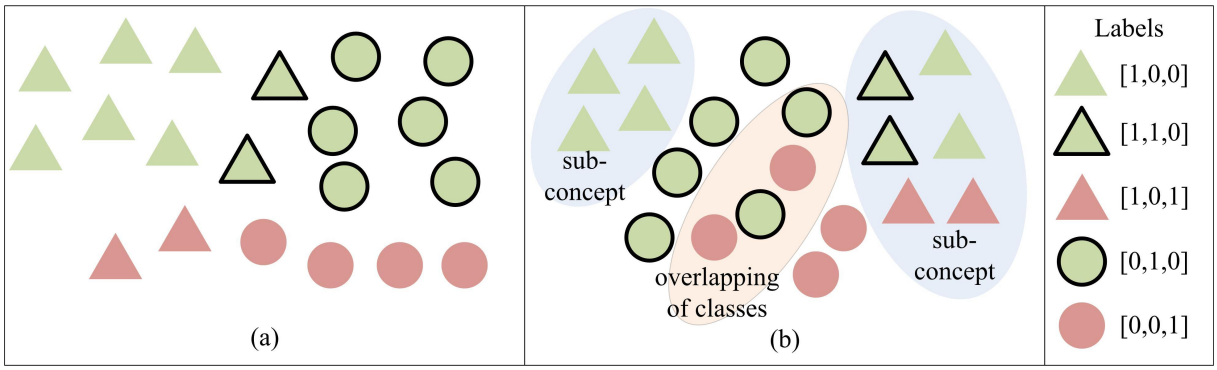

Figura 2.10: Ejemplo de dos MLD con mismo nivel de desbalanceo. Fuente: [99].

Frente a este problema, se propone MLSOL [99], un algoritmo en el que la selección de instancias y la posterior generación de instancias sintéticas se llevan a cabo teniendo en cuenta las **distribuciones locales**, y no el nivel de desbalanceo global.

En el Algoritmo 9 se recoge el pseudocódigo de MLSOL. Como se puede apreciar, lo primero que hace es calcular diversas variables auxiliares. En concreto, las que se usarán en la selección y generación de instancias son, respectivamente, el vector de pesos w y la matriz de tipos de instancias y etiquetas T.

Para ello, primero se deben calcular los vecinos de cada instancia, para así obtener la matriz C, en la que se almacenan, para cada una de las etiquetas de cada instancia, la proporción de vecinos que tienen el valor contrario para dicha etiqueta. De este modo, valores próximos a 1 indican entornos hostiles, y los cercanos a 0 indican que hay poca variabilidad (un vecindario "tranquilo"). De hecho, si el valor es igual a 1, podemos considerar que la instancia en cuestión es un *outlier*.

El siguiente paso es asignar un peso w a cada instancia, que será un factor en la selección. Para ello, se agregan los valores de C, teniendo en cuenta la condición de que $C < 1$, para evitar *outliers*.

Tras ello, se obtiene una matriz de tipos T, en la que para cada par instancia-etiqueta, se asigna uno de cuatro tipos, en función del valor de C, los cuales son SF (*safe*, segura), BD (*borderline*, próxima a la frontera de decisión), RR (*rare*, rara, muy difícil de clasificar), y OT (*outlier*, cuando $C = 1$).

De este modo, MLSOL será capaz de generar instancias sintéticas **más diversas**, poblando zonas más complicadas, en lugar de hacerlo de manera aleatoria.

REMEDIAL

Hasta ahora, los algoritmos que se han visto toman como indicador del desbalanceo la métrica $MeanIR$. No obstante, existen otras métricas, como vimos en el apartado 2.4.2. Una de ellas es $SCUMBLE$, la cual se centra en los casos en los que existe concurrencia entre etiquetas mayoritarias y minoritarias.

Para tener en cuenta el $SCUMBLE$ al llevar a cabo el sobremuestreo, se propone REMEDIAL [100], un algoritmo que realiza el remuestreo desacoplando etiquetas desbalanceadas. La idea es que, al aparecer juntas, las etiquetas mayoritarias están "enmascarando" a las minoritarias, por lo que se introduce un sesgo en el clasificador.

Algoritmo 9: Pseudocódigo del algoritmo MLSOL. Fuente: [99]

Input: <Dataset> D, <Porcentaje> P, <Entero> k
Output: MLD preprocesado

$GenNum \leftarrow |D| \cdot P$; ▷Número de instancias a generar
$D' \leftarrow D$
for ***each*** *instancia* i ***in*** D **do**
    $vecinos_i \leftarrow obtenerVecinos(k)$
end
for ***each*** *instancia* i ***in*** D **do**
    for ***each*** *etiqueta* j ***in*** L **do**
        $C_{ij} = \frac{1}{k} \sum_{x_m \in vecinos_i} [[y_{mj} \neq y_{ij}]]$
    end
end
for ***each*** *instancia* i ***in*** D **do**
    $w_i = \sum_{j=1}^{q} C_{ij} \frac{[[y_{ij}=1]][[C_{ij}<1]]}{\sum_{i=1}^{n} C_{ij}[[y_{ij}=1]][[C_{ij}<1]]}$
end
$T \leftarrow iniciarTipos(C, k)$; ▷Tipos de instancias
while $GenNum > 0$ **do**
    $semilla \leftarrow seleccionarSegunW(D, w)$
    $referencia \leftarrow seleccionarAleatoria(vecinos_{semilla})$
    $sintetica \leftarrow generarInstancia(semilla, T_{semilla}, referencia, T_{referencia})$
    $D \leftarrow D' + sintetica$
    $GenNum \leftarrow GenNum - 1$
end

Por ello, si se desacoplaran las etiquetas, para que aparecieran por separado, este sesgo desaparecería.

En el Algoritmo 10 se recoge el pseudocódigo del algoritmo REMEDIAL. Como se puede apreciar, lo primero que se hace es calcular el IR de cada etiqueta, y el valor medio del MLD, así como el $SCUMBLE$ de cada instancia y el promedio para el MLD. Después, para aquellas instancias con un $SCUMBLE$ superior a la media, se generan dos nuevas instancias: una en la que se mantienen solo las etquetas mayoritarias y se desactivan las minoritarias, y otra en la que solo se mantienen las minoritarias. Es por el hecho de generarse dos nuevas instancias por cada una con alto $SCUMBLE$ que se considera un algoritmo de sobremuestreo.

RHwRSMT

Como se ha explicado, REMEDIAL tan solo desacopla el conjunto de etiquetas, obteniendo un par de instancias, pero realmente no se generan nuevas instancias sintéticas, sino que las dos proceden de una base.

Por ello, se propone RHwRSMT [101]: un híbrido que combina REMEDIAL y SMOTE, llevándose a cabo el desacoplamiento primero, y la generación de instancias sinté-

Algoritmo 10: Pseudocódigo del algoritmo REMEDIAL. Fuente: [100]

Input: <Dataset> D
Output: MLD preprocesado
for ***each*** $etiqueta$ ***in*** L **do**
    $IRLbl_{etiqueta} \leftarrow calcularIRporEtiqueta(D, i)$
end
$MeanIR \leftarrow calcularMeanIR(D)$
for ***each*** *instancia* i ***in*** D **do**
    $SCUMBLE_i \leftarrow calcularSCUMBLE(D, i)$
end
$meanSCUMBLE \leftarrow calcularMeanSCUMBLE(D)$
for ***each*** *instancia* i ***in*** D **do**
    if $SCUMBLE_{etiqueta} > meanSCUMBLE$ **then**
        $D_i' \leftarrow D_i$; ▷Clonar la instancia afectada
        $D_i[etiquetas_{IRLbl \leq MeanIR}] \leftarrow 0$; ▷Mantener etiquetas mayoritarias
        $D_i'[etiquetas_{IRLbl > MeanIR}] \leftarrow 0$; ▷Mantener etiquetas minoritarias
        $D \leftarrow D + D_i'$
    end
end

ticas después. De este modo, las instancias generadas no tendrán un alto $SCUMBLE$, pues el desacoplamiento lo habrá evitado, y por tanto serán más útiles a la hora de mejorar el rendimiento del clasificador.

En este apartado se han visto algunos ejemplos de algoritmos de sobremuestreo, tanto tradicionales como multietiqueta. Puesto que el objetivo de este trabajo es proponer un nuevo algoritmo de sobremuestreo multietiqueta que adapte los nuevos modelos generativos profundos, que tanto éxito han tenido al ser aplicados sobre imágenes, la siguiente sección se centra en dichos modelos.

2.7. Nuevos modelos generativos profundos

Los **modelos generativos profundos** representan un paradigma que ha ganado muchísimo interés a lo largo de los últimos años. Se trata de modelos de aprendizaje automático, más concretamente **aprendizaje profundo** (tienen, por tanto, estructura de redes neuronales) cuyo objetivo es aproximar una función de distribución de probabilidad lo máximo posible a la distribución de los datos originales. Esto permitiría, por tanto, generar nuevos datos sintéticos que, en teoría, no podrían diferenciarse estadísticamente de los originales.

La contraparte a este enfoque son los **modelos discriminativos profundos**, los cuales se podrían calificar como “tradicionales”. Dentro de la tarea de clasificación, estos modelos tienen como fin determinar, para cada clase (o etiqueta, en MLC), la

probabilidad condicional $P(Y|X)$, es decir, la probabilidad de que, para los valores de entrada X, la clase de la instancia sea Y (o que la etiqueta Y esté presente).

Sin embargo, los modelos generativos calculan una distribución de probabilidad $P(X,Y)$, o, si se desea generar variables de entrada teniendo en cuenta el valor de la salida, $P(X|Y)$, de modo que esta se aproxime lo máximo posible a la distribución real de los datos. Por tanto, se modela una función para todas las variables (o solo las de entrada), en lugar de solo para las variables de salida.

La historia de los modelos generativos profundos [103] comienza con las *Sigmoid Belief Networks* [104], cuyo entrenamiento era impracticable [105]. Por ese motivo, surge otro tipo de modelo generativo: las *Deep Belief Networks* [106], cuyo entrenamiento es mucho más rápido, debido a que sus dos últimas capas forman una máquina de Boltzmann restringida o RBM [107]. Posteriormente se propusieron otros modelos, como redes neuronales recurrentes [108] o máquinas de Boltzmann profundas [109].

Desde entonces hasta la actualidad, han surgido una gran cantidad de modelos generativos orientados, por lo general, a la **generación de imágenes**. En este apartado se describen brevemente algunos de ellos.

2.7.1. Variational Autoencoders

Un **autoencoder** [110] es un modelo de red neuronal con una estructura sencilla, la cual se puede observar en la parte de arriba de la Figura 2.11. A la vista de dicha estructura se puede entender su funcionamiento a alto nivel: **existen dos componentes**, uno que aprende a obtener una representación latente de los datos de entrada (*encoder*) y otro que, a partir de ese espacio latente, es capaz de reconstruir el dato original (*decoder*). Un **espacio latente** es una representación de menor dimensionalidad de un conjunto de datos, obtenida combinando atributos para dar lugar a características (*features*), intentando **minimizar la pérdida de información**.

Por tanto, el proceso de entrenamiento consistirá en presentar la instancia de entrada y **forzar que la salida sea igual que dicha entrada**, ajustándose los pesos de la red para que así sea. También pueden plantearse otros tipos de entrenamiento para que el modelo pueda llevar a cabo distintas tareas, como reducción de dimensionalidad, detección de anomalías e incluso clasificación o agrupamiento [111].

Ahora bien, el modelo aprendido es un vector dentro de un espacio latente n-dimensional. No obstante, si el objetivo es generar instancias nuevas, es necesaria

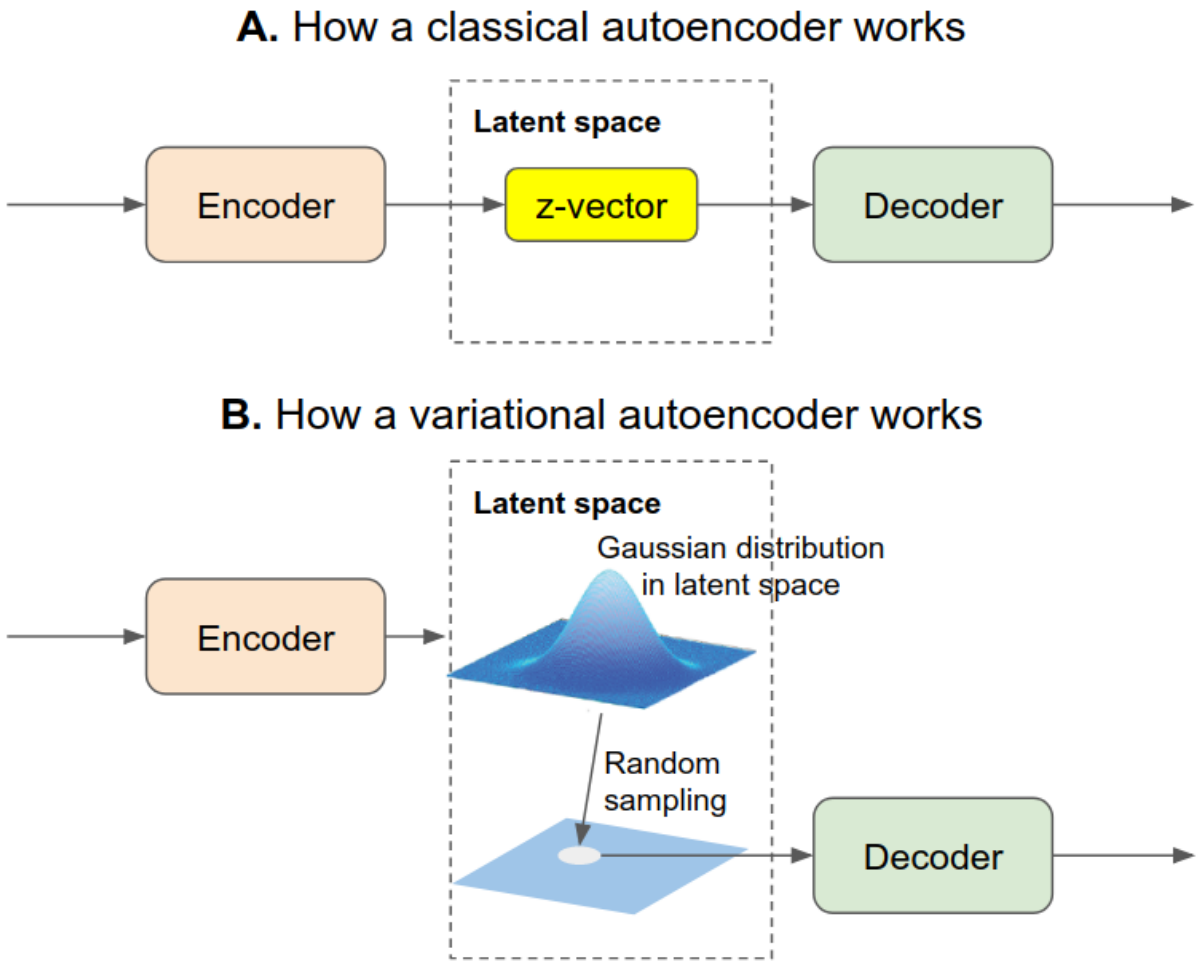

Figura 2.11: Estructura de un autoencoder y de un VAE. Fuente: [112].

una **distribución de probabilidad** que permita generar vectores dentro del espacio latente, los cuales posteriormente puedan ser decodificados para dar lugar a una **instancia sintética** (una imagen, por ejemplo) que siga la distribución del conjunto de datos de entrenamiento. Aquí es donde entra el concepto de *variational autoencoder* o VAE [113], cuya única diferencia con un autoencoder normal es que, tal como se observa en la parte de abajo de la Figura 2.11, en lugar de mapear la entrada a un vector, se obtiene una **distribución de probabilidad gaussiana**, definida por un vector media y un vector desviación típica. Esta distribución de probabilidad permite generar nuevos vectores latentes, los cuales serán decodificados por el *decoder* y darán lugar a **nuevas instancias**.

2.7.2. Generative Adversarial Networks

Las redes generativas antagónicas o GAN [114] basan su funcionamiento en el **aprendizaje antagónico**. Este concepto hace referencia al entrenamiento de dos modelos cuya función a optimizar es opuesta; es decir, se trata de la misma función, pero un modelo intenta minimizarla, y el otro, maximizarla. De este modo, los modelos **se entrenan a sí mismos**.

Un modelo GAN es una red neuronal compuesta por **dos módulos**: un generador y un discriminador. El **generador** recibe como entrada un vector con ruido, producido a partir de una distribución (normalmente gaussiana), y debe proporcionar como salida una nueva instancia transformando dicho ruido. Por otro lado, el **discriminador** recibe como entrada una instancia, y debe ser capaz de distinguir si esta es verdadera (parte del conjunto de entrenamiento) o falsa (creada por el generador). La arquitectura de una red GAN se muestra en la Figura 2.12.

De este modo, el entrenamiento se desarrolla suministrando instancias de cada una de las dos fuentes (datos de entrenamiento y generador) al discriminador, y llevando a cabo *backpropagation* en ambos módulos, teniendo en cuenta que **la función pérdida del discriminador debe ser minimizada por el mismo y maximizada por el generador**.

Idealmente, el entrenamiento finalizará cuando el grado de confianza de las decisiones del discriminador sea del 50 %, es decir, cuando este sea incapaz de distinguir si una instancia proviene de los datos de entrenamiento o si, por el contrario, se trata de una instancia sintética. En ese momento, **el módulo generador es el modelo generativo buscado**.

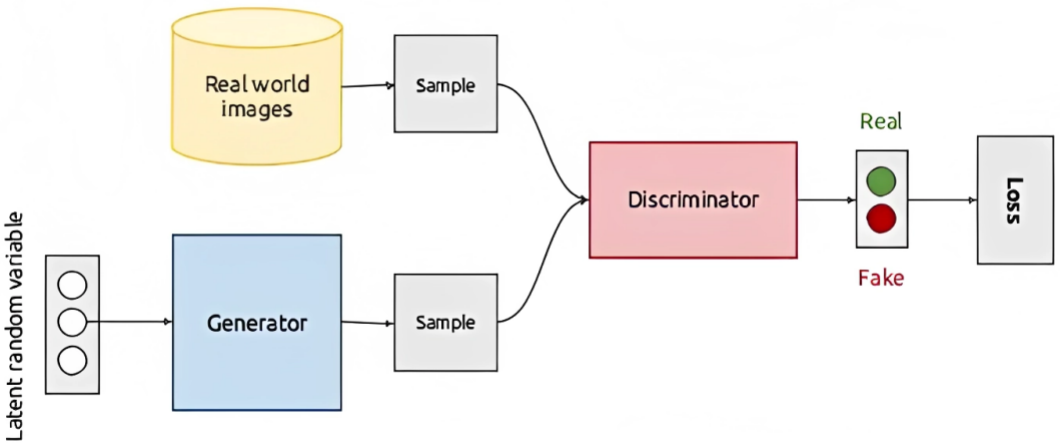

Figura 2.12: Estructura de una GAN. Fuente: [115].

Por lo general, **las GAN ofrecen mejores resultados que los VAE**, al menos en imágenes [116]. No obstante, su entrenamiento es mucho más lento y costoso computacionalmente, debido a la inmensa cantidad de parámetros a ajustar durante el mismo.

2.7.3. Adversarial Autoencoders

Los **autoencoders antagónicos** o AAE [117] son un híbrido entre un autoencoder normal (no variacional) y una GAN, proporcionando mejores resultados que el primero pero manteniendo su principal ventaja: la facilidad del entrenamiento.

La estructura es algo compleja, pero se puede entender fácilmente a la vista de la Figura 2.13.

Como se observa en en la parte de arriba de dicha figura, el modelo cuenta con un **autoencoder clásico**, con una entrada, varias capas ocultas, un vector latente como capa central, y otro conjunto de capas simétricas que producen la salida. No obstante, este modelo incluye también una **segunda red neuronal**, que toma como entrada muestras tomadas de una distribución deseada y muestras tomadas de la distribución descrita por el vector latente del autoencoder. Dicha red neuronal deberá ser capaz de distinguir entre los dos tipos de muestras, al igual que el discriminador de una GAN. De este modo, el autoencoder hace la función de generador de una GAN, y el discriminador es la red neuronal ya mencionada.

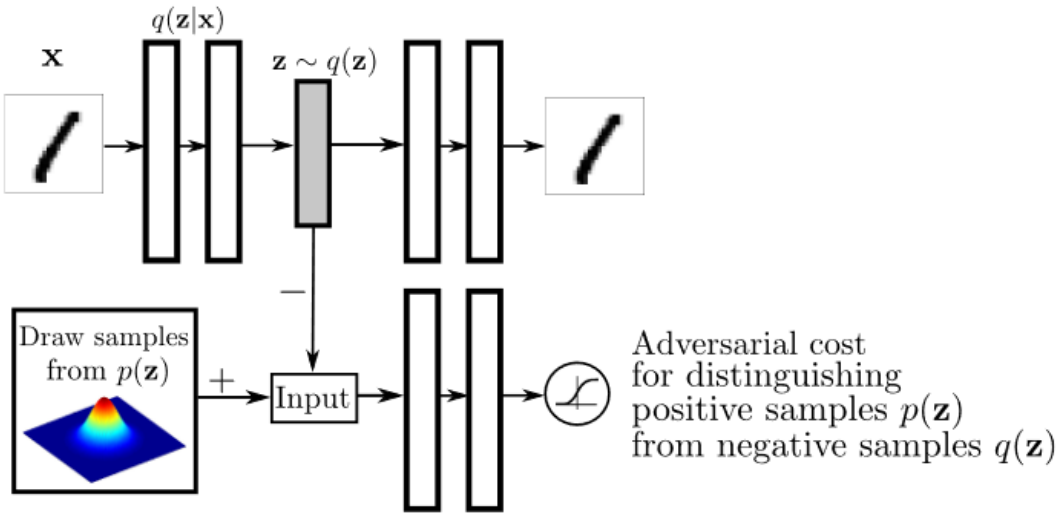

Figura 2.13: Estructura de un AAE. Fuente: [117].

Al igual que en una GAN, la función a optimizar es opuesta para ambas redes neuronales: el autoencoder trata de maximizarla, y la red neuronal discriminadora trata de minimizarla.

El entrenamiento se lleva a cabo de manera un tanto peculiar, pues el autoencoder tiene **dos funciones a optimizar**: la de un autoencoder normal, es decir, minimizar la diferencia entre la imagen de entrada y la de salida; y la función antagónica, para conseguir engañar al discriminador.

En definitiva, los AAE son una buena opción si se desea obtener buenos resultados, sin el elevado coste computacional que acarrea el entrenamiento de una GAN.

2.7.4. Normalizing flows

Los modelos *Normalizing Flows* o NF [118] tratan de estimar la distribución de probabilidad a partir de la cual se generan los datos del conjunto de entrenamiento basándose en transformaciones invertibles.

Una **transformación invertible** no es más que una función biyectiva: aquella en la que, para un valor de salida dado, existe **un único valor de entrada posible**.

El funcionamiento de los NF consiste en tomar como entrada las instancias de entrenamiento, cuya distribución de probabilidad se desea aproximar, y **forzar que la salida del modelo sea ruido gaussiano**. El concepto de ruido gaussiano hace referencia a instancias generadas a partir de una distribución normal. De este modo, el modelo es entrenado para que, gracias a la combinación de funciones invertibles,

sea capaz de obtener una instancia que siga una distribución normal, partiendo de una muestra que siga la distribución a estimar.

De partida, no parece muy útil entrenar un modelo que obtenga ruido. No obstante, en este momento es cuando cobra importancia el hecho de que las transformaciones aprendidas sean invertibles, pues una vez el modelo ha sido entrenado, **es posible obtener la inversa de la combinación de funciones aprendidas**.

De este modo, para generar instancias nuevas, que sigan la distribución de los datos de entrenamiento, bastará con **aplicar las inversas de las transformaciones aprendidas** por el modelo sobre ruido gaussiano.

En la Figura 2.14 se puede visualizar el proceso descrito, con un primer paso en el que se obtienen las funciones "originales", y una segunda etapa en la que, a partir de su inversa, se generan instancias sintéticas.

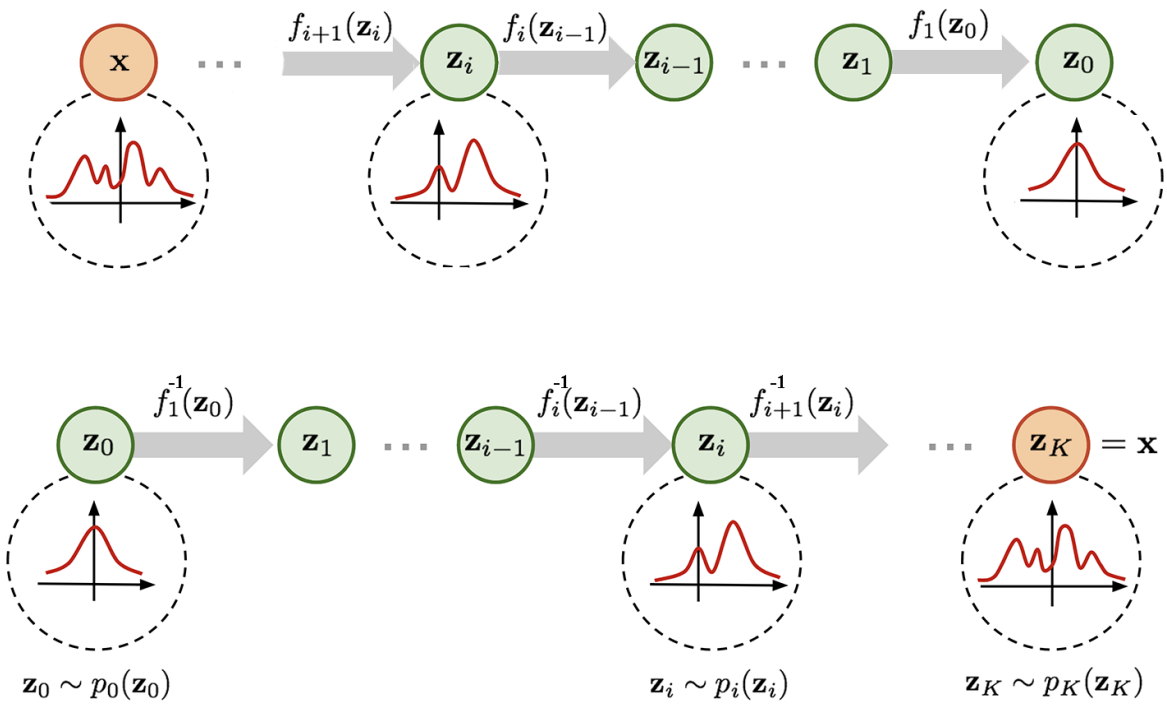

Figura 2.14: Estructura de un modelo NF. Elaboración propia a partir de [119].

Como se puede apreciar, los modelos NF tienen una gran base matemática. Esta, junto con su sencillez, son las mayores ventajas que ofrecen. En cuanto a resultados, son modelos competitivos, aunque se mantienen por detrás de las GAN [116].

2.7.5. Gaussian Mixture Models

Los **modelos mixtos gaussianos**, en inglés *Gaussian Mixture Models* o GMM [120], suelen emplearse para agrupamiento, como alternativa al famoso algoritmo *k-means*, ya que funcionan mejor cuando las instancias de distintas clases se solapan.

Los algoritmos que construyen modelos GMM funcionan de la siguiente manera: se generan aleatoriamente tantos gaussianos como *clusters* se quieran obtener, normalmente tantos como clases en un problema de clasificación. Un gaussiano no es más que una **distribución normal**, con una media y una desviación típica. Además, cada gaussiano también cuenta con un parámetro peso, que gráficamente se corresponde con la altura del máximo de la función, y que afecta a las probabilidades de pertenencia de las instancias al gaussiano.

A partir de este punto, se seguirá un proceso iterativo: el llamado algoritmo de **maximización de expectativa** [121], que calcula la probabilidad de pertenencia de cada instancia a cada gaussiano (expectativa) y, tras ello, se modifican los tres parámetros que lo definen, para **maximizar la pertenencia de las instancias al gaussiano correspondiente, y la no correspondencia otros**. Se puede observar el resultado de la aplicación de este algoritmo en la Figura 2.15.

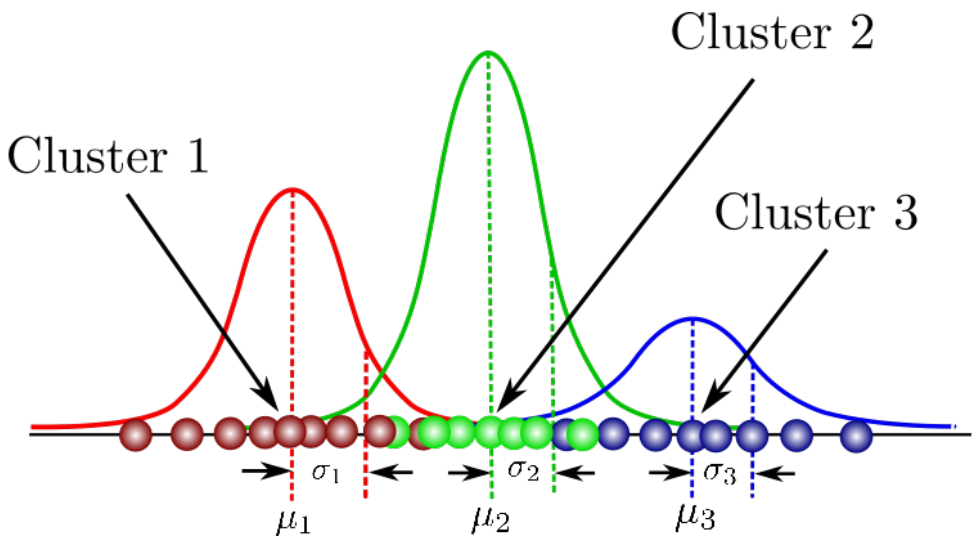

Figura 2.15: Resultado del algoritmo de maximización de expectativa en un GMM. Fuente: [122].

Aunque, como se ha apuntado, los GMM se suelen emplear para tareas de *clustering*, una vez finalizada la etapa de maximización de expectativa, el modelo consiste en un conjunto de distribuciones distintas que describen los datos pertenecientes a cada clase. De este modo, se pueden generar nuevas instancias de una clase concreta, **obteniéndolas a partir de dichas distribuciones** [123].

2.7.6. Diffusion Models

Los **modelos de difusión**, en inglés *Diffusion Models* o DM [124] son los más recientes de entre los recogidos en este apartado y, pese a no haber sido foco de tanta investigación, ya **superan en síntesis de imágenes** a muchos modelos basados en GAN [125].

Al igual que los modelos NF, presentados anteriormente, se basan en la **obtención de una muestra partiendo del ruido generado por una distribución gaussiana**. También comparten con dichos modelos la presencia de dos procesos: uno hacia delante y otro hacia atrás, tal y como se aprecia en la Figura 2.16.

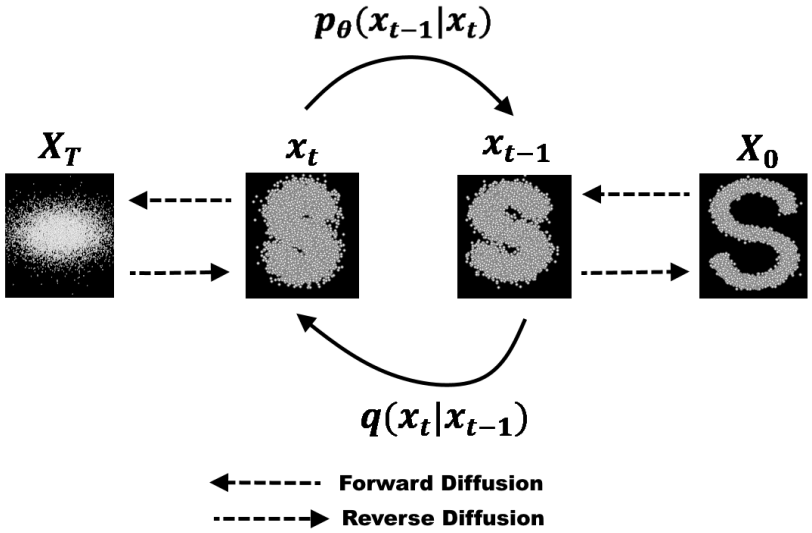

Figura 2.16: Estructura de un DM. Fuente: [126].

En la fase hacia delante, **se añade ruido a una instancia** paso a paso, hasta que el resultado es indistinguible de la distribución estándar del ruido gaussiano. Posteriormente, en la fase hacia atrás, se entrena un modelo de difusión, normalmente una red neuronal, para que **elimine el ruido** de la instancia, de nuevo paso a paso, hasta obtener la muestra original. El motivo por el cual el ruido se añade y elimina de manera incremental es que se obtienen mejores resultados que si se intentara forzar la muestra inicial a partir del ruido gaussiano en una sola transformación. De este modo, en cada etapa del proceso hacia atrás se predice el ruido introducido en la misma etapa de la fase hacia delante, representado como una distribución con media y varianza.

Un detalle importante es que el ruido introducido en cada paso del proceso hacia delante **no es constante**, pues en ese caso las predicciones de cada etapa hacia atrás siempre serían las mismas. Por este motivo, existen dos alternativas para incrementar el ruido en cada paso: **incremento lineal y cosenoidal**. En el primer caso, la instancia

se llena de ruido en pocos pasos, de modo que la información se destruye muy rápido. En el caso del incremento cosenoidal, definido por una ecuación basada en el coseno, la convergencia (es decir, la transformación en ruido) es más lenta.

Como se ha mencionado, los DM son los modelos más recientes de entre los que se recogen en este apartado, datando su propuesta de 2015. En ellos se basan los generadores de imágenes a partir de texto que se han popularizado desde finales del año 2022, como *Dall-E 2* [16] o *Stable Diffusion* [127].

2.7.7. Tabla comparativa

A continuación se incluye una tabla comparativa entre los modelos presentados anteriormente. En ella se recogen las ventajas e inconvenientes de cada modelo, con el fin de compararlos para determinar cuál de ellos servirá como base para el algoritmo a proponer en el trabajo.

Modelo	Ventajas	Inconvenientes
VAE	Rapidez, tanto de entrenamiento como de funcionamiento Facilidad de entrenamiento (comparado con GAN) Existencia de métricas objetivas de comparación (por ejemplo, *log-likelihood*)	Imágenes más borrosas que GAN debido al ruido inyectado y a la imperfecta reconstrucción del decodificador Al tener una función específica a maximizar,puede ocurrir que los resultados no optimicen otros aspectos (como las GAN)
GAN	Rapidez en funcionamiento, solo superadas por VAE Muy buenos resultados (al menos en imágenes) Interpretabilidad del espacio latente generado	Dificultad del entrenamiento Inexistencia de una función objetivo Difícil convergencia Sensible al desvanecimiento de gradientes si el discriminador es muy bueno

Modelo	Ventajas	Inconvenientes
AAE	Mejores resultados que VAE (la distribución generada se adapta mejor a la distribución de los datos) A diferencia de *GAN*, más fáciles de entrenar y capaces de intentar aproximarse a una distribución normal	Peores resultados que GAN Poca investigación
NF	Se obtiene una estimación exacta de la probabilidad de la muestra generada Mayor exactitud en muestreo e inferencia de datos Mayor simplicidad de las distribuciones aprendidas, al forzarse su similaridad a una normal (0,1)	Menos expresividad que otras aproximaciones Para conseguir biyectividad, se debe preservar el volumen sobre transformaciones, lo cual lleva a espacios latentes de mayor dimensión que, a su vez, provoca un menor rendimiento e interpretabilidad No tan buena calidad en imágenes comparado con GAN y VAE Algunas transformaciones llevadas a cabo en NF ignoran la interacción entre píxeles cercanos, procesando la información en canal (solo es un problema en imágenes)
DM	Muy buena calidad en los resultados Tratabilidad y flexibilidad (normalmente opuestas). Tratabilidad significa que es analíticamente evaluable y adaptable a los datos sencillos. Flexibilidad significa que es capaz de describir estructuras de datos más complejas, eso sí, a costa de una mayor complejidad	Más lentas que las GAN, incluso con mejoras. Esto es debido a las largas cadenas de *Markov* de las que se componen
GMM	Pueden adaptarse a *clusters* con gran variedad de formas En cuanto al *clustering*, tiene la ventaja de que su clasificación es suave, produciendo probabilidades, en lugar de binaria (como *k-means*)	Funcionamiento más lento que otros modelos, dado que tarda en converger Pueden dar lugar a distribuciones muy complejas Dificultad de entrenamiento (en parte debido al punto anterior)

Tabla. 2.2: Ventajas e inconvenientes de los modelos generativos presentados

Capítulo 3

OBJETIVOS

En este breve capítulo se presentan los objetivos, tanto generales como específicos, que se pretenden cumplir llegada la finalización de este trabajo.

3.1. Objetivo general

Como se ha explicado con anterioridad, el desbalanceo es uno de los problemas más estudiados en la tarea de clasificación, y más aún en la clasificación multietiqueta, al verse más afectada por el mismo.

Una solución al desbalanceo que funcione correctamente puede mejorar el rendimiento de los clasificadores en gran medida. Por ese motivo, **en este trabajo se propone una**.

Los modelos generativos, también explicados anteriormente, han supuesto un cambio de paradigma en el ámbito de la generación de imágenes, al conseguir resultados prácticamente indistinguibles de aquellas generadas por personas. Por ese motivo, el objetivo principal de este trabajo es **adaptar uno de estos modelos** (en concreto, un modelo de difusión), diseñados para funcionar con imágenes, y **conseguir que genere datos multietiqueta**.

De esta manera, se propondrá un **algoritmo de sobremuestreo de datos multietiqueta**, capaz de generar datos con etiquetas minoritarias, por medio de modelos generativos.

3.2. Objetivos específicos

Se pueden desglosar los objetivos generales, presentados en el apartado anterior, en una serie de objetivos específicos, recogidos a continuación.

- **Obtención de una visión general del campo de la Ciencia de datos y los métodos de Minería de datos.** Con ese fin, se debe llevar a cabo un estudio de conceptos de este ámbito, como el proceso KDD o el procedimiento de evaluación de un método. De este modo, se puede adquirir un conocimiento esencial para enmarcar el trabajo desempeñado en el campo al que pertenece, que es el aprendizaje multietiqueta, pero también el aprendizaje profundo.
- **Recopilación de conjuntos de datos multietiqueta y análisis de los problemas de desbalanceo en los mismos.** De este modo, es posible identificar el problema a resolver, y comprender su impacto en el proceso de clasificación. Para ello, se recurre a recursos como el repositorio Cometa [128] y el paquete de `R` `mldr.datasets` [129], que facilitan la obtención de los MLD, y el paquete de `R` `mldr` [130], que permite analizar dichos MLD, y así apreciar el impacto del desbalanceo en los mismos.
- **Revisión del estado del arte en el ámbito del remuestreo en conjunto de datos multietiqueta, a fin de reducir el mencionado desbalanceo.** Con este fin, se estudiaron los artículos con propuestas de métodos existentes en la bibliografía especializada. A esta revisión se ha dedicado la sección 2.6.2 del capítulo anterior.
- **Desarrollo de un paquete con algoritmos de remuestreo multietiqueta.** El paquete se desarrollará en el lenguaje de programación `R`, y tendrá el nombre de `mldr.resampling`. Esta herramienta permitirá conocer en profundidad el funcionamiento de los métodos propuestos hasta el momento, realizar estudios experimentales con la propuesta desarrollada, asociada al objetivo general del TFG, y poner a disposición de la comunidad científica las implementaciones de referencia de métodos de remuestreo para MLD. La sección 4.1 se dedica a describir el mencionado paquete.
- **Estudio de los nuevos modelos generativos, basados en técnicas de aprendizaje profundo.** Ejemplos de estos modelos son los *Variational Autoencoder*, *Generative Adversarial Networks* o *Diffusion Models*. El fin de este estudio es comprender su funcionamiento, para poder determinar cuál de ellos tiene un mejor rendimiento, y elegirlo como modelo a adaptar a datos multietiqueta. Este estudio se ha abordado en la sección 2.7 del capítulo anterior.

- **Diseño de un modelo de difusión para la generación de datos multietiqueta sintéticos, con el fin de reducir el desbalanceo.** Para ello, se toma como base un método existente, que permite el uso de modelos de difusión sobre datos tabulares binarios y multiclase, y se diseña a partir de él una propuesta que sea capaz de generar datos multietiqueta sintéticos con el objetivo de reducir el problema del desbalanceo. La propuesta del modelo tiene lugar en la sección 4.2.2 del siguiente capítulo.

- **Implementación del modelo, empleando el framework adecuado.**. Para ello, previamente se estudiarán las distintas herramientas disponibles, eligiendo finalmente un *framework* de aprendizaje profundo con el que llevar a cabo la implementación, en este caso PyTorch [131]. En la sección 4.2.3 se recoge el mencionado estudio.

- **Evaluación del modelo mediante experimentación y análisis de resultados.** Se evaluará el rendimiento del modelo, comparándolo con otros del estado del arte ya descritos, implementados en el paquete `mldr.resampling`. Dicha evaluación tendrá lugar comparando el rendimiento de varios clasificadores multietiqueta, implementados en el *framework* MULAN [132], de `Java`. El capítulo 5 describe todo este proceso.

- **Redacción de manuales de instalación y uso.** Estos manuales están incluidos en los apéndices y en el repositorio del modelo y paquete desarrollados, así como la memoria correspondiente al TFG. Como en todo proyecto de investigación, en este trabajo se incluye la documentación pertinente sobre el método propuesto, así como sobre el proceso de instalación y uso del mismo, la cual se encuentra disponible en el repositorio de Github [1] asociado al método. La misma información, pero relativa al paquete, se puede encontrar en otro repositorio[2].

Una vez expuestos los objetivos del trabajo, se puede continuar con los mismos, presentando los materiales y métodos empleados e incluyendo la propuesta del nuevo método.

[1] https://github.com/madr0008/mldm
[2] https://github.com/madr0008/mldr.resampling

Capítulo 4

MATERIALES Y MÉTODOS

En este capítulo se describen los materiales que se han empleado en este trabajo para conseguir los objetivos propuestos en el capítulo anterior. Además, se detallan aspectos generales del paquete de métodos de remuestreo multietiqueta realizado en paralelo a este trabajo. De igual modo, se describe el diseño y desarrollo del modelo propuesto, y se detalla la experimentación llevada a cabo para determinar el rendimiento del mismo en comparación con los algoritmos del estado del arte.

4.1. Paquete con los algoritmos del estado del arte

En paralelo con la escritura de esta memoria, se ha desarrollado un paquete para el entorno de programación `R`, llamado `mldr.resampling` [95]. En dicho paquete se incluyen las implementaciones de los algoritmos de sobremuestreo multietiqueta descritos en el apartado 2.6.2, además de otros algoritmos de bajomuestreo multietiqueta. En esta sección se describe en detalle la arquitectura y funcionamiento de este paquete

4.1.1. Arquitectura del paquete

El paquete `mldr.resampling` trabaja con MLD, para lo cual aprovecha la infraestructura ofrecida por el paquete `mldr` [130], como una clase de `R` de tipo `S3` llamada también `mldr`. `S3` es un mecanismo orientado a objetos que depende de funciones genéricas. Por tanto, la mayoría de funciones exportadas por `mldr.resampling` toman un objeto `mldr` como parámetro, que puede importarse y exportarse fácilmente con funciones del paquete `mldr`.

Los algoritmos de remuestreo incluidos en el paquete, recogidos en la sección 4.1.2, están implementados como **métodos independientes**. De este modo, se permite a los usuarios experimentar con los algoritmos que desee, sin necesidad de capas adicionales.

Además de aplicar un algoritmo a un MLD, el paquete también considera la posibilidad de **ejecutar varios métodos** para compararlos. Para conseguir este objetivo, se facilita una función que automatiza el proceso de llamada a cada uno de los algoritmos especificados sobre un mismo MLD, usando los parámetros indicados también por el usuario.

4.1.2. Algoritmos implementados

Los algoritmos implementados en el paquete `mldr.resampling` se recogen en la Tabla 4.1.

Algoritmo	Tipo de muestreo	Referencia
LPROS	Sobremuestreo	[96]
MLRkNNOS	Sobremuestreo	[97]
MLROS	Sobremuestreo	[96]
MLSMOTE	Sobremuestreo	[98]
MLSOL	Sobremuestreo	[99]
REMEDIAL	Sobremuestreo	[100]
LPRUS	Bajomuestreo	[96]
MLeNN	Bajomuestreo	[133]
MLRUS	Bajomuestreo	[96]
MLTL	Bajomuestreo	[134]
MLUL	Bajomuestreo	[99]

Tabla. 4.1: Algoritmos implementados en el paquete `mldr.resampling`

En la ayuda integrada en el paquete [95] se describe cómo usar sus métodos, permitiéndose la ejecución individual de cada algoritmo o de forma colectiva, aprovechando el cálculo de vecinos (necesario en muchos algoritmos) para **reducir el tiempo de espera**.

4.1.3. Funcionalidades del paquete

Las principales contribuciones del paquete `mldr.resampling` son las siguientes:

- **Implementaciones de referencia** de once algoritmos de remuestreo para MLD, con código fuente disponible tras la instalación del paquete.

- **Una interfaz unificada** que facilita la aplicación de varios algoritmos a un MLD, automatizando varios de los pasos requeridos para compararlos.

- **Incorporación de optimizaciones** con el fin de acelerar la búsqueda de vecinos a través de la paralelización de tareas y el almacenamiento de vecinos en caché.

Estas funcionalidades pueden ser accedidas de maneras distintas dependiendo de las necesidades del usuario. Para aplicar alguno de los algoritmos implementados en el paquete, basta con llamar a una **función con el nombre del propio algoritmo**, todo en mayúscula y junto, especificando los parámetros pertinentes, que pueden consultarse en la documentación.

Es bastante común estar interesado en probar varios enfoques de remuestreo distintos sobre el mismo MLD, para poder compararlos y así elegir el mejor. Aunque los métodos individuales siguen distintas estrategias para crear o eliminar instancias, muchos de ellos dependen de la información obtenida a raíz de los vecinos de cada muestra. Debido a que la mayoría de los MLD tienen cientos o incluso miles de atributos, así como de instancias, **encontrar los vecinos más cercanos a cada una de las instancias tiene un gran impacto en el tiempo necesario para la ejecución**.

El paquete `mldr.resampling` provee un método adicional, llamado `resample(mld, algorithms)`, capaz de ejecutar todos los métodos especificados por el parámetro `algorithms` sobre el mismo `mld`. Esto no solo automatiza un trabajo que, de lo contrario, requeriría múltiples llamadas a funciones, sino que también **mejora la eficiencia** de estos métodos a través de un **mecanismo de caché**. Una vez se obtiene la información sobre vecinos más cercanos, esta es almacenada en caché y pasada como entrada a los métodos individuales, de modo que esta tarea no consume tiempo.

Otra prestación del paquete, accesible a través de la función `setParallel()`, es la habilidad de **distribuir el cómputo entre los hilos del sistema**, en lugar de usar uno solo. Es posible limitar el número de hilos usados por los algoritmos usando la función `setNumCores()`.

El conjunto de algoritmos incluidos en el paquete seguirá creciendo conforme se publiquen nuevas propuestas que cuenten con guías de implementación (pseudocódigo, diagrama de flujo, etc). La estructura del código del paquete hace este proceso muy sencillo.

4.1.4. Detalles de implementación

Todos los métodos han sido implementados en R, de modo que el paquete no tiene que ser compilado, y **el código fuente está disponible** para cualquier usuario de R interesado en profundizar en el mismo. Cada método ha sido escrito teniendo en cuenta las pautas establecidas en sus propuestas. En los casos en los que el código fuente original estaba disponible, los resultados de la nueva implementación se compararon con los de la original, para asegurar la consistencia.

Aunque tener todos los métodos disponibles en un lenguaje como R los hace accesibles a un amplio grupo de investigadores, R no es uno de los lenguajes más eficientes en cuanto a velocidad de ejecución. Por esta razón, el paquete `mldr.resampling` ha sido escrito **teniendo en cuenta la eficiencia** a través de los siguientes enfoques:

- El proceso de recolección de basura en la mayoría de lenguajes dinámicos puede tener un gran impacto en los tiempos de ejecución. Puesto que la mayoría de objetos de R son inmutables, el crecimiento de ciertas estructuras de datos, como vectores y matrices, implica asignar nuevos bloques de memoria, copiando los datos antiguos y liberando la memoria previamente asignada. El paquete `mldr.resampling` **aprovecha las facilidades de vectorización del lenguaje** siempre que es posible, evitando bucles para minimizar el tiempo dedicado a recolectar basura.
- Aprovechar el poder de los actuales procesadores multihilo es esencial para reducir el tiempo de ejecución. Sin embargo, pocos lenguajes ofrecen una manera estandarizada de hacerlo, que funcione en todos los sistemas operativos. El código de `mldr.resampling` está diseñado para que algunas tareas puedan ejecutarse secuencialmente o **en paralelo**, dependiendo de si el paquete `parallel` de R está instalado. Por defecto, la paralelización no está activa, por lo que el paquete `parallel` no se requiere. En caso de estar disponible[1], el usuario puede activar el paralelismo llamando a la función `setParallel()` con el valor `TRUE` como parámetro.
- Muchos de los métodos de remuestreo del paquete se basan en búsqueda de vecinos. En su implementación original, esta es la tarea que consume la mayor parte del tiempo de ejecución. Para atributos numéricos, encontrar los vecinos más cercanos de una muestra implica cálculo de la distancia, casi siempre euclídea, al resto de instancias del MLD. Si el conjunto de datos contiene atributos nominales, el cálculo de distancias es mucho más costoso con VDM. Todos

[1] En las últimas versiones de R este paquete se incluye en la instalación base

los métodos disponibles en `mldr.resampling` toman dos parámetros opcionales: `neighbors` y `tableVDM`, los cuales **permiten reusar cálculos previos**. Este método de caché se usa automáticamente al llamar a la función `resample()`, la cual sirve para ejecutar más de un algoritmo sobre el mismo MLD.

4.1.5. Ejemplos ilustrativos

La última versión estable de `mldr.resampling` está disponible en CRAN, de modo que puede ser instalada como cualquier otro paquete de `R`: llamando a la función `install.packages()` como se muestra a continuación. Una vez instalado, la función `library()` cargará el paquete y mostrará un mensaje al usuario sobre cómo activar el procesamiento paralelo.

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

Se necesitan algunos MLD para ejecutar los algoritmos. La instalación del paquete `mldr.resampling` implica que `mldr` también ha sido instalado. Mediante el uso del comando `data(package="mldr")` se mostrarán por pantalla los MLD disponibles en este paquete. Cualquier otro MLD puede ser obtenido del sistema de archivos o descargado del repositorio Cometa [128] usando las funciones proporcionadas por el paquete `mldr.datasets` [129].

Por medio de la mencionada función `data()`, el MLD `emotions` se carga en memoria y se obtienen algunas métricas, como el número de instancias, $MeanIR$ y $SCUMBLE$:

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

> emotions$meanIR
[1] 1.478068

> emotions$scumble
[1] 0.01095238
```

Este MLD está levemente desbalanceado, siendo $MeanIR = 1{,}4781$, y el acoplamiento entre etiquetas mayoritarias y minoritarias también es moderado, teniendo un $SCUMBLE = 0{,}0109$. Sin embargo, se puede beneficiar de la aplicación de algún al-

goritmo de remuestreo. Una primera opción podría ser el método de sobremuestreo simple LPROS:

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

$meanIR
[1] 1.355174

$scumble
[1] 0.007740464
```

La llamada a la función `LPROS()` devuelve un nuevo objeto `mldr` que representa el MLD después del remuestreo. Como se puede apreciar a raíz de las métricas, esta versión del conjunto de datos tiene unas pocas muestras más, y tanto el $MeanIR$ como el $SCUMBLE$ han disminuido, lo cual es un punto positivo.

Puesto que todas las funciones correspondientes a métodos de remuestreo devuelven un objeto `mldr` como resultado, encadenar varios métodos es bastante fácil. Por ejemplo, LPROS crea nuevas muerstras basadas en combinaciones de etiquetas minoritarias, en lugar de etiquetas individuales. Un MLD con un alto grado de acoplamiento entre etiquetas minoritarias y mayoritarias apenas se beneficiaría de este algoritmo. Sin embargo, se pueden desacoplar las etiquetas antes de LPROS usando el algoritmo REMEDIAL, tal y como se muestra a continuación:

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

$meanIR
[1] 1.18863

$scumble
[1] 0.00224451
```

Tanto el $MeanIR$ como el $SCUMBLE$ son ahora mucho mejores que al aplicar solo LPROS.

Por defecto, las funciones de `mldr.resampling` no usan paralelización. Por este motivo, algunos algoritmos (aquellos que llevan a cabo cálculo de vecinos) consumen una cantidad de tiempo considerable. Para comunicar al usuario en qué estado se

encuentra el proceso y estimar el tiempo restante se usa una barra de progreso. Una vez termina la ejecución del método, como en el siguiente ejemplo, el MLD resultante está disponible como con cualquier otra función.

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

$meanIR
[1] 1.298585

$scumble
[1] 0.004986872
```

Distribuir la carga de trabajo a través de todos los hilos de procesamiento del equipo del usuario tan solo requiere una llamada a la función `setParallel(TRUE)`. Esta función comprueba si el paquete `parallel` está instalado, y notifica al usuario en caso de no estarlo. Una vez activado el procesamiento paralelo, cualquier ejecución de un método aprovechará esta capacidad. En el siguiente ejemplo, el mismo algoritmo MLSMOTE tardó aproximadamente un cuarto del tiempo en ejecutarse.

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

Aunque llamar a las funciones asociadas con métodos individuales permite al usuario probar cualquiera de ellas, un proceso de experimentación implicará, por regla general, la ejecución de varios algoritmos. Este trabajo puede automatizarse usando la función `resample()`, como se muestra en el siguiente ejemplo. En este caso, se aplican cuatro algoritmos de remuestreo distintos al mismo MLD. Como se puede leer en la salida de la función, el primer paso es calcular varias estructuras que actúan como una caché de vecinos cercanos. De esta forma, el tiempo de ejecución total será más corto que si cada algoritmo buscara su propio conjunto de vecinos.

```
> resample(emotions,
    c("MLROS", "MLRUS", "MLeNN", "MLSOL"),
    outputDirectory="~/datasets")
# Calculating structures for dataset musicout , if necessary. Once this is done, algorithms will be
    applied faster
# Calculating VDM table for dataset musicout
# Time taken (in seconds): 0.0046391487121582
# Calculating neighbors structure for dataset musicout . Once this is done, algorithms will be applied
    faster
```

```
# Time taken (in seconds): 485.649948835373
# Running MLROS on musicout with P = 25
# Time taken (in seconds): 0.0200722217559814
# Running MLRUS on musicout with P = 25
# Time taken (in seconds): 0.0121262073516846
# Running MLeNN on musicout with TH = 0.5 and k = 3
# Time taken (in seconds): 0.33689546585083
# Running MLSOL on musicout with P = 25 and k = 3
# Part 1/3: Neighbors were already calculated. That just saved us a lot of time!
# Part 2/3: Calculating auxiliary structures
# Part 3/3: Generating new instances
# Time taken (in seconds): 3.47672557830811
# End of execution. Generated MLDs stored under directory  /datasets
# algorithm time
# 1 MLROS 0.0200722217559814
# 2 MLRUS 0.0121262073516846
# 3 MLeNN 485.991483449936
# 4 MLSOL 489.131313562393
```

Puesto que se crean múltiples versiones del MLD, una por cada algoritmo, la función `resample()` los almacenará en el sistema de archivos en lugar de en memoria. Por defecto se usará un directorio temporal, pero el usuario puede especificar cualquier ruta con el parámetro `outputDirectory`. El nombre de los ficheros generados, que se puede observar en la Figura 4.1, será una combinación del nombre del MLD original, el algoritmo aplicado y sus parámetros.

```
mdavila@mdavila-HP:~$ ls datasets/ -l
total 1720
-rw-rw-r-- 1 mdavila mdavila 379839 jun 27 10:25 musicout_MLeNN_TH_0.5_k_3.arff
-rw-rw-r-- 1 mdavila mdavila 477168 jun 27 10:25 musicout_MLROS_P_25.arff
-rw-rw-r-- 1 mdavila mdavila 327743 jun 27 10:25 musicout_MLRUS_P_25.arff
-rw-rw-r-- 1 mdavila mdavila 568023 jun 27 10:26 musicout_MLSOL_P_25_k_3.arff
```

Figura 4.1: Ficheros generados por la función resample()

Una vez todos los MLD procesados están disponibles, las distintas funcionalidades del paquete `mldr` permiten al usuario analizar cómo cada algoritmo ha afectado al MLD a través de una serie de métricas y gráficas.

4.1.6. Publicación y uso del paquete

Como ya se ha indicado, la versión estable del paquete `mldr.resampling` está publicada en el **repositorio oficial** de paquetes de `R`, llamado CRAN. Esta publicación permite la sencilla instalación del paquete, como se ha mostrado anteriormente.

Igualmente, se puede acceder al código fuente de versiones más recientes a través de la plataforma Github[2], por medio de la cual también puede instalarse el paquete, como se explica en el propio repositorio.

Desde la publicación de `mldr.resampling` en CRAN, el 21 de abril de 2023, este cuenta con un total de **820 descargas**. Se espera que el número de instalaciones aumente con la publicación del artículo [95] asociado al paquete en una revista científica.

Una vez explicado el paquete `mldr.resampling` [95], que implementa los algoritmos del estado del arte a ejecutar durante la experimentación, se puede proceder a presentar el método propuesto en este trabajo, llamado MLDM: *Multilabel Diffusion Model*.

4.2. Propuesta de modelo

En esta sección se propone el algoritmo propio de sobremuestreo de conjuntos de datos multietiqueta, que es el principal motivo de este trabajo. Primero, se detalla en profundidad el funcionamiento del algoritmo propuesto y, posteriormente, se describen las herramientas empleadas para el desarrollo del mismo, justificando cada una de ellas frente a las principales alternativas.

4.2.1. Diseño del algoritmo

En esta sección se detalla la **propuesta del algoritmo de sobremuestreo multietiqueta** que supone la razón de ser de este trabajo. En concreto, se trata de un modelo de difusión que, en lugar de imágenes, es entrenado con MLD y puede ser usado, posteriormente, para generar datos sintéticos que sigan la misma distribución que los originales, y que cuenten con etiquetas minoritarias, para poder balancear el conjunto de datos.

La propuesta se basa, como se ha mencionado, en modelos de difusión. En concreto, se tomará como referente el modelo TabDDPM [135], propuesto a comienzos del año 2023, que llevó a cabo la tarea de adaptar los DM para imágenes a datos tabulares binarios y multiclase, **pero no multietiqueta**. Además, en el artículo en el que se propone el modelo, la generación de instancias **no se aplica al sobremuestreo**, sino que el estudio se centra en demostrar que la distribución de los datos se

[2]Código fuente de mldr.resampling: https://github.com/madr0008/mldr.resampling

mantiene. Por tanto, la aportación del nuevo modelo consiste en la **adaptación a datos multietiqueta**, y en la **implementación de la tarea de sobremuestreo** sobre las etiquetas minoritarias.

La principal característica de TabDDPM, que supone la mayor diferencia con respecto a los DM para imágenes, es el tratamiento de los atributos categóricos. Las imágenes son, en realidad, matrices numéricas, en las que cada píxel está representado por tres valores numéricos, que denotan la intensidad de los colores rojo, verde y azul (RGB), en el caso de las pantallas de ordenador. Por su parte, **los datos tabulares pueden tener también valores categóricos** entre sus atributos. Por este motivo, los DM, que están orientados a imágenes, no contemplan la presencia de datos no numéricos, por lo que se precisa de una adaptación del modelo con este fin.

De este modo, surgen los **modelos de difusión multinomial** [136], que usan funciones de probabilidad multinomiales, capaces de modelar poblaciones de datos no numéricos, sino categóricos. Lo que hace TabDDPM es separar los atributos en numéricos y categóricos. Para los primeros, la distribución a buscar es una **gaussiana**, como en los DM originales. Sin embargo, para los segundos, la función de probabilidad que se contempla es una función **multinomial**. De este modo, **todos los atributos del conjunto de datos pueden ser modelados por sendas funciones de probabilidad**, haciendo posible el proceso de adición y eliminación de ruido en el que se basan los DM.

En la Figura 4.2 se muestra un esquema del modelo TabDDPM. Como se puede observar, el primer paso consiste en una **preparación de los datos**. Los datos numéricos se normalizan, llevando a cabo transformaciones cuantiles [137], que aproximan los valores numéricos a una distribución normal, facilitando el trabajo para el modelo. Por su parte, los atributos categóricos son transformados para que tengan una codificación *one-hot*, que no es más que un vector binario de tamaño igual al número de categorías posibles, en el que todos los valores están a cero, salvo el de la posición de la clase de la instancia, que toma el valor uno.

Una vez los atributos han sido transformados, estos son suministrados a una **red neuronal multicapa** (*Multilayer Perceptron* o MLP), que lleva a cabo la transformación de los datos en **ruido gaussiano**. Posteriormente, se emplea otro modelo, que puede ser otro MLP, para llevar a cabo el proceso inverso. Este último modelo es la **función de eliminación del ruido** (*denoising function*), la cual permite la generación de instancias sintéticas.

Una vez entendido el funcionamiento del modelo original, se llevaron a cabo una serie de cambios sobre el mismo, que permitieron su adaptación a datos multietiqueta,

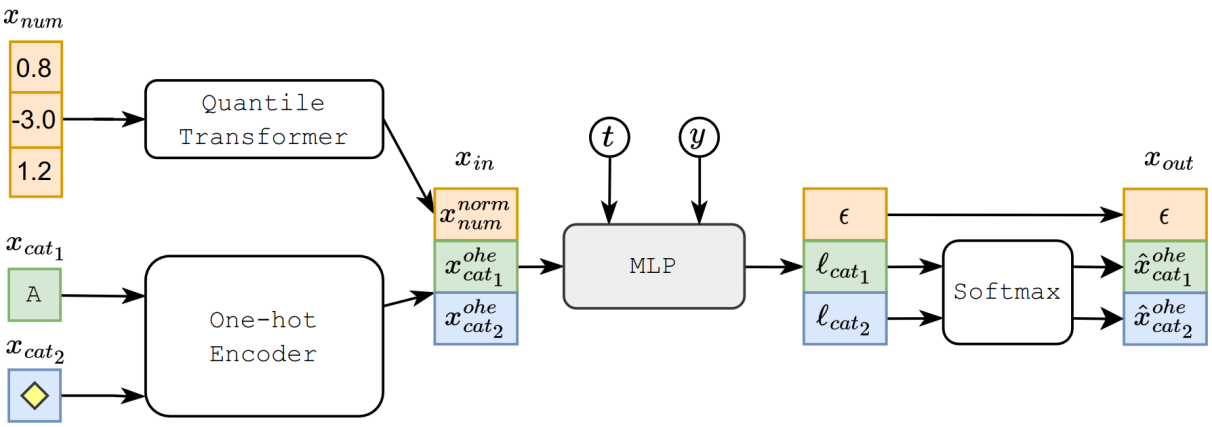

Figura 4.2: Esquema del modelo TabDDPM. Fuente: [135].

así como su uso para sobremuestreo de instancias minoritarias, aspectos para los cuales el algoritmo no fue concebido.

4.2.2. Adaptación del algoritmo

Como base para el desarrollo del nuevo algoritmo, se empleó el código fuente de TabDDPM, disponible en Github [3]. Dicho código incluye no solo el modelo TabDDPM, sino una serie de modelos para comparar su funcionamiento, como SMOTE o adaptaciones tabulares de VAE y GAN. De igual manera, se incluyen implementaciones de clasificadores que permiten medir la bondad de los modelos de sobremuestreo.

Por todo ello, el primer paso fue **eliminar los módulos innecesarios**, como los modelos de sobremuestreo adicionales y los clasificadores, así como aquellos *scripts* dedicados a la ejecución de los mismos. De este modo, el código restante tan solo se correspondía con el entrenamiento del modelo y la generación de nuevas muestras sintéticas.

A partir de ese momento, se debe llevar a cabo una **adaptación del modelo**. Este proceso puede dividirse en dos fases: hacer posible el manejo de datos multietiqueta, y orientar la generación de instancias específicamente a etiquetas minoritarias. Sin embargo, otras tareas como la corrección de la salida y la solución a problemas introducidos por el uso de distribuciones multinomiales también son necesarias.

[3] Código original: https://github.com/yandex-research/tab-ddpm

Adaptación para manejo de MLD

Para conseguir que el modelo pudiera trabajar con datos multietiqueta, el primer paso era **posibilitar la lectura de los MLD**. El formato elegido para los MLD es el usado por el *framework* MULAN [132], consistente en dos ficheros: uno en formato ARFF (*Attribute-relation file format*), en el que se recogen los nombres y tipo de los atributos, así como las instancias; y otro en formato XML (*Extensible markup language*), en el que se recopilan los nombres de las etiquetas.

Una vez se pueden leer ficheros que contienen MLD, es momento de adaptar la clase *Dataset*, que representa un conjunto de datos tradicional, para que sea capaz de representar un conjunto de datos multietiqueta. En el código original, dicha clase contaba con los siguientes atributos:

- X_num y X_cat, dos diccionarios, en los que la clave indica la partición de los datos (entrenamiento, validación y test), y el valor es una matriz de datos. Uno de ellos incluye los atributos numéricos, y el otro, los categóricos.
- y, un diccionario en el que se incluyen los valores de la clase.
- y_info, un diccionario con información sobre la clase.
- $task\_type$, que puede ser clasificación o regresión.
- $n\_classes$, valores que puede tomar la clase.

Cabe destacar la presencia de un atributo para almacenar el **valor de la clase** de cada instancia. A pesar de contar con dichos valores, estos **no son tenidos en cuenta en el entrenamiento** del modelo, sino que se usan para modelar una distribución multinomial "externa" al mismo. En otras palabras, el proceso de adición y eliminación de ruido se lleva a cabo sobre los atributos de entrada, mientras que la clase se asigna de manera mucho más sencilla, simplemente generando un valor aleatorio procedente de una distribución ajustada a los datos de entrenamiento.

La aproximación descrita anteriormente **no parece la más adecuada** por dos motivos. El primero, y más evidente, es que, para MLD, no hay un único valor para la clase, sino varias etiquetas. El segundo motivo es que el método descrito **no aprovecha las posibilidades de los modelos de difusión al completo**, al excluir la clase del proceso. En consecuencia, el nuevo algoritmo incluirá los valores de las etiquetas entre los datos categóricos, desechando el proceso de ajuste de una distribución para la clase.

Para modelar un MLD, se deben reemplazar o adaptar estos atributos, para dar lugar a los siguientes:

- X_num y X_cat. En este caso, no son diccionarios, sino que se trata directamente de dos matrices, pues se considera una única partición. En el caso de los atributos categóricos, como se ha mencionado anteriormente, se incluyen también las etiquetas.
- $numIndexes$, $catIndexes$ y $labelIndexes$, listas con los índices globales de los datos de cada tipo.
- $arffHeader$, cadena con la cabecera del fichero ARFF. De igual manera, el atributo $sparse$ es un valor booleano que indica si el formato de los datos del fichero ARFF es o no disperso.

Además de los mencionados atributos, se han definido diversas funciones. Una de ellas se usa para obtener las etiquetas minoritarias, es decir, aquellas cuyo $IRlbl$ es superior al $MeanIR$ del MLD. Esta puede ser empleada con el fin de obtener aquellas instancias que presentan cada una de estas etiquetas.

En este punto, **el modelo es capaz de generar datos multietiqueta sintéticos que siguen la distribución del MLD original**. No obstante, dicha generación no se centra en las etiquetas minoritarias.

Adaptación para sobremuestreo de etiquetas minoritarias

Una vez el modelo es capaz de generar muestras sintéticas, es necesario adoptar estrategias que permitan obtener instancias en las que estén presentes las etiquetas minoritarias. Se han contemplado tres estrategias distintas, descritas a continuación.

- **Estrategia general:** consiste en entrenar un único modelo de difusión que se ajuste a la **distribución global** del MLD. De este modo, se podrán generar instancias con cualquier *labelset*, incluyendo o no etiquetas minoritarias. Para forzar el sobremuestreo de las etiquetas minoritarias, el proceso de generación de instancias será iterativo, a fin de extraer solo aquellas muestras que contienen etiquetas minoritarias. El proceso se detendrá tras haberse generado una cantidad de instancias minoritarias determinada, o cuando se haya llegado a un máximo de iteraciones. Esta estrategia tiene un inconveniente, y es que **puede ser muy difícil generar instancias de etiquetas con muy pocos ejemplos**, al no quedar estar representadas en el modelo inducido.

- **Estrategia específica:** esta estrategia trata de entrenar **tantos modelos como etiquetas minoritarias** haya en el MLD. Cada modelo debe ser entrenado únicamente con instancias que presenten cada una de esas etiquetas. Para ello, deben emplearse los métodos implementados sobre la clase *Dataset*, que permiten obtener estos "sub-MLD", poblados con instancias determinadas. El objetivo es que cada modelo pueda generar instancias que presenten cada una de las etiquetas minoritarias. El problema de esta estrategia es que **algunas etiquetas minoritarias cuentan con muy pocos ejemplos**, por lo que es probable que los modelos entrenados **no tengan un buen rendimiento**.

- **Estrategia mixta:** esta estrategia tiene como objetivo **combinar las dos anteriores**, con el fin de solucionar los problemas introducidos por las mismas. La idea es entrenar **un único modelo**, en lugar de modelos individuales, con aquellos ejemplos que tengan al menos una etiqueta minoritaria. Es decir, no se consideran las etiquetas minoritarias de forma individual, sino que se agrupan para formar un único MLD, con el que se entrena el modelo. De este modo, las muestras generadas siempre contarán al menos con una etiqueta minoritaria, pero, al haber entrenado el modelo con un volumen de datos mayor, **la calidad de las mismas será, probablemente, superior**.

Por tanto, el proceso de ejecución completa del algoritmo consistirá en entrenar el modelo (o los modelos, según la estrategia), y generar tantas instancias como el usuario desee. El resultado será un fichero ARFF en el que se incluirá el MLD original, con las muestras generadas.

Corrección de la salida

Cabe destacar que la implementación original permite escoger un **método de normalización de los datos**, que se aplica a los mismos en una transformación previa al proceso de entrenamiento, y que se deshace sobre los datos de salida. Esta característica ha sido conservada en el método adaptado, pero es necesario mencionar algunos problemas encontrados tras la realización de una serie de pruebas. Antes, sin embargo, se debe especificar cuáles son los métodos de normalización implementados.

- **Normalización por cuartiles:** este método tiene como fin aproximar la distribución de los datos originales a una normal, lo cual mejora el rendimiento del modelo.

- **Normalización estándar:** se trata del método habitual, restando la media y dividiendo entre la desviación típica.
- **Normalización minmax:** consiste en adaptar el rango de los datos originales a un intervalo concreto; en este caso, [0,1].

No obstante, se observó que la normalización de los datos **no funcionaba acorde con lo esperado**. Al realizar una serie de pruebas, se pudo apreciar que los datos generados no se encontraban en el rango de los valores originales, para cada atributo. Es decir, podía ocurrir que un atributo del conjunto original tuviera como valor máximo el 10, y, sin embargo, una muestra sintética tuviera valor 1000.

Para descartar posibles causas, se llevó a cabo una prueba con el algoritmo original, sin adaptaciones, sobre un conjunto de datos multiclase. El resultado fue, sorprendentemente, **el mismo**. Por este motivo, se decidió adaptar una estrategia para la normalización de la salida del modelo, de modo que el rango de los datos de salida fuera el mismo que el de los datos de entrada. Para ello, se lleva a cabo una **normalización minmax** sobre la salida, con lo que el problema queda solucionado.

Problemas con distribuciones multinomiales

Desde un principio, se pensó que la separación de atributos según su naturaleza llevada a cabo por TabDDPM no es la mejor opción. El principal motivo es que, al modelar dos funciones de probabilidad por separado, **no se contemplan las relaciones entre atributos numéricos y categóricos**. Esto, teniendo en cuenta que las etiquetas son consideradas atributos categóricos, hace que el modelo de difusión entrenado **no sea capaz de respetar la distribución original**. Las primeras pruebas realizadas confirmaron esta hipótesis.

La alternativa a la separación de atributos es el empleo de una codificación *one-hot* para los atributos categóricos (incluyendo las etiquetas). Esta codificación **permite trabajar con atributos discretos como si fueran numéricos**, al representarse cada uno de ellos como un vector, en el que uno de los valores toma el valor uno, y el resto el valor cero. Cada posición del vector representa un valor posible de la variable, y aquella que tiene valor uno es la categoría de la variable.

Diversos artículos [138, 139] han comparado el rendimiento de redes neuronales aportando los datos sin ninguna transformación, y transformando los atributos categóricos con técnicas como la codificación *one-hot*. **Los resultados han decantado la**

balanza a favor de la codificación de los datos categóricos, al propiciar un mejor funcionamiento de los modelos.

Por todos estos motivos, el modelo MLDM, por defecto, lleva a cabo una codificación *one-hot* de los atributos categóricos. Modificando los parámetros del modelo, puede llevarse a cabo la distinción entre atributos original, pero en este trabajo no se tendrá en cuenta esa opción.

4.2.3. Herramientas empleadas

En la actualidad, existe un amplio abanico de posibilidades a la hora de implementar un modelo de red neuronal, como es el caso de los modelos de difusión y, por tanto, del algoritmo que se propone en este trabajo. En la Tabla 4.2 se recogen algunas de las bibliotecas de aprendizaje profundo que permiten construir modelos basados en ANN, así como sus fechas de lanzamiento, si siguen siendo mantenidas, si son *open source software* (OSS) y si ofrecen soporte para CUDA [140], herramienta capaz de mejorar la capacidad computacional, que se explica más adelante.

Herramienta	Lanzamiento	Activo	OSS	CUDA
Wolfram Mathematica	1988	Sí	No	Sí
MATLAB + Deep Learning Toolbox	1992	Sí	No	Sí
Dlib	2002	N/A	Sí	Sí
Torch	2002	No	Sí	Sí
OpenNN	2003	N/A	Sí	Sí
Intel Math Kernel Library	2003	Sí	No	No
Theano	2007	No	Sí	Sí
Neural Designer	2012	N/A	No	No
Caffe	2013	No	Sí	Sí
Deeplearning4j	2014	N/A	Sí	Sí
Intel Data Analytics Acceleration Library	2015	N/A	Sí	No
Apache SINGA	2015	N/A	Sí	Sí
Chainer	2015	No	Sí	Sí
Keras	2015	Sí	Sí	Sí
Apache MXNet	2015	Sí	Sí	Sí
TensorFlow	2015	Sí	Sí	Sí
BigDL	2016	N/A	Sí	No
Microsoft Cognitive Toolkit (CNTK)	2016	No	Sí	Sí
PyTorch	2016	Sí	Sí	Sí
Flux	2017	Sí	Sí	Sí
PlaidML	2017	Sí	Sí	No

Tabla. 4.2: Herramientas para construcción de modelos ANN. Fuente: [141]

Como se puede apreciar, la lista es bastante extensa, especialmente si tenemos en cuenta que existen **aún más herramientas** que no se recogen en la tabla. Por

ese motivo, se debe acotar la búsqueda de acuerdo con los requerimientos de este proyecto.

Teniendo en cuenta que se dispone de una máquina con una tarjeta gráfica Nvidia posterior a la serie G8X para el entrenamiento del modelo, interesa contar con una biblioteca que ofrezca soporte para la arquitectura CUDA. Dicha arquitectura permite la paralelización del código ejecutado en la GPU, lo cual supone una notable disminución del tiempo de entrenamiento, que de otro modo podría llegar a ser muy elevado.

De igual manera, interesa una herramienta que no solo siga en activo, mantenida por el equipo detrás de la misma, sino que, además, sea relativamente reciente, para garantizar su uso en el estado del arte. Los requisitos expuestos permiten acotar las opciones, y por preferencias personales, así como por popularidad a día de hoy, destacan dos candidatos: **TensorFlow y PyTorch**.

TensorFlow

TensorFlow [142] es uno de los *frameworks* más populares para el desarrollo de aplicaciones de aprendizaje profundo. Desarrollado por el equipo de Google Brain, se trata de una biblioteca de software de código abierto para el lenguaje de programación `Python`, que permite construir modelos de ML de manera sencilla.

Una de las características más distintivas de TensorFlow es su capacidad para construir grafos computacionales de alto rendimiento, que representan los cálculos que se realizan en un modelo de ML. Estos grafos computacionales se ejecutan en una amplia variedad de plataformas de hardware, desde CPU hasta GPU y TPU (unidades de procesamiento de tensores, que básicamente son matrices n-dimensionales), lo que permite aprovechar el hardware para acelerar el proceso de entrenamiento y ejecución de modelos de aprendizaje automático.

TensorFlow también cuenta con una gran cantidad de herramientas y bibliotecas adicionales que pueden ser utilizadas para simplificar el proceso de desarrollo de modelos de aprendizaje automático. Por ejemplo, TensorFlow incluye Keras, una biblioteca de alto nivel que permite crear modelos de aprendizaje profundo con una sintaxis simple y concisa. Además, TensorFlow también incluye TensorBoard, una herramienta de visualización para visualizar los grafos de TensorFlow y supervisar el rendimiento de los modelos en tiempo real.

PyTorch

PyTorch [131] es un *framework* de aprendizaje profundo de código abierto desarrollado por Meta Research. Al igual que TensorFlow, PyTorch permite la construcción modelos de ML, gracias a una extensa interfaz con cientos de funciones y clases, desarrollada en `Python`.

A diferencia de TensorFlow, donde los grafos computacionales se construyen de forma estática y luego se ejecutan, en PyTorch los grafos se construyen y ejecutan en tiempo real, a medida que se realiza el entrenamiento del modelo. Esto facilita la experimentación con diferentes arquitecturas de modelo y técnicas de entrenamiento, ya que se puede modificar el grafo computacional en tiempo real; a expensas, eso sí, de un menor rendimiento.

PyTorch también es conocido por su facilidad de uso y flexibilidad. La sintaxis de PyTorch es intuitiva y fácil de entender. Además, cuenta con una gran comunidad de desarrolladores que contribuyen con bibliotecas y herramientas adicionales, siendo, a día de hoy, una comunidad **más activa que la de TensorFlow**.

En resumen, PyTorch y TensorFlow son dos de los frameworks más populares para el desarrollo de aplicaciones de aprendizaje automático y, en concreto, de aprendizaje profundo. No existe una gran diferencia entre ambos: quizás PyTorch sea algo más flexible, y TensorFlow, gracias a Keras, más sencillo. En la Figura 4.3 se puede observar un gráfico que muestra, de manera visual, la evolución de la popularidad de ambos frameworks.

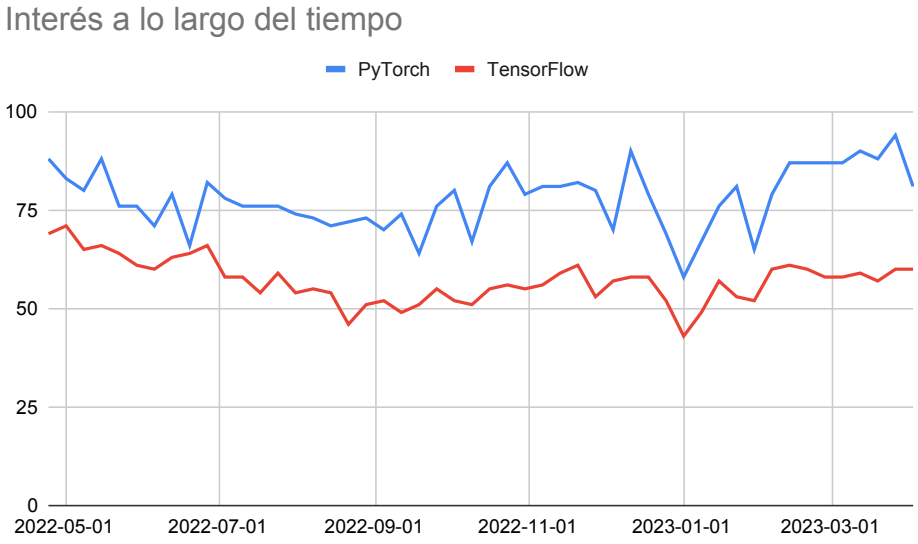

Figura 4.3: Evolución del interés en PyTorch y TensorFlow. Fuente: [143].

Como se puede apreciar, ambos han sido empleados en la misma medida, aunque quizás actualmente PyTorch es ligeramente más popular. En este trabajo, **la implementación del modelo a proponer se llevará a cabo usando PyTorch**. El motivo de esta elección no es tanto el incremento de la popularidad de Pytorch, sino el hecho de que el modelo que va a servir como base para la propuesta de este trabajo fue implementado usando dicho *framework*.

4.3. Experimentación

En este apartado se detalla el proceso de experimentación llevado a cabo para comparar el funcionamiento del algoritmo propuesto con otros algoritmos del estado del arte.

4.3.1. Descripción de la experimentación

La experimentación consistió en la ejecución de una serie de algoritmos de sobremuestreo sobre **distintas particiones** de varios MLD, obteniendo como salida MLD procesados, que pueden compararse entre sí para estimar **cuál es el algoritmo que mejores resultados obtiene**. En el Apéndice A se describe el proceso de instalación de las herramientas necesarias para la ejecución de todos los algoritmos que componen la experimentación.

4.3.2. Algoritmos aplicados

Los algoritmos que se van a ejecutar sobre los distintos conjuntos de datos son, aparte del algoritmo propuesto, aquellos recogidos en la Tabla 4.1 que son de sobremuestreo, con los valores por defecto para todos sus parámetros. En la Tabla 4.3 se recogen tanto los algoritmos como dichos valores por defecto. Estos parámetros, P y k, hacen referencia, respectivamente, al porcentaje en que se incrementa el número de instancias del MLD y al número de vecinos más cercanos a considerar. Nótese que se ha descartado el algoritmo `MLRkNNOS`, debido a su elevada complejidad computacional, que ralentizaba en gran medida la experimentación.

Algoritmo	P	k
LPROS	25	-
MLROS	25	-
MLSMOTE	-	3
MLSOL	25	3
REMEDIAL	-	-
MLDM	25	-

Tabla. 4.3: Algoritmos aplicados en la experimentación y sus valores por defecto

Una vez enumerados los algoritmos a ejecutar, así como los valores que tomarán sus parámetros, se puede especificar sobre qué conjuntos de datos de prueba van a aplicarse.

4.3.3. MLD empleados para la experimentación

En la Tabla 4.4 se recogen los MLD sobre los que se aplicarán los diferentes algoritmos de sobremuestreo multietiqueta, así como sus valores para distintas métricas, tanto de caracterización, como específicas sobre desbalanceo.

Como se puede apreciar, algunos MLD tienen un gran número de etiquetas, llegando incluso a ser mayor que el número de atributos de entrada, algo inconcebible en clasificación tradicional. Esto, sumado a la gran cantidad de instancias que pueden llegar a tener, nos hace ver que los MLD tienen una dimensionalidad muy elevada.

MLD	Ins.	Atrib.	Etiq.	Card	Dens	MeanIR	SCUMBLE	TCS	Ref.
cal500	502	68	174	26.0438	0.1497	20.5778	0.3372	15.5972	[144]
corel5k	5 000	499	374	3.5220	0.0094	189.5676	0.3941	20.1999	[145]
emotions	593	72	6	1.8685	0.3114	1.4781	0.0110	9.3643	[146]
genbase	662	1 186	27	1.2523	0.0464	37.3146	0.0288	13.8399	[147]
medical	978	1 449	45	1.2454	0.0277	89.5014	0.0471	15.6286	[148]
scene	2 407	294	6	1.0740	0.1790	1.2538	0.0003	10.1834	[149]
chess [a]	1 675	585	227	2.4113	0.0106	85.7898	0.2625	18.7794	[150]
yeast	2 417	103	14	4.2371	0.3026	7.1968	0.1044	12.5621	[151]

[a] Su nombre completo es stackex_chess, extraído del subforo de ajedrez la web *Stack Exchange*

Tabla. 4.4: Métricas de desbalanceo para distintos MLD. Fuente: [128]

Con el fin de tener en cuenta la hipótesis de generalización en la posterior evaluación de los MLD resultantes, los algoritmos se aplicaron sobre particiones de los conjuntos de datos recogidos en la Tabla 4.4, llevando a cabo un proceso de **validación cruzada con 5 particiones**. Dichas particiones están disponibles en el repositorio Cometa [128]. En concreto, se trata de las cinco primeras particiones estratificadas de tipo *2x5-fold cross validation*, que pueden descargarse en distintos formatos.

4.3.4. Características del equipo

Los experimentos se han ejecutado sobre uno de los **servidores de cómputo** del grupo de investigación SimiDat de la Universidad de Jaén, cuyas características se recogen en la Tabla 4.5.

Componente	**Modelo**
Sistema Operativo	Debian GNU/Linux
Procesador	Intel(R) Xeon(R) Silver 4210R CPU
Memoria RAM	256 GB
Disco duro	SSD 256 GB + HDD 9 TB
Tarjeta Gráfica	Nvidia A100

Tabla. 4.5: Características del equipo en el que se ha llevado a cabo la experimentación

Como extensión a los datos recogidos en la Tabla 4.5, cabe mencionar que la máquina cuenta con dos *sockets* del procesador Intel(R) Xeon(R) Silver 4210R CPU, cada uno de los cuales cuenta con diez núcleos, capaces de ejecutar dos hilos en paralelo cada uno de ellos. De este modo, se consiguen un total de **40 hilos**, que hacen posible la ejecución de 40 subprocesos paralelos.

En cuanto a la tarjeta gráfica, se trata de una GPU con 80 GB de memoria, que cuenta con tecnología Tensor Core, la cual **mejora notablemente el rendimiento en tareas de inteligencia artificial y análisis de datos**, al trabajar de manera muy eficiente con tensores. Cuenta con 6 912 núcleos CUDA FP32, 3 456 núcleos CUDA FP64 y 422 Tensor Cores. Se trata, en concreto, del modelo PCIe, pues la máquina solo cuenta con una GPU, por lo que la versión SXM no supone una gran ventaja en cuanto a la velocidad de transferencia.

Capítulo 5

RESULTADOS

En este capítulo se recopilan los resultados obtenidos en la experimentación, los cuales permiten comparar el rendimiento de los diferentes algoritmos aplicados, y así determinar si el modelo propuesto en este trabajo es o no competente en su ámbito de aplicación.

Primero, se enumeran los clasificadores que se ejecutarán y las métricas con las que se estimará su rendimiento, a fin de comparar algoritmos de remuestreo. Posteriormente, se lleva a cabo un análisis de variantes, en el que se determina cuál es la mejor versión de MLDM, de acuerdo con la estrategia de sobremuestreo y el tipo de normalización de los datos numéricos. Tras ellos, se compara la mejor variante de MLDM con otros algoritmos de sobremuestreo multietiqueta del estado del arte, teniendo en cuenta el nivel de desblanceo alcanzado tras la aplicacion de los algoritmos, el rendimiento de clasificadores y el tiempo de ejecución. Por último, se incluye un análisis de los resultados obtenidos.

5.1. Clasificadores y métricas evaluados

La manera de saber, a ciencia cierta, si el rendimiento de los clasificadores aumenta al llevar a cabo un sobremuestreo sobre el MLD original, es precisamente **probar dichos clasificadores antes y después del sobremuestreo**. De esta manera, si la clasificación mejora, se asume que el método es beneficioso.

En este apartado se enumeran los clasificadores que se han ejecutado sobre los datos, así como las métricas empleadas para estimar el rendimiento de los mismos.

5.1.1. Clasificadores empleados

Para esta comparación se han empleado distintos algoritmos de MLC del estado del arte, de entre los muchos que hay propuestos en diversos artículos y congresos. Los cinco algoritmos seleccionados han sido los siguientes:

- **BPMLL:** se trata de una red neuronal diseñada para datos multietiqueta [152]. La arquitectura es similar a la de un MLP sencillo y su principal característica es que la capa de salida cuenta con tantas neuronas como etiquetas quieran predecirse.

- **BR-J48:** este algoritmo consiste en aplicar una transformación BR al MLD, para así obtener tantos conjuntos de datos binarios como etiquetas a predecir. Posteriormente, se emplean árboles de decisión J48 para predecir, por separado, cada una de las etiquetas.

- **HOMER:** se trata de un *ensemble*: un conjunto de clasificadores que llevan a cabo una predicción cada uno, agregándose sus salidas para dar lugar a la clasificación final. En el caso de HOMER [153], el MLD se divide en subconjuntos con menos etiquetas que el original, siendo cada uno de ellos tratado por un clasificador distinto para obtener un *labelset*. La agregación de todos los *labelsets* constituye la salida del modelo.

- **LP-J48:** al igual que el BR-J48, se lleva a cabo una transformación sobre el MLD para poder aplicar un árbol de decisión tradicional. En este caso, la transformación es LP.

- **MLkNN:** se trata de una adaptación del algoritmo kNN, en la cual, en lugar de predecirse el valor de una sola clase, se predicen las etiquetas de una nueva instancia en función de las de sus vecinos más cercanos [154].

Este conjunto de clasificadores ha sido elegido con el objetivo de representar el amplio espectro de enfoques dentro de la clasificación multietiqueta. Por ese motivo, se han seleccionado una red neuronal, dos métodos basados en transformaciones del MLD (BR y LP), que en este caso son árboles, un algoritmo basado en búsqueda de vecinos y un *ensemble*.

5.1.2. Métricas empleadas

La bondad de una tarea de MLC puede medirse empleando distintas métricas, algunas de las cuales se han repasado en el apartado 2.3.2. A raíz de la experimentación llevada a cabo, se han generado 18 métricas, de las cuales se han seleccionado las siguientes cinco.

- **F-Measure basada en ejemplos:** se trata de la mencionada métrica $F\text{-}Measure$, cuya fórmula se recoge en la ecuación 2.6, agregada por instancias; es decir, calculando el valor de la métrica para cada una de las instancias del conjunto de datos de test, y agregando dichos valores. Es una métrica a maximizar.
- **Hamming-Loss:** explicada anteriormente, y descrita en la Ecuación 2.9, se trata de una métrica que mide el número de diferencias entre etiquetas predichas y reales. Se debe minimizar su valor.
- **Macro F-Measure:** consiste en la agregación por etiquetas tipo macro de la métrica $F\text{-}Measure$. Dicha agregación se lleva a cabo según la fórmula recogida en la Ecuación 2.7.
- **Micro F-Measure:** exactamente igual que la métrica anterior, solo que la agregación es de tipo micro, calculada siguiendo la Ecuación 2.8.
- **One-Error:** se trata de una métrica de ranking, cuya fórmula se recoge en la Ecuación 2.12, que computa el número de veces que la primera etiqueta del ranking pertenece al *labelset* real.

Al igual que los clasificadores, las métricas se han seleccionado con el fin de obtener un conjunto representativo, teniendo en cuenta los diferentes tipos de métricas de evaluación de MLC. De esta manera, se han incluido métricas basadas en instancias y basadas en etiquetas (con ambos tipos de agregación); así como métricas de bipartición y de ranking.

5.2. Análisis de variantes

Como se ha indicado en el apartado 4.2.2, el algoritmo MLDM admite diversos parámetros, entre ellos la estrategia y el tipo de normalización, los cuales dan lugar a distintas variantes, recogidas en la Tabla 5.1.

Variante	Estrategia	Normalización
MLDM-SS	Específica	Estándar
MLDM-SM	Específica	Minmax
MLDM-SQ	Específica	Por cuantiles
MLDM-MS	Mixta	Estándar
MLDM-MM	Mixta	Minmax
MLDM-MQ	Mixta	Por cuantiles

Tabla. 5.1: Variantes de MLDM incluidas en la experimentación

El motivo por el que no se ha incluido la estrategia general en la experimentación es que, debido a que la mayoría de los MLD que componen las pruebas presentan un muy alto nivel de desbalanceo (algo propio de los MLD), al generarse nuevas instancias que siguen la distribución global, es muy difícil dar con nuevas instancias que presenten etiquetas minoritarias, de modo que el algoritmo es demasiado lento, generando muy pocas muestras minoritarias en demasiado tiempo.

Una vez se han presentado las variantes del algoritmo que han sido ejecutadas durante la experimentación, se pueden comparar entre ellas, a fin de obtener una visión general acerca de cuál funciona mejor, en general. En las siguientes tablas se recogen los resultados para cada una de las métricas y clasificadores enumerados en el apartado 5.1, tras aplicar cada una de las variantes al MLD original. En negrita se destaca el mejor valor obtenido para una métrica. En caso de igualdad en el valor, se premiará una menor desviación.

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.452±0.010	0.347±0.016	**0.395±0.011**	**0.328±0.014**	0.320±0.010
	MLDM-SM	**0.467±0.019**	0.336±0.023	0.387±0.019	0.322±0.012	**0.326±0.012**
	MLDM-SS	0.465±0.013	0.346±0.016	0.393±0.006	0.323±0.015	0.323±0.017
	MLDM-SQ	0.464±0.016	**0.350±0.012**	0.384±0.015	0.319±0.011	0.321±0.011
	MLDM-MM	0.464±0.015	0.336±0.023	0.391±0.014	0.322±0.012	**0.326±0.012**
	MLDM-MS	0.463±0.010	0.346±0.016	0.384±0.011	0.323±0.015	0.323±0.017
	MLDM-MQ	0.466±0.012	**0.350±0.012**	0.387±0.014	0.319±0.011	0.321±0.011
corel5k		0.024±0.001	**0.088±0.008**	0.149±0.010	0.102±0.010	0.016±0.003
	MLDM-SM	**0.025±0.001**	0.085±0.009	0.148±0.011	**0.108±0.008**	**0.017±0.002**
	MLDM-SS	0.024±0.001	0.085±0.009	**0.162±0.016**	**0.108±0.008**	**0.017±0.002**
	MLDM-SQ	0.024±0.001	0.085±0.009	0.161±0.010	**0.108±0.008**	**0.017±0.002**
	MLDM-MM	0.024±0.001	0.085±0.009	0.156±0.007	**0.108±0.008**	**0.017±0.002**
	MLDM-MS	0.024±0.000	0.085±0.009	0.148±0.007	**0.108±0.008**	**0.017±0.002**
	MLDM-MQ	0.024±0.001	0.085±0.009	0.161±0.008	**0.108±0.008**	**0.017±0.002**
emotions		0.654±0.013	**0.552±0.021**	0.546±0.026	0.534±0.041	0.629±0.014
	MLDM-SM	0.661±0.038	0.548±0.026	0.556±0.023	0.527±0.025	**0.639±0.013**
	MLDM-SS	0.650±0.023	0.537±0.031	0.517±0.031	0.532±0.041	0.622±0.011
	MLDM-SQ	**0.667±0.018**	0.511±0.037	0.526±0.015	**0.542±0.046**	0.619±0.022

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLDM-MM	0.654±0.028	0.548±0.026	0.545±0.034	0.527±0.025	**0.639±0.013**
	MLDM-MS	0.651±0.015	0.537±0.031	**0.560±0.033**	0.532±0.041	0.622±0.011
	MLDM-MQ	0.665±0.021	0.511±0.037	0.534±0.024	**0.542±0.046**	0.619±0.022
medical		0.053±0.031	0.771±0.021	**0.784±0.018**	0.765±0.039	0.585±0.016
	MLDM-SM	**0.059±0.005**	**0.772±0.013**	0.772±0.013	**0.766±0.041**	**0.587±0.017**
	MLDM-SS	0.057±0.004	**0.772±0.013**	0.771±0.018	**0.766±0.041**	**0.587±0.017**
	MLDM-SQ	0.057±0.006	**0.772±0.013**	0.767±0.019	**0.766±0.041**	**0.587±0.017**
	MLDM-MM	0.024±0.033	**0.772±0.013**	0.781±0.016	**0.766±0.041**	**0.587±0.017**
	MLDM-MS	0.047±0.026	**0.772±0.013**	0.775±0.026	**0.766±0.041**	**0.587±0.017**
	MLDM-MQ	0.046±0.026	**0.772±0.013**	0.767±0.036	**0.766±0.041**	**0.587±0.017**
scene		0.472±0.042	**0.580±0.019**	0.551±0.024	**0.588±0.034**	0.679±0.016
	MLDM-SM	0.488±0.042	0.545±0.023	0.543±0.019	0.583±0.030	0.663±0.035
	MLDM-SS	0.465±0.076	0.570±0.023	0.545±0.019	0.586±0.036	**0.688±0.026**
	MLDM-SQ	0.434±0.059	0.566±0.028	**0.561±0.026**	0.582±0.036	0.659±0.042
	MLDM-MM	**0.518±0.057**	0.545±0.023	0.548±0.020	0.583±0.030	0.663±0.035
	MLDM-MS	0.477±0.069	0.570±0.023	0.558±0.016	0.586±0.036	**0.688±0.026**
	MLDM-MQ	0.367±0.083	0.566±0.028	0.555±0.033	0.582±0.036	0.659±0.042
chess		0.016±0.002	0.249±0.017	0.247±0.015	0.139±0.008	0.046±0.018
	MLDM-SM	0.016±0.001	0.254±0.019	0.260±0.016	**0.154±0.011**	0.052±0.022
	MLDM-SS	0.017±0.001	0.256±0.022	**0.270±0.012**	0.150±0.016	0.052±0.015
	MLDM-SQ	**0.018±0.002**	**0.261±0.022**	0.266±0.026	0.146±0.016	**0.054±0.018**
	MLDM-MM	0.017±0.002	0.254±0.019	0.258±0.010	**0.154±0.011**	0.052±0.022
	MLDM-MS	0.017±0.002	0.256±0.022	0.252±0.021	0.150±0.016	0.052±0.015
	MLDM-MQ	0.018±0.001	**0.261±0.022**	0.262±0.012	0.146±0.016	**0.054±0.018**
yeast		0.623±0.023	0.554±0.020	0.552±0.032	0.506±0.018	0.614±0.029
	MLDM-SM	**0.635±0.028**	**0.565±0.016**	0.551±0.030	0.504±0.007	**0.616±0.031**
	MLDM-SS	0.633±0.027	0.552±0.015	0.547±0.014	0.500±0.007	0.615±0.030
	MLDM-SQ	0.630±0.025	0.546±0.026	0.554±0.019	**0.510±0.019**	0.615±0.031
	MLDM-MM	0.632±0.029	**0.565±0.016**	**0.562±0.027**	0.504±0.007	**0.616±0.031**
	MLDM-MS	0.627±0.022	0.552±0.015	0.553±0.018	0.500±0.007	0.615±0.030
	MLDM-MQ	0.635±0.020	0.546±0.026	0.558±0.024	**0.510±0.019**	0.615±0.031

Tabla. 5.2: Valores para la métrica F-Measure basada en ejemplos para variantes de MLDM

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.213±0.009	**0.148±0.015**	0.173±0.007	0.156±0.011	0.089±0.012
	MLDM-SM	**0.234±0.009**	0.133±0.012	0.159±0.008	**0.159±0.005**	**0.091±0.015**
	MLDM-SS	0.217±0.015	0.137±0.011	**0.174±0.010**	0.154±0.009	0.091±0.019
	MLDM-SQ	0.220±0.021	0.142±0.006	0.159±0.008	0.157±0.006	0.090±0.019
	MLDM-MM	0.227±0.014	0.133±0.012	0.161±0.008	**0.159±0.005**	**0.091±0.015**
	MLDM-MS	0.222±0.011	0.137±0.011	0.157±0.006	0.154±0.009	0.091±0.019
	MLDM-MQ	0.228±0.017	0.142±0.006	0.165±0.012	0.157±0.006	0.090±0.019

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
corel5k		0.139±0.020	**0.196±0.014**	0.193±0.016	**0.121±0.014**	**0.185±0.016**
	MLDM-SM	0.140±0.010	0.185±0.020	0.190±0.015	0.116±0.022	0.182±0.015
	MLDM-SS	0.134±0.016	0.185±0.020	0.187±0.013	0.116±0.022	0.182±0.015
	MLDM-SQ	0.131±0.012	0.185±0.020	**0.194±0.014**	0.116±0.022	0.182±0.015
	MLDM-MM	**0.141±0.010**	0.185±0.020	0.184±0.013	0.116±0.022	0.182±0.015
	MLDM-MS	0.140±0.010	0.185±0.020	0.189±0.014	0.116±0.022	0.182±0.015
	MLDM-MQ	0.134±0.013	0.185±0.020	0.190±0.017	0.116±0.022	0.182±0.015
emotions		0.674±0.010	**0.588±0.006**	0.585±0.018	0.556±0.033	0.642±0.025
	MLDM-SM	0.671±0.035	0.585±0.015	0.588±0.020	0.551±0.022	**0.653±0.020**
	MLDM-SS	0.667±0.019	0.584±0.028	0.562±0.027	0.555±0.032	0.646±0.022
	MLDM-SQ	0.675±0.017	0.547±0.040	0.563±0.021	**0.557±0.037**	0.643±0.026
	MLDM-MM	0.665±0.032	0.585±0.015	0.575±0.028	0.551±0.022	**0.653±0.020**
	MLDM-MS	0.670±0.009	0.584±0.028	**0.589±0.027**	0.555±0.032	0.646±0.022
	MLDM-MQ	**0.680±0.031**	0.547±0.040	0.566±0.017	**0.557±0.037**	0.643±0.026
medical		0.226±0.129	**0.661±0.061**	**0.636±0.048**	**0.598±0.047**	**0.531±0.075**
	MLDM-SM	0.098±0.051	0.634±0.067	0.627±0.073	0.581±0.086	0.527±0.072
	MLDM-SS	0.102±0.033	0.634±0.067	0.612±0.069	0.581±0.086	0.527±0.072
	MLDM-SQ	0.102±0.040	0.634±0.067	0.591±0.029	0.581±0.086	0.527±0.072
	MLDM-MM	**0.242±0.134**	0.634±0.067	0.618±0.039	0.581±0.086	0.527±0.072
	MLDM-MS	0.106±0.071	0.634±0.067	0.623±0.047	0.581±0.086	0.527±0.072
	MLDM-MQ	0.141±0.088	0.634±0.067	0.603±0.051	0.581±0.086	0.527±0.072
scene		0.545±0.022	**0.636±0.013**	0.612±0.013	**0.592±0.033**	0.734±0.011
	MLDM-SM	0.543±0.030	0.612±0.015	0.604±0.018	0.587±0.027	0.724±0.025
	MLDM-SS	0.520±0.049	0.635±0.017	0.608±0.013	0.588±0.034	**0.737±0.015**
	MLDM-SQ	0.528±0.023	0.631±0.022	**0.620±0.020**	0.585±0.034	0.723±0.027
	MLDM-MM	**0.569±0.035**	0.612±0.015	0.603±0.016	0.587±0.027	0.724±0.025
	MLDM-MS	0.536±0.046	0.635±0.017	0.615±0.017	0.588±0.034	**0.737±0.015**
	MLDM-MQ	0.481±0.055	0.631±0.022	0.617±0.030	0.585±0.034	0.723±0.027
chess		0.021±0.004	0.325±0.023	0.275±0.018	0.174±0.018	**0.277±0.020**
	MLDM-SM	0.021±0.004	0.288±0.029	0.274±0.032	0.172±0.030	0.274±0.025
	MLDM-SS	0.020±0.002	0.317±0.029	**0.286±0.025**	**0.182±0.047**	0.274±0.019
	MLDM-SQ	**0.024±0.008**	**0.326±0.025**	0.270±0.036	0.139±0.042	0.274±0.023
	MLDM-MM	0.022±0.006	0.288±0.029	0.269±0.025	0.172±0.030	0.274±0.025
	MLDM-MS	0.022±0.004	0.317±0.029	0.282±0.033	**0.182±0.047**	0.274±0.019
	MLDM-MQ	0.023±0.006	**0.326±0.025**	0.274±0.017	0.139±0.042	0.274±0.023
yeast		0.438±0.015	0.381±0.008	0.398±0.013	0.374±0.013	0.370±0.004
	MLDM-SM	0.443±0.008	0.387±0.008	0.394±0.019	0.370±0.013	**0.381±0.011**
	MLDM-SS	0.435±0.013	**0.387±0.005**	0.387±0.007	**0.375±0.018**	0.375±0.004
	MLDM-SQ	0.439±0.009	0.384±0.014	0.402±0.009	0.374±0.004	0.376±0.009
	MLDM-MM	0.443±0.012	0.387±0.008	**0.402±0.008**	0.370±0.013	**0.381±0.011**
	MLDM-MS	0.437±0.011	**0.387±0.005**	0.397±0.012	**0.375±0.018**	0.375±0.004
	MLDM-MQ	**0.452±0.009**	0.384±0.014	0.392±0.016	0.374±0.004	0.376±0.009

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN

Tabla. 5.3: Valores para la métrica Macro F-Measure para variantes de MLDM

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.456±0.014	0.348±0.016	**0.398±0.011**	**0.332±0.015**	0.316±0.010
	MLDM-SM	**0.472±0.015**	0.337±0.024	0.389±0.019	0.320±0.014	**0.322±0.013**
	MLDM-SS	0.467±0.014	0.346±0.017	0.395±0.006	0.321±0.015	0.319±0.017
	MLDM-SQ	0.467±0.016	**0.350±0.012**	0.386±0.015	0.315±0.013	0.317±0.013
	MLDM-MM	0.468±0.014	0.337±0.024	0.393±0.015	0.320±0.014	**0.322±0.013**
	MLDM-MS	0.470±0.012	0.346±0.017	0.386±0.011	0.321±0.015	0.319±0.017
	MLDM-MQ	0.470±0.008	**0.350±0.012**	0.389±0.015	0.315±0.013	0.317±0.013
corel5k		0.026±0.001	**0.113±0.011**	0.165±0.010	0.104±0.009	0.027±0.005
	MLDM-SM	0.027±0.001	0.110±0.011	0.165±0.010	**0.109±0.008**	**0.027±0.004**
	MLDM-SS	0.027±0.001	0.110±0.011	0.176±0.016	**0.109±0.008**	**0.027±0.004**
	MLDM-SQ	0.026±0.000	0.110±0.011	**0.178±0.010**	**0.109±0.008**	**0.027±0.004**
	MLDM-MM	0.026±0.000	0.110±0.011	0.172±0.007	**0.109±0.008**	**0.027±0.004**
	MLDM-MS	0.026±0.000	0.110±0.011	0.165±0.006	**0.109±0.008**	**0.027±0.004**
	MLDM-MQ	**0.027±0.000**	0.110±0.011	0.177±0.007	**0.109±0.008**	**0.027±0.004**
emotions		0.683±0.012	**0.600±0.004**	0.599±0.012	0.564±0.032	0.676±0.016
	MLDM-SM	0.683±0.034	0.592±0.013	0.598±0.018	0.561±0.020	**0.680±0.010**
	MLDM-SS	0.675±0.021	0.593±0.030	0.570±0.028	0.564±0.035	0.674±0.013
	MLDM-SQ	0.688±0.016	0.562±0.033	0.574±0.012	**0.568±0.039**	0.669±0.020
	MLDM-MM	0.677±0.026	0.592±0.013	0.583±0.024	0.561±0.020	**0.680±0.010**
	MLDM-MS	0.679±0.006	0.593±0.030	**0.600±0.030**	0.564±0.035	0.674±0.013
	MLDM-MQ	**0.690±0.031**	0.562±0.033	0.576±0.018	**0.568±0.039**	0.669±0.020
medical		0.053±0.031	**0.807±0.012**	**0.804±0.012**	**0.760±0.034**	**0.665±0.017**
	MLDM-SM	0.059±0.005	0.803±0.011	0.796±0.014	0.755±0.035	**0.665±0.017**
	MLDM-SS	0.059±0.004	0.803±0.011	0.791±0.019	0.755±0.035	**0.665±0.017**
	MLDM-SQ	**0.060±0.004**	0.803±0.011	0.792±0.013	0.755±0.035	**0.665±0.017**
	MLDM-MM	0.024±0.033	0.803±0.011	**0.804±0.012**	0.755±0.035	**0.665±0.017**
	MLDM-MS	0.047±0.026	0.803±0.011	0.795±0.018	0.755±0.035	**0.665±0.017**
	MLDM-MQ	0.048±0.027	0.803±0.011	0.791±0.027	0.755±0.035	**0.665±0.017**
scene		0.528±0.033	**0.628±0.012**	0.603±0.016	**0.583±0.035**	0.730±0.012
	MLDM-SM	0.521±0.032	0.599±0.016	0.593±0.017	0.578±0.029	0.721±0.025
	MLDM-SS	0.498±0.041	0.625±0.019	0.597±0.014	0.580±0.036	**0.731±0.013**
	MLDM-SQ	0.512±0.018	0.620±0.025	**0.610±0.022**	0.577±0.035	0.719±0.028
	MLDM-MM	**0.544±0.041**	0.599±0.016	0.592±0.017	0.578±0.029	0.721±0.025
	MLDM-MS	0.511±0.050	0.625±0.019	0.605±0.017	0.580±0.036	**0.731±0.013**
	MLDM-MQ	0.457±0.057	0.620±0.025	0.608±0.030	0.577±0.035	0.719±0.028
chess		**0.023±0.000**	0.297±0.017	0.276±0.015	0.145±0.011	0.063±0.024
	MLDM-SM	**0.023±0.000**	0.292±0.021	0.287±0.020	**0.156±0.015**	0.071±0.032
	MLDM-SS	0.023±0.001	**0.300±0.020**	**0.300±0.009**	0.156±0.017	0.073±0.022

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLDM-SQ	0.023±0.001	**0.308±0.022**	0.293±0.027	0.147±0.012	**0.076±0.029**
	MLDM-MM	0.023±0.001	0.292±0.021	0.286±0.011	**0.156±0.015**	0.071±0.032
	MLDM-MS	0.023±0.001	0.300±0.020	0.282±0.017	0.156±0.017	0.073±0.022
	MLDM-MQ	0.023±0.001	**0.308±0.022**	0.294±0.009	0.147±0.012	**0.076±0.029**
yeast		0.641±0.018	0.576±0.017	0.577±0.027	0.530±0.016	0.640±0.023
	MLDM-SM	0.650±0.025	**0.587±0.014**	0.576±0.025	0.529±0.005	**0.642±0.026**
	MLDM-SS	0.649±0.022	0.577±0.015	0.570±0.010	0.527±0.010	0.642±0.024
	MLDM-SQ	0.646±0.021	0.569±0.022	0.580±0.016	**0.532±0.013**	0.641±0.024
	MLDM-MM	0.647±0.024	**0.587±0.014**	**0.583±0.024**	0.529±0.005	**0.642±0.026**
	MLDM-MS	0.644±0.018	0.577±0.015	0.578±0.015	0.527±0.010	0.642±0.024
	MLDM-MQ	**0.652±0.016**	0.569±0.022	0.579±0.021	**0.532±0.013**	0.641±0.024

Tabla. 5.4: Valores para la métrica Micro F-Measure para variantes de MLDM

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.135±0.051	0.691±0.089	0.806±0.065	**0.988±0.008**	0.117±0.040
	MLDM-SM	0.144±0.041	0.731±0.045	0.786±0.066	**0.988±0.008**	0.129±0.047
	MLDM-SS	0.144±0.042	**0.766±0.046**	0.843±0.043	**0.988±0.008**	0.129±0.052
	MLDM-SQ	**0.156±0.048**	0.735±0.035	0.830±0.025	**0.988±0.008**	**0.132±0.054**
	MLDM-MM	0.146±0.048	0.731±0.045	**0.857±0.039**	**0.988±0.008**	0.129±0.047
	MLDM-MS	0.146±0.038	**0.766±0.046**	0.842±0.037	**0.988±0.008**	0.129±0.052
	MLDM-MQ	0.140±0.032	0.735±0.035	0.820±0.042	**0.988±0.008**	**0.132±0.054**
corel5k		0.996±0.003	0.711±0.016	0.797±0.018	**0.988±0.003**	0.741±0.010
	MLDM-SM	0.994±0.004	**0.717±0.013**	**0.807±0.011**	0.984±0.007	**0.747±0.010**
	MLDM-SS	0.997±0.002	**0.717±0.013**	0.792±0.016	0.984±0.007	**0.747±0.010**
	MLDM-SQ	0.997±0.004	**0.717±0.013**	0.778±0.017	0.984±0.007	**0.747±0.010**
	MLDM-MM	0.996±0.004	**0.717±0.013**	0.799±0.018	0.984±0.007	**0.747±0.010**
	MLDM-MS	**0.999±0.001**	**0.717±0.013**	0.803±0.013	0.984±0.007	**0.747±0.010**
	MLDM-MQ	0.996±0.004	**0.717±0.013**	0.799±0.018	0.984±0.007	**0.747±0.010**
emotions		0.297±0.041	0.395±0.042	0.403±0.032	0.445±0.034	0.245±0.038
	MLDM-SM	0.300±0.054	0.422±0.028	0.426±0.030	**0.450±0.028**	0.250±0.027
	MLDM-SS	0.289±0.045	0.414±0.045	**0.463±0.056**	0.440±0.038	**0.251±0.015**
	MLDM-SQ	0.287±0.026	**0.427±0.026**	0.450±0.064	0.442±0.068	0.240±0.016
	MLDM-MM	**0.320±0.052**	0.422±0.028	0.430±0.028	**0.450±0.028**	0.250±0.027
	MLDM-MS	0.294±0.028	0.414±0.045	0.397±0.043	0.440±0.038	**0.251±0.015**
	MLDM-MQ	0.320±0.053	**0.427±0.026**	0.429±0.036	0.442±0.068	0.240±0.016
medical		**0.976±0.020**	0.183±0.022	0.202±0.018	**0.231±0.035**	0.241±0.032
	MLDM-SM	0.968±0.019	**0.185±0.027**	0.226±0.019	0.230±0.035	**0.243±0.031**
	MLDM-SS	0.973±0.024	**0.185±0.027**	0.222±0.015	0.230±0.035	**0.243±0.031**
	MLDM-SQ	0.970±0.024	**0.185±0.027**	0.210±0.029	0.230±0.035	**0.243±0.031**
	MLDM-MM	0.969±0.018	**0.185±0.027**	0.204±0.024	0.230±0.035	**0.243±0.031**
	MLDM-MS	0.974±0.024	**0.185±0.027**	0.222±0.026	0.230±0.035	**0.243±0.031**

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLDM-MQ	0.965±0.016	**0.185±0.027**	**0.234±0.033**	0.230±0.035	**0.243±0.031**
scene		0.514±0.126	0.406±0.020	0.435±0.016	0.414±0.029	0.229±0.021
	MLDM-SM	0.469±0.077	**0.438±0.024**	0.438±0.015	0.424±0.022	0.229±0.020
	MLDM-SS	0.549±0.075	0.423±0.024	0.451±0.017	0.415±0.033	0.229±0.022
	MLDM-SQ	0.538±0.048	0.408±0.036	0.439±0.020	**0.426±0.033**	**0.238±0.029**
	MLDM-MM	0.495±0.140	**0.438±0.024**	**0.447±0.019**	0.424±0.022	0.229±0.020
	MLDM-MS	0.529±0.100	0.423±0.024	0.429±0.026	0.415±0.033	0.229±0.022
	MLDM-MQ	**0.611±0.092**	0.408±0.036	0.428±0.032	**0.426±0.033**	**0.238±0.029**
chess		0.996±0.005	0.553±0.031	**0.702±0.031**	**0.954±0.021**	0.696±0.038
	MLDM-SM	0.992±0.005	**0.568±0.019**	0.674±0.026	0.940±0.014	**0.704±0.033**
	MLDM-SS	0.998±0.004	0.564±0.030	0.663±0.029	0.947±0.011	0.699±0.036
	MLDM-SQ	0.998±0.002	0.554±0.016	0.669±0.039	0.946±0.012	0.698±0.037
	MLDM-MM	0.998±0.002	**0.568±0.019**	0.674±0.019	0.940±0.014	**0.704±0.033**
	MLDM-MS	0.997±0.002	0.564±0.030	0.684±0.031	0.947±0.011	0.699±0.036
	MLDM-MQ	**0.999±0.002**	0.554±0.016	0.683±0.025	0.946±0.012	0.698±0.037
yeast		**0.253±0.010**	**0.425±0.068**	0.410±0.058	**0.538±0.031**	**0.229±0.026**
	MLDM-SM	0.252±0.023	0.377±0.035	0.403±0.063	0.507±0.015	0.227±0.027
	MLDM-SS	0.250±0.018	0.408±0.058	0.413±0.050	0.514±0.021	0.227±0.024
	MLDM-SQ	0.247±0.025	0.383±0.056	0.430±0.044	0.525±0.025	**0.229±0.026**
	MLDM-MM	0.248±0.021	0.377±0.035	**0.454±0.053**	0.507±0.015	0.227±0.027
	MLDM-MS	0.241±0.024	0.408±0.058	0.391±0.034	0.514±0.021	0.227±0.024
	MLDM-MQ	0.248±0.022	0.383±0.056	0.396±0.052	0.525±0.025	**0.229±0.026**

Tabla. 5.5: Valores para la métrica One-Error para variantes de MLDM

Una vez se han recopilado los resultados obtenidos, se puede llevar a cabo un conteo del número de veces que cada variante queda en primer lugar con respecto a las demás. Este sencillo enfoque permite determinar cuál es, en promedio, la mejor variante y, por tanto, la que se comparará con el resto de algoritmos. En la Tabla 5.6 se recoge el mencionado conteo.

Variante	Número de veces que es la mejor
MLDM-SS	27
MLDM-SM	39
MLDM-SQ	38
MLDM-MS	24
MLDM-MM	45
MLDM-MQ	35

Tabla. 5.6: Conteo del número de veces que cada variante es la mejor

A la vista de la Tabla 5.6, se puede concluir que la variante MLDM-MM, que sigue la estrategia mixta y lleva a cabo una normalización minmax, es la que más veces

supera a las demás y, por tanto, **es la que se empleará a partir de este punto**, haciendo referencia a la misma únicamente como MLDM.

5.3. Impacto del sobremuestreo en el nivel de desbalanceo

Como se ha explicado en el apartado 2.4.2, el desbalanceo en un MLD puede ser descrito por una serie de métricas. En la Tabla 5.7 se recogen los valores de algunas de esas métricas para cada MLD antes y después de aplicar cada uno de los algoritmos. De nuevo, se ha calculado la media de dichos valores para los MLD generados por cada uno de los algoritmos que conforman la experimentación.

MLD	Algoritmo	Ins.	MeanIR	SCUMBLE	SCUMBLE.CV
cal500		398	21.3629	0.3372	0.3752
	LPROS	398	21.3629	0.3372	0.3752
	MLROS	510	22.0170	0.3187	0.4515
	MLSMOTE	880	25.6635	0.4241	0.2868
	MLSOL	581	21.3629	0.1725	0.6380
	REMEDIAL	496	25.2666	0.3413	0.4185
	MLDM	457	**6.2053**	**0.1483**	15.5041
corel5k		3 996	164.2963	0.3933	0.5716
	LPROS	4 995	153.6945	0.4013	0.5493
	MLROS	5 042	175.3099	0.3768	0.6161
	MLSMOTE	4 289	101.8507	0.3885	0.5925
	MLSOL	5 924	164.2963	**0.2476**	0.9869
	REMEDIAL	4 993	192.3708	0.3784	0.6126
	MLDM	4 064	**42.7687**	0.3245	20.0389
emotions		470	1.4838	0.0111	1.2597
	LPROS	580	1.3753	0.0085	1.1678
	MLROS	590	1.4665	0.0105	1.2743
	MLSMOTE	960	1.4939	0.0084	1.2668
	MLSOL	647	1.4838	**0.0007**	2.6512
	REMEDIAL	587	1.4870	0.0114	1.2010
	MLDM	560	**1.2795**	0.0055	9.6894
genbase		526	31.7604	0.0285	3.6045
	LPROS	648	**7.4295**	0.0497	2.2504
	MLROS	660	34.7501	0.0256	3.8329
	MLSMOTE	541	18.5159	0.0351	3.1187
	MLSOL	598	31.7604	**0.0131**	3.4909
	REMEDIAL	656	24.8065	0.0334	2.5901

MLD	Algoritmo	Ins.	MeanIR	SCUMBLE	SCUMBLE.CV
	MLDM	529	18.9618	0.0277	13.8791
medical		778	70.6477	0.0466	3.0404
	LPROS	974	**26.4806**	0.0769	2.2207
	MLROS	982	74.4952	0.0474	2.9793
	MLSMOTE	802	42.0877	0.0547	2.7966
	MLSOL	932	70.6477	0.0245	3.4226
	REMEDIAL	971	76.6356	0.0504	2.7377
	MLDM	784	31.1921	**0.0453**	15.6088
scene		1 926	1.2543	0.0003	4.2570
	LPROS	2 047	1.2543	**0.0002**	5.0247
	MLROS	2 408	1.4491	0.0020	2.2870
	MLSMOTE	2 408	1.2502	0.0003	5.3925
	MLSOL	2 405	1.2549	0.0007	3.2389
	REMEDIAL	2 802	1.4491	0.0010	5.9436
	MLDM	2145	**1.1938**	0.0005	11.2656
chess		1 340	91.5980	0.2610	0.9866
	LPROS	1 675	85.1091	0.2702	0.9317
	MLROS	1 722	100.9499	0.2538	1.0147
	MLSMOTE	1 507	91.5980	0.2321	1.1042
	MLSOL	1 922	91.5980	**0.1539**	1.4120
	REMEDIAL	1 674	102.3410	0.2531	1.0049
	MLDM	1379	**23.4003**	0.1856	18.6319
yeast		1 934	7.2128	0.1044	1.0652
	LPROS	2 396	5.1004	0.0870	1.1068
	MLROS	2 419	7.2461	0.0983	1.1088
	MLSMOTE	2 223	4.9979	0.1049	0.9911
	MLSOL	2 580	7.2128	**0.0633**	1.1249
	REMEDIAL	2 416	4.8604	0.0995	1.0953
	MLDM	1999	**4.8532**	0.1001	12.6699

Tabla. 5.7: Métricas de cada MLD tras aplicar cada algoritmo

5.4. Impacto del sobremuestreo en el rendimiento de clasificadores

Las métricas recopiladas en el apartado anterior denotan el grado de desbalanceo de un MLD, que suele ser indicativo del rendimiento que un clasificador puede tener sobre el mismo. Sin embargo, como se ha indicado anteriormente, la mejor manera de determinar este rendimiento es medirlo con **métricas para cada clasificador**.

En las tablas presentadas a continuación se recogen los valores para las métricas descritas en el apartado 5.1.2 tras clasificar cada uno de los MLD con los métodos recogidos en la sección 5.1.1, antes y después de aplicar los algoritmos de sobremuestreo. De nuevo, se destacan los mejores resultados en negrita.

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.452±0.010	0.347±0.016	**0.395±0.011**	0.328±0.014	0.320±0.010
	LPROS	0.456±0.019	0.347±0.016	0.393±0.017	0.328±0.014	0.320±0.010
	MLROS	0.452±0.020	0.339±0.011	0.349±0.019	0.331±0.017	0.326±0.010
	MLSMOTE	0.463±0.017	0.314±0.007	0.335±0.013	0.304±0.010	0.276±0.006
	MLSOL	0.456±0.014	**0.359±0.015**	0.383±0.010	**0.334±0.011**	**0.338±0.012**
	REMEDIAL	0.408±0.011	0.154±0.022	0.392±0.011	0.231±0.017	0.088±0.010
	MLDM	**0.464±0.015**	0.336±0.023	0.391±0.014	0.322±0.012	0.326±0.012
corel5k		0.024±0.001	0.088±0.008	0.149±0.010	0.102±0.010	0.016±0.003
	LPROS	0.021±0.001	**0.102±0.007**	0.143±0.007	**0.108±0.006**	0.024±0.006
	MLROS	0.025±0.002	0.097±0.006	**0.157±0.008**	0.102±0.001	0.028±0.005
	MLSMOTE	0.015±0.002	0.085±0.006	0.148±0.006	0.104±0.006	0.018±0.002
	MLSOL	**0.026±0.002**	0.096±0.007	0.145±0.005	0.107±0.005	**0.035±0.005**
	REMEDIAL	0.007±0.001	0.034±0.004	0.115±0.005	0.025±0.005	0.012±0.003
	MLDM	0.024±0.001	0.085±0.009	0.156±0.007	0.108±0.008	0.017±0.002
emotions		0.654±0.013	**0.552±0.021**	0.546±0.026	0.534±0.041	0.629±0.014
	LPROS	0.647±0.040	0.540±0.043	0.528±0.025	**0.559±0.008**	0.582±0.033
	MLROS	**0.663±0.021**	0.510±0.026	0.523±0.027	0.529±0.015	0.607±0.018
	MLSMOTE	0.620±0.039	0.520±0.023	0.513±0.026	0.522±0.029	0.607±0.027
	MLSOL	0.658±0.044	0.525±0.023	**0.547±0.023**	0.531±0.027	0.630±0.009
	REMEDIAL	0.557±0.026	0.317±0.026	0.438±0.065	0.505±0.027	0.371±0.021
	MLDM	0.654±0.028	0.548±0.026	0.545±0.034	0.527±0.025	**0.639±0.013**
genbase		0.039±0.044	0.984±0.020	0.985±0.017	0.978±0.023	0.953±0.041
	LPROS	0.052±0.048	0.936±0.110	0.943±0.096	0.936±0.099	0.879±0.161
	MLROS	**0.093±0.020**	0.980±0.011	0.983±0.010	0.969±0.009	**0.966±0.012**
	MLSMOTE	0.072±0.028	0.929±0.096	0.920±0.117	0.930±0.109	0.864±0.183
	MLSOL	0.073±0.047	0.934±0.094	0.932±0.101	0.920±0.114	0.854±0.202
	REMEDIAL	0.065±0.041	0.932±0.105	0.937±0.103	0.928±0.108	0.877±0.168
	MLDM	0.065±0.023	**0.989±0.010**	**0.987±0.011**	**0.985±0.009**	0.955±0.009
medical		0.053±0.031	0.771±0.021	**0.784±0.018**	0.765±0.039	0.585±0.016
	LPROS	0.033±0.030	**0.781±0.021**	0.746±0.020	0.754±0.023	0.574±0.052
	MLROS	0.052±0.030	0.766±0.023	0.751±0.018	0.750±0.025	0.498±0.028
	MLSMOTE	0.035±0.032	0.766±0.014	0.774±0.016	0.753±0.039	0.583±0.016
	MLSOL	0.046±0.042	0.775±0.016	0.758±0.012	0.765±0.019	**0.608±0.037**
	REMEDIAL	**0.064±0.004**	0.700±0.021	0.734±0.045	0.670±0.053	0.466±0.060
	MLDM	0.024±0.033	0.772±0.013	0.781±0.016	**0.766±0.041**	0.587±0.017
scene		0.472±0.042	**0.580±0.019**	0.551±0.024	0.588±0.034	0.679±0.016
	LPROS	0.450±0.024	0.566±0.018	0.555±0.011	0.578±0.024	0.668±0.013
	MLROS	0.508±0.041	0.579±0.035	0.537±0.011	0.592±0.018	0.673±0.011

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLSMOTE	0.470±0.051	0.565±0.022	0.548±0.024	**0.601±0.023**	0.682±0.022
	MLSOL	0.459±0.054	0.571±0.017	**0.561±0.015**	0.598±0.020	**0.709±0.023**
	REMEDIAL	0.499±0.020	0.555±0.035	0.550±0.025	0.572±0.021	0.655±0.031
	MLDM	**0.518±0.057**	0.545±0.023	0.548±0.020	0.583±0.030	0.663±0.035
chess		0.016±0.002	0.249±0.017	0.247±0.015	0.139±0.008	0.046±0.018
	LPROS	0.014±0.002	0.241±0.031	0.241±0.007	0.147±0.018	**0.064±0.018**
	MLROS	0.015±0.004	0.252±0.016	0.238±0.017	0.150±0.016	0.044±0.012
	MLSMOTE	0.014±0.005	0.236±0.022	0.253±0.017	0.135±0.013	0.030±0.011
	MLSOL	**0.017±0.001**	0.243±0.024	**0.272±0.007**	**0.154±0.013**	0.059±0.009
	REMEDIAL	0.007±0.007	0.080±0.029	0.203±0.017	0.033±0.012	0.005±0.004
	MLDM	0.017±0.002	**0.254±0.019**	0.258±0.010	0.154±0.011	0.052±0.022
yeast		0.623±0.023	0.554±0.020	0.552±0.032	0.506±0.018	0.614±0.029
	LPROS	0.627±0.022	0.528±0.011	0.526±0.015	0.500±0.020	0.598±0.029
	MLROS	0.621±0.019	0.524±0.025	0.530±0.019	**0.513±0.017**	0.531±0.033
	MLSMOTE	0.614±0.014	0.525±0.023	0.535±0.014	0.498±0.014	0.590±0.021
	MLSOL	0.629±0.018	0.537±0.020	0.531±0.023	0.497±0.018	0.611±0.021
	REMEDIAL	0.589±0.009	0.484±0.068	0.557±0.043	0.355±0.034	0.486±0.059
	MLDM	**0.632±0.029**	**0.565±0.016**	**0.562±0.027**	0.504±0.007	**0.616±0.031**

Tabla. 5.8: Valores para la métrica F-Measure basada en ejemplos

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.252±0.013	0.165±0.005	0.186±0.011	0.198±0.004	**0.140±0.002**
	LPROS	0.245±0.008	0.165±0.005	0.185±0.004	0.198±0.004	0.140±0.002
	MLROS	0.243±0.008	0.189±0.002	0.197±0.004	0.200±0.005	0.141±0.002
	MLSMOTE	**0.203±0.012**	0.170±0.003	**0.178±0.003**	**0.180±0.002**	0.142±0.002
	MLSOL	0.256±0.010	0.174±0.003	0.191±0.004	0.199±0.005	0.142±0.002
	REMEDIAL	0.220±0.007	**0.150±0.004**	0.182±0.005	0.184±0.004	0.146±0.003
	MLDM	0.245±0.003	0.167±0.005	0.195±0.006	0.214±0.013	0.141±0.003
corel5k		0.565±0.015	0.010±0.000	0.013±0.000	0.017±0.000	**0.009±0.000**
	LPROS	0.583±0.033	0.011±0.000	0.014±0.000	0.017±0.000	0.010±0.000
	MLROS	0.534±0.017	0.010±0.000	0.014±0.000	0.017±0.000	0.010±0.000
	MLSMOTE	0.562±0.120	0.010±0.000	0.013±0.000	0.017±0.000	**0.009±0.000**
	MLSOL	0.525±0.043	0.010±0.000	0.013±0.000	0.016±0.000	0.010±0.000
	REMEDIAL	**0.267±0.059**	**0.009±0.000**	**0.012±0.000**	0.010±0.000	**0.009±0.000**
	MLDM	0.571±0.020	0.010±0.00	0.013±0.000	0.017±0.000	**0.009±0.000**
emotions		0.211±0.007	**0.248±0.010**	**0.252±0.007**	0.273±0.019	0.188±0.008
	LPROS	0.223±0.024	0.256±0.024	0.253±0.012	0.261±0.008	0.213±0.012
	MLROS	**0.204±0.006**	0.270±0.011	0.263±0.011	0.269±0.014	0.205±0.004
	MLSMOTE	0.221±0.021	0.257±0.010	0.265±0.012	0.271±0.017	0.204±0.011
	MLSOL	0.208±0.023	0.263±0.013	0.258±0.023	0.278±0.015	0.194±0.006
	REMEDIAL	0.228±0.014	0.268±0.013	0.262±0.018	**0.258±0.014**	0.255±0.004

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLDM	0.214±0.012	0.258±0.010	0.262±0.015	0.276±0.011	**0.186±0.008**
genbase		**0.424±0.433**	0.002±0.003	0.002±0.002	0.003±0.003	0.005±0.004
	LPROS	0.511±0.436	0.004±0.004	0.003±0.004	0.004±0.005	0.007±0.006
	MLROS	0.741±0.169	0.002±0.001	0.002±0.001	0.004±0.001	**0.005±0.001**
	MLSMOTE	0.724±0.283	0.005±0.005	0.005±0.005	0.005±0.005	0.008±0.006
	MLSOL	0.666±0.361	0.005±0.005	0.005±0.005	0.006±0.007	0.008±0.007
	REMEDIAL	0.645±0.385	0.004±0.005	0.004±0.004	0.005±0.006	0.007±0.005
	MLDM	0.621±0.281	**0.001±0.001**	**0.001±0.001**	**0.002±0.001**	**0.005±0.001**
medical		0.629±0.355	**0.011±0.001**	**0.011±0.001**	**0.013±0.002**	**0.016±0.001**
	LPROS	0.577±0.503	**0.011±0.001**	0.012±0.001	0.014±0.001	0.017±0.001
	MLROS	0.638±0.359	**0.011±0.001**	0.012±0.001	0.014±0.002	0.018±0.002
	MLSMOTE	0.531±0.460	**0.011±0.000**	**0.011±0.001**	0.014±0.002	**0.016±0.001**
	MLSOL	0.400±0.340	**0.011±0.001**	0.012±0.001	**0.013±0.001**	**0.016±0.001**
	REMEDIAL	0.804±0.044	0.013±0.000	0.012±0.002	0.016±0.002	0.018±0.002
	MLDM	**0.357±0.451**	**0.011±0.001**	**0.011±0.001**	0.013±0.002	**0.016±0.001**
scene		0.243±0.060	0.133±0.004	0.142±0.005	0.150±0.014	**0.088±0.004**
	LPROS	0.308±0.049	0.136±0.002	0.140±0.004	0.154±0.007	0.100±0.005
	MLROS	**0.230±0.033**	0.135±0.008	0.144±0.007	0.148±0.008	0.092±0.005
	MLSMOTE	0.240±0.050	0.135±0.006	0.143±0.008	**0.143±0.007**	0.088±0.005
	MLSOL	0.244±0.042	0.140±0.005	0.141±0.003	0.151±0.007	0.093±0.004
	REMEDIAL	0.280±0.031	**0.131±0.006**	**0.136±0.009**	0.147±0.006	0.090±0.006
	MLDM	0.243±0.044	0.144±0.006	0.149±0.007	0.152±0.011	0.089±0.007
chess		0.575±0.047	**0.011±0.000**	0.014±0.000	0.018±0.001	**0.011±0.000**
	LPROS	0.548±0.133	0.012±0.000	0.015±0.000	0.018±0.000	**0.011±0.000**
	MLROS	0.512±0.128	**0.011±0.000**	0.014±0.000	0.017±0.001	**0.011±0.000**
	MLSMOTE	0.488±0.223	**0.011±0.000**	0.013±0.000	0.017±0.000	**0.011±0.000**
	MLSOL	0.613±0.036	**0.011±0.000**	0.013±0.001	0.017±0.000	**0.011±0.000**
	REMEDIAL	**0.310±0.370**	**0.011±0.000**	**0.012±0.001**	**0.012±0.000**	**0.011±0.000**
	MLDM	0.616±0.027	**0.011±0.000**	0.014±0.001	0.018±0.001	**0.011±0.000**
yeast		0.228±0.010	0.251±0.010	0.263±0.014	0.284±0.009	**0.196±0.012**
	LPROS	0.235±0.013	0.272±0.007	0.274±0.011	0.293±0.009	0.209±0.013
	MLROS	0.225±0.010	0.270±0.011	0.271±0.010	**0.278±0.011**	0.213±0.010
	MLSMOTE	0.232±0.008	0.266±0.012	0.272±0.009	0.289±0.009	0.205±0.008
	MLSOL	0.235±0.011	0.270±0.011	0.278±0.011	0.295±0.010	0.205±0.009
	REMEDIAL	0.233±0.012	**0.237±0.013**	**0.246±0.012**	0.281±0.012	0.220±0.011
	MLDM	**0.224±0.013**	0.245±0.010	0.259±0.017	0.285±0.001	**0.196±0.012**

Tabla. 5.9: Valores para la métrica Hamming-Loss

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.213±0.009	0.148±0.015	0.173±0.007	0.156±0.011	0.089±0.012
	LPROS	0.215±0.021	0.148±0.015	0.169±0.011	0.156±0.011	0.089±0.012

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLROS	0.220±0.019	0.159±0.010	0.172±0.010	0.156±0.015	0.102±0.011
	MLSMOTE	0.196±0.014	0.136±0.010	0.152±0.004	0.144±0.012	0.098±0.009
	MLSOL	0.226±0.008	**0.169±0.017**	**0.180±0.013**	**0.164±0.013**	**0.111±0.016**
	REMEDIAL	0.186±0.005	0.075±0.017	0.178±0.015	0.136±0.014	0.048±0.013
	MLDM	**0.227±0.014**	0.133±0.012	0.161±0.008	0.159±0.005	0.091±0.015
corel5k		0.139±0.020	**0.196±0.014**	**0.193±0.016**	0.121±0.014	0.185±0.016
	LPROS	0.124±0.012	0.192±0.011	0.171±0.012	0.130±0.018	0.176±0.015
	MLROS	0.147±0.010	0.190±0.016	0.188±0.015	0.117±0.013	0.185±0.016
	MLSMOTE	0.071±0.013	0.178±0.015	0.174±0.012	0.127±0.019	0.182±0.015
	MLSOL	**0.156±0.016**	0.195±0.014	0.189±0.021	0.125±0.005	**0.189±0.015**
	REMEDIAL	0.049±0.010	0.188±0.014	0.192±0.014	**0.168±0.018**	0.182±0.016
	MLDM	0.141±0.010	0.185±0.020	0.184±0.013	0.116±0.022	0.182±0.015
emotions		0.674±0.010	**0.588±0.006**	0.585±0.018	0.556±0.033	0.642±0.025
	LPROS	0.653±0.043	0.583±0.039	0.579±0.023	**0.575±0.007**	0.610±0.029
	MLROS	**0.678±0.019**	0.560±0.027	0.567±0.025	0.560±0.018	0.618±0.005
	MLSMOTE	0.656±0.039	0.568±0.014	0.558±0.009	0.548±0.026	0.642±0.020
	MLSOL	0.673±0.042	0.570±0.025	**0.588±0.031**	0.547±0.026	0.643±0.010
	REMEDIAL	0.617±0.021	0.384±0.042	0.465±0.040	0.518±0.021	0.370±0.037
	MLDM	0.665±0.032	0.585±0.015	0.575±0.028	0.551±0.022	**0.653±0.020**
genbase		0.141±0.059	0.870±0.159	0.875±0.141	0.851±0.102	**0.748±0.151**
	LPROS	**0.192±0.082**	0.839±0.141	0.838±0.137	0.828±0.147	0.741±0.142
	MLROS	0.168±0.077	0.895±0.076	0.865±0.079	0.798±0.055	0.720±0.050
	MLSMOTE	0.124±0.056	0.787±0.169	0.771±0.182	0.776±0.129	0.697±0.080
	MLSOL	0.162±0.068	0.818±0.186	0.825±0.172	0.783±0.210	0.707±0.172
	REMEDIAL	0.168±0.120	0.829±0.182	0.844±0.168	0.804±0.219	0.716±0.131
	MLDM	0.111±0.025	**0.933±0.073**	**0.907±0.061**	**0.890±0.055**	0.741±0.049
medical		0.226±0.129	**0.661±0.061**	**0.636±0.048**	**0.598±0.047**	0.531±0.075
	LPROS	0.158±0.155	0.628±0.045	0.592±0.053	0.590±0.048	0.506±0.070
	MLROS	0.199±0.118	0.654±0.069	0.616±0.070	0.584±0.039	0.477±0.087
	MLSMOTE	0.180±0.137	0.627±0.050	0.609±0.043	0.555±0.065	0.530±0.072
	MLSOL	**0.296±0.067**	0.632±0.065	0.612±0.059	0.577±0.038	**0.543±0.081**
	REMEDIAL	0.156±0.034	0.626±0.055	0.612±0.049	0.560±0.044	0.505±0.087
	MLDM	0.242±0.134	0.634±0.067	0.618±0.039	0.581±0.086	0.527±0.072
scene		0.545±0.022	**0.636±0.013**	0.612±0.013	0.592±0.033	0.734±0.011
	LPROS	0.507±0.018	0.630±0.009	0.619±0.009	0.586±0.020	0.720±0.010
	MLROS	0.561±0.042	0.634±0.028	0.603±0.012	0.598±0.017	0.723±0.008
	MLSMOTE	0.541±0.040	0.625±0.017	0.606±0.020	0.605±0.018	0.732±0.016
	MLSOL	0.532±0.018	0.626±0.008	**0.620±0.004**	**0.606±0.019**	**0.740±0.014**
	REMEDIAL	0.547±0.014	0.621±0.028	0.609±0.027	0.585±0.021	0.718±0.022
	MLDM	**0.569±0.035**	0.612±0.015	0.603±0.016	0.587±0.027	0.724±0.025
chess		0.021±0.004	**0.325±0.023**	0.275±0.018	0.174±0.018	0.277±0.020
	LPROS	0.024±0.008	0.308±0.026	0.263±0.016	0.178±0.008	0.266±0.019
	MLROS	**0.025±0.014**	0.315±0.027	0.270±0.021	0.185±0.018	0.276±0.019

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLSMOTE	0.020±0.001	0.318±0.024	0.284±0.024	0.207±0.013	0.275±0.019
	MLSOL	0.024±0.009	0.318±0.021	**0.288±0.015**	0.196±0.011	**0.279±0.025**
	REMEDIAL	0.017±0.002	0.289±0.016	0.288±0.026	**0.242±0.020**	0.272±0.020
	MLDM	0.022±0.006	0.288±0.029	0.269±0.025	0.172±0.030	0.274±0.025
yeast		0.438±0.015	0.381±0.008	0.398±0.013	0.374±0.013	0.370±0.004
	LPROS	**0.453±0.011**	0.397±0.006	0.393±0.016	0.373±0.019	0.395±0.014
	MLROS	0.436±0.014	0.390±0.011	0.392±0.015	**0.386±0.014**	0.349±0.013
	MLSMOTE	0.436±0.005	0.390±0.011	0.399±0.006	0.375±0.014	0.394±0.012
	MLSOL	0.445±0.017	**0.401±0.011**	0.395±0.014	0.371±0.020	**0.409±0.012**
	REMEDIAL	0.411±0.025	0.267±0.015	0.365±0.037	0.277±0.017	0.256±0.011
	MLDM	0.443±0.012	0.387±0.008	**0.402±0.008**	0.370±0.013	0.381±0.011

Tabla. 5.10: Valores para la métrica Macro F-Measure

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.456±0.014	0.348±0.016	**0.398±0.011**	0.332±0.015	0.316±0.010
	LPROS	0.461±0.019	0.348±0.016	0.396±0.016	0.332±0.015	0.316±0.010
	MLROS	0.459±0.018	0.342±0.010	0.353±0.019	0.334±0.018	0.325±0.009
	MLSMOTE	0.462±0.020	0.318±0.008	0.340±0.013	0.317±0.009	0.279±0.006
	MLSOL	0.457±0.013	**0.363±0.014**	0.387±0.009	**0.339±0.012**	**0.338±0.013**
	REMEDIAL	0.417±0.011	0.155±0.023	0.394±0.011	0.273±0.018	0.085±0.010
	MLDM	**0.468±0.014**	0.337±0.024	0.393±0.015	0.320±0.014	0.322±0.013
corel5k		0.026±0.001	0.113±0.011	0.165±0.010	0.104±0.009	0.027±0.005
	LPROS	0.025±0.000	**0.126±0.009**	0.159±0.005	**0.109±0.006**	0.033±0.007
	MLROS	0.028±0.001	0.123±0.007	**0.173±0.009**	0.105±0.002	0.044±0.008
	MLSMOTE	0.021±0.000	0.109±0.008	0.164±0.005	0.106±0.006	0.029±0.005
	MLSOL	**0.029±0.001**	0.122±0.010	0.163±0.005	0.108±0.005	**0.054±0.009**
	REMEDIAL	0.021±0.000	0.049±0.007	0.138±0.007	0.043±0.009	0.019±0.004
	MLDM	0.026±0.000	0.110±0.011	0.172±0.007	0.109±0.008	0.027±0.004
emotions		0.683±0.012	**0.600±0.004**	**0.599±0.012**	0.564±0.032	0.676±0.016
	LPROS	0.668±0.037	0.592±0.039	0.586±0.019	**0.583±0.009**	0.632±0.026
	MLROS	0.687±0.018	0.567±0.027	0.577±0.022	0.566±0.019	0.647±0.015
	MLSMOTE	0.662±0.038	0.575±0.014	0.566±0.013	0.554±0.024	0.652±0.020
	MLSOL	**0.688±0.041**	0.579±0.020	0.592±0.030	0.559±0.025	0.674±0.004
	REMEDIAL	0.621±0.023	0.417±0.033	0.502±0.050	0.530±0.024	0.440±0.022
	MLDM	0.677±0.026	0.592±0.013	0.583±0.024	0.561±0.020	**0.680±0.010**
genbase		0.051±0.047	0.977±0.033	0.979±0.028	0.965±0.035	0.941±0.053
	LPROS	0.052±0.048	0.943±0.081	0.947±0.077	0.931±0.099	0.892±0.126
	MLROS	**0.101±0.012**	0.982±0.011	0.979±0.011	0.953±0.011	0.945±0.012
	MLSMOTE	0.088±0.021	0.927±0.080	0.923±0.092	0.919±0.108	0.870±0.140
	MLSOL	0.074±0.047	0.932±0.082	0.930±0.088	0.905±0.117	0.867±0.150
	REMEDIAL	0.072±0.041	0.935±0.083	0.944±0.085	0.918±0.111	0.886±0.126

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLDM	0.096±0.014	**0.988±0.011**	**0.985±0.010**	**0.977±0.011**	**0.946±0.009**
medical		0.053±0.031	**0.807±0.012**	**0.804±0.012**	**0.760±0.034**	0.665±0.017
	LPROS	0.033±0.030	0.805±0.015	0.779±0.010	0.743±0.021	0.645±0.037
	MLROS	0.052±0.030	0.795±0.018	0.779±0.012	0.748±0.026	0.605±0.034
	MLSMOTE	0.035±0.032	0.798±0.008	0.795±0.009	0.747±0.036	0.664±0.016
	MLSOL	0.046±0.042	0.802±0.014	0.784±0.013	0.756±0.012	**0.679±0.027**
	REMEDIAL	**0.064±0.004**	0.734±0.015	0.760±0.039	0.685±0.054	0.556±0.060
	MLDM	0.024±0.033	0.803±0.011	**0.804±0.012**	0.755±0.035	0.665±0.017
scene		0.528±0.033	**0.628±0.012**	0.603±0.016	0.583±0.035	0.730±0.012
	LPROS	0.482±0.021	0.618±0.008	0.608±0.011	0.575±0.021	0.711±0.008
	MLROS	**0.544±0.033**	0.624±0.029	0.592±0.016	0.587±0.017	0.721±0.011
	MLSMOTE	0.517±0.048	0.618±0.019	0.598±0.021	**0.596±0.020**	0.731±0.019
	MLSOL	0.515±0.016	0.616±0.011	**0.610±0.010**	0.594±0.018	**0.733±0.014**
	REMEDIAL	0.520±0.018	0.613±0.025	0.601±0.022	0.575±0.020	0.716±0.023
	MLDM	0.544±0.041	0.599±0.016	0.592±0.017	0.578±0.029	0.721±0.025
chess		**0.023±0.000**	0.297±0.017	0.276±0.015	0.145±0.011	0.063±0.024
	LPROS	0.022±0.001	0.284±0.028	0.268±0.006	0.152±0.015	**0.087±0.025**
	MLROS	0.023±0.001	**0.298±0.017**	0.267±0.016	**0.160±0.015**	0.065±0.016
	MLSMOTE	0.023±0.001	0.282±0.020	0.286±0.011	0.150±0.009	0.042±0.015
	MLSOL	0.023±0.001	0.291±0.018	**0.302±0.008**	0.158±0.013	0.087±0.016
	REMEDIAL	0.020±0.001	0.105±0.032	0.248±0.017	0.048±0.019	0.005±0.006
	MLDM	0.023±0.001	0.292±0.021	0.286±0.011	0.156±0.015	0.071±0.032
yeast		0.641±0.018	0.576±0.017	0.577±0.027	0.530±0.016	0.640±0.023
	LPROS	0.641±0.017	0.551±0.009	0.553±0.015	0.524±0.017	0.626±0.025
	MLROS	0.638±0.016	0.549±0.021	0.553±0.016	**0.540±0.015**	0.587±0.020
	MLSMOTE	0.633±0.011	0.550±0.020	0.559±0.009	0.525±0.013	0.623±0.016
	MLSOL	0.641±0.015	0.560±0.016	0.554±0.021	0.524±0.017	0.638±0.017
	REMEDIAL	0.614±0.009	0.522±0.057	0.581±0.037	0.427±0.034	0.529±0.046
	MLDM	**0.647±0.024**	**0.587±0.014**	**0.583±0.024**	0.529±0.005	**0.642±0.026**

Tabla. 5.11: Valores para la métrica Micro F-Measure

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
cal500		0.135±0.051	0.691±0.089	0.806±0.065	**0.988±0.008**	0.117±0.040
	LPROS	0.146±0.059	0.691±0.089	0.812±0.043	**0.988±0.008**	0.117±0.040
	MLROS	0.132±0.063	**0.826±0.061**	0.842±0.052	0.977±0.016	0.139±0.065
	MLSMOTE	0.171±0.056	0.707±0.024	0.789±0.045	0.977±0.017	**0.182±0.042**
	MLSOL	0.123±0.048	0.745±0.043	0.847±0.020	**0.988±0.008**	0.127±0.056
	REMEDIAL	**0.220±0.051**	0.679±0.060	0.745±0.034	**0.988±0.008**	0.175±0.032
	MLDM	0.146±0.048	0.731±0.045	**0.857±0.039**	**0.988±0.008**	0.129±0.047
corel5k		0.996±0.003	0.711±0.016	0.797±0.018	0.988±0.003	0.741±0.010
	LPROS	0.996±0.002	**0.755±0.005**	0.825±0.013	0.973±0.007	**0.800±0.012**

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLROS	0.997±0.002	0.725±0.016	0.801±0.019	0.984±0.003	0.756±0.012
	MLSMOTE	**0.999±0.002**	0.713±0.014	0.805±0.009	0.985±0.008	0.748±0.013
	MLSOL	0.998±0.002	0.721±0.014	0.799±0.011	0.985±0.002	0.748±0.021
	REMEDIAL	0.999±0.001	0.712±0.005	**0.830±0.010**	**0.989±0.002**	0.734±0.016
	MLDM	0.996±0.004	0.717±0.013	0.799±0.018	0.984±0.007	0.747±0.010
emotions		0.297±0.041	0.395±0.042	0.403±0.032	0.445±0.034	0.245±0.038
	LPROS	0.305±0.021	0.399±0.049	0.427±0.035	0.421±0.032	0.279±0.015
	MLROS	0.294±0.025	0.427±0.027	0.403±0.020	**0.453±0.037**	0.284±0.034
	MLSMOTE	**0.332±0.053**	0.396±0.060	0.432±0.045	0.440±0.054	0.272±0.054
	MLSOL	0.299±0.035	**0.452±0.050**	0.406±0.019	0.444±0.045	0.289±0.046
	REMEDIAL	0.313±0.042	0.395±0.031	**0.436±0.046**	0.448±0.085	**0.336±0.045**
	MLDM	0.320±0.052	0.422±0.028	0.430±0.028	0.450±0.028	0.250±0.027
genbase		0.988±0.026	0.004±0.007	0.007±0.007	0.028±0.037	0.003±0.004
	LPROS	0.993±0.008	0.042±0.086	0.047±0.089	0.070±0.093	0.064±0.107
	MLROS	0.987±0.015	0.013±0.007	0.007±0.009	0.018±0.008	0.000±0.017
	MLSMOTE	0.992±0.011	0.046±0.083	**0.067±0.133**	**0.086±0.103**	0.063±0.121
	MLSOL	0.978±0.025	0.047±0.089	0.054±0.096	0.081±0.114	**0.083±0.136**
	REMEDIAL	0.992±0.007	**0.048±0.096**	0.056±0.100	0.077±0.112	0.061±0.117
	MLDM	**0.997±0.004**	0.006±0.006	0.012±0.008	0.010±0.008	0.018±0.012
medical		0.976±0.020	0.183±0.022	0.202±0.018	0.231±0.035	0.241±0.032
	LPROS	0.969±0.016	0.201±0.018	**0.256±0.031**	0.237±0.026	0.264±0.043
	MLROS	0.974±0.023	**0.205±0.040**	0.230±0.013	0.245±0.016	**0.293±0.035**
	MLSMOTE	**0.990±0.009**	0.202±0.013	0.216±0.011	0.250±0.036	0.254±0.035
	MLSOL	0.977±0.021	0.185±0.031	0.233±0.022	0.238±0.011	0.253±0.046
	REMEDIAL	0.960±0.023	0.183±0.019	0.221±0.063	**0.370±0.026**	0.254±0.042
	MLDM	0.969±0.018	0.185±0.027	0.204±0.024	0.230±0.035	0.243±0.031
scene		0.514±0.126	0.406±0.020	0.435±0.016	0.414±0.029	0.229±0.021
	LPROS	0.575±0.044	0.422±0.012	0.436±0.013	**0.433±0.023**	0.241±0.022
	MLROS	0.513±0.057	0.413±0.023	**0.466±0.016**	0.408±0.021	**0.242±0.018**
	MLSMOTE	0.559±0.112	0.401±0.024	0.450±0.036	0.399±0.022	0.229±0.014
	MLSOL	0.524±0.061	0.425±0.015	0.430±0.018	0.414±0.025	0.238±0.020
	REMEDIAL	**0.589±0.033**	0.391±0.021	0.420±0.029	0.429±0.018	0.233±0.019
	MLDM	0.495±0.140	**0.438±0.024**	0.447±0.019	0.424±0.022	0.229±0.020
chess		0.996±0.005	0.553±0.031	0.702±0.031	0.954±0.021	0.696±0.038
	LPROS	0.996±0.005	**0.584±0.012**	0.715±0.018	0.930±0.017	0.730±0.028
	MLROS	0.995±0.005	0.574±0.026	0.697±0.041	0.921±0.009	**0.746±0.019**
	MLSMOTE	0.995±0.006	0.561±0.024	0.682±0.024	0.952±0.012	0.704±0.035
	MLSOL	0.998±0.003	0.575±0.020	0.675±0.026	0.917±0.016	0.711±0.033
	REMEDIAL	**0.998±0.002**	0.584±0.008	**0.724±0.021**	**0.979±0.012**	0.715±0.040
	MLDM	**0.998±0.002**	0.568±0.019	0.674±0.019	0.940±0.014	0.704±0.033
yeast		0.253±0.010	**0.425±0.068**	0.410±0.058	0.538±0.031	0.229±0.026
	LPROS	0.267±0.017	0.344±0.035	0.368±0.044	0.467±0.043	0.237±0.029
	MLROS	0.273±0.027	0.355±0.036	0.390±0.034	0.501±0.022	0.237±0.035

MLD	Algoritmo	BPMLL	BR-J48	HOMER	LP-J48	MLkNN
	MLSMOTE	0.273±0.027	0.384±0.039	0.357±0.071	0.537±0.017	0.241±0.026
	MLSOL	0.258±0.015	0.380±0.057	0.405±0.044	0.551±0.025	**0.247±0.019**
	REMEDIAL	**0.285±0.015**	0.400±0.024	0.395±0.104	**0.660±0.037**	0.245±0.029
	MLDM	0.248±0.021	0.377±0.035	**0.454±0.053**	0.507±0.015	0.227±0.027

Tabla. 5.12: Valores para la métrica One-Error

5.5. Tiempo de ejecución de los algoritmos

Un aspecto muy importante a la hora de considerar el empleo de un algoritmo u otro es el **tiempo de ejecución**. Durante la experimentación, se ha medido el tiempo empleado por cada uno de los algoritmos para llevar a cabo el sobremuestreo del MLD original. En la Tabla 5.13 se recogen dichos valores, siendo estos la media del tiempo empleado para cada una de las 5 particiones de entrenamiento y test.

MLD	Algoritmo	Tiempo (segundos)
cal500		
	LPROS	0.0532
	MLROS	0.3046
	MLSMOTE	5.1938
	MLSOL	90.5037
	REMEDIAL	0.3631
	MLDM	237.4000
corel5k		
	LPROS	4.1254
	MLROS	3.8532
	MLSMOTE	46.1337
	MLSOL	173 074.2035
	REMEDIAL	6.0238
	MLDM	264.6000
emotions		
	LPROS	0.0262
	MLROS	0.0195
	MLSMOTE	183.3267
	MLSOL	123.6310
	REMEDIAL	0.0339
	MLDM	322.2000
genbase		
	LPROS	0.1333
	MLROS	0.2245

MLD	Algoritmo	Tiempo (segundos)
	MLSMOTE	9.7148
	MLSOL	7 738.6551
	REMEDIAL	0.1697
	MLDM	231.6000
medical		
	LPROS	0.2010
	MLROS	0.3528
	MLSMOTE	8.3248
	MLSOL	21 110.3830
	REMEDIAL	0.3009
	MLDM	220.8000
scene		
	LPROS	0.1027
	MLROS	0.0602
	MLSMOTE	4 492.3691
	MLSOL	7 755.5717
	REMEDIAL	0.0869
	MLDM	257.8000
chess		
	LPROS	0.8729
	MLROS	1.1502
	MLSMOTE	2.8889
	MLSOL	7 938.8328
	REMEDIAL	1.2465
	MLDM	262.6000
yeast		
	LPROS	0.1354
	MLROS	0.0792
	MLSMOTE	420.5287
	MLSOL	2 943.3043
	REMEDIAL	0.1444
	MLDM	232.4000

Tabla. 5.13: Tiempo empleado para cada MLD por cada algoritmo

5.6. Análisis de resultados

Una vez se han recogido los resultados de la experimentación en las tablas anteriores, se puede proceder al análisis de los mismos, intentando extraer conclusiones y facilitar su interpretación a través de representaciones gráficas.

5.6.1. Impacto en el desbalanceo

El primer objetivo de la experimentación ha sido observar el impacto de los métodos de sobremuestreo en la distribución de los MLD que componen el estudio y, por tanto, de su nivel de desbalanceo. Este impacto puede medirse por medio de **métricas de descripción**, algunas de las cuales fueron explicadas en el apartado 2.4.2.

Los resultados recogidos en la Tabla 5.7 muestran que el método propuesto disminuye el desbalanceo en los MLD sobre los que se han llevado a cabo pruebas. En cuanto a la métrica $MeanIR$, se puede apreciar que MLDM disminuye este valor, lo que indica un nivel de desbalanceo medio inferior no solo al MLD original, sino **a la mayoría de los MLD tras aplicar otros algoritmos de sobremuestreo**. En la Figura 5.1 se puede apreciar cómo, para cinco de los ocho MLD que componen el experimento, el valor de $MeanIR$ obtenido tras aplicar MLDM es menor que tras aplicar cualquier otro algoritmo. En los tres MLD restantes, es uno de los tres algoritmos que más disminuye el $MeanIR$, aunque no sea el mejor.

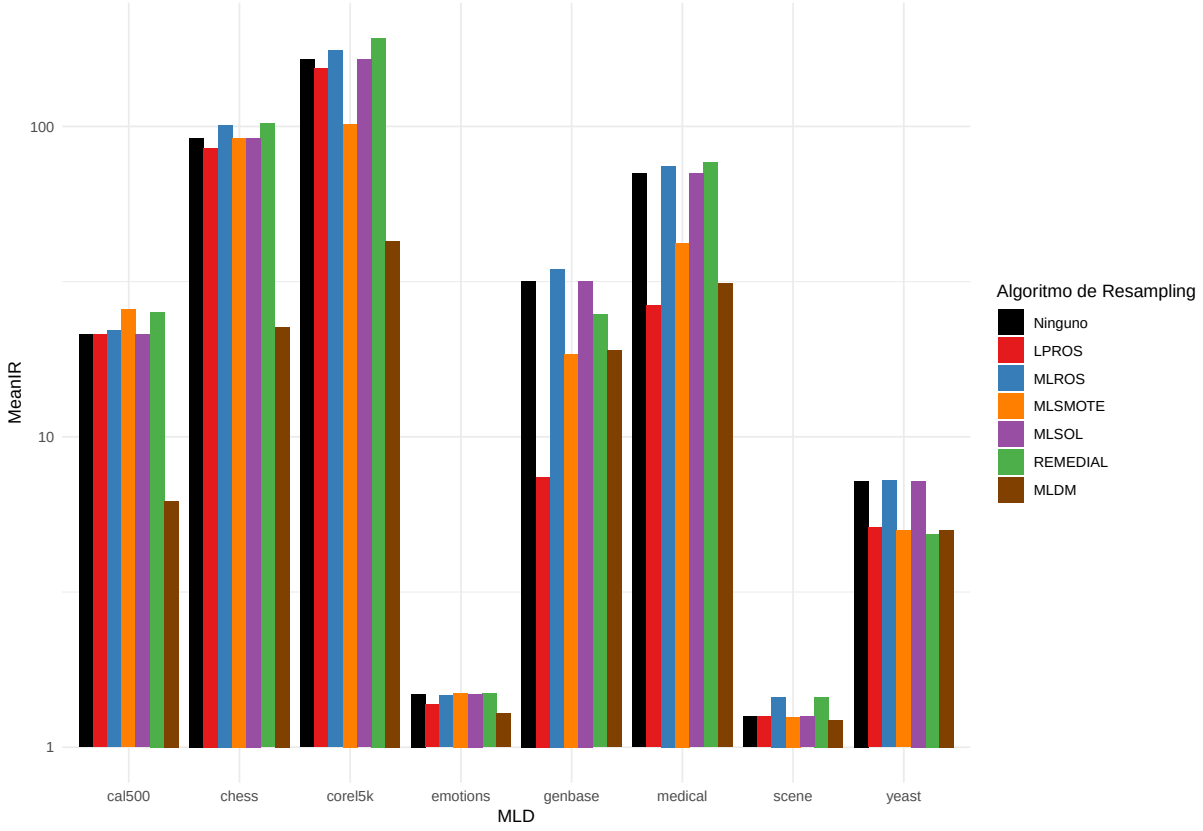

Figura 5.1: MeanIR al aplicar algoritmos, en escala logarítmica.

Centrando la atención en los MLD en los que mejor funciona MLDM, al menos en términos de reducción del $MeanIR$, cabe destacar que aquellos en los que la diferencia con el resto de algoritmos es mayor son los que tienen un **mayor número de atributos**

categóricos. Una posible explicación frente a este hecho es que la codificación *one-hot* hace que MLDM trabaje con dichos atributos como si fueran numéricos, mejorando el rendimiento con respecto al resto, que sí que realizan una distinción.

En cuanto a la métrica $SCUMBLE$, en la Figura 5.2 se puede visualizar, de nuevo, la diferencia entre la aplicación de los distintos algoritmos. En este caso, la mejora **no es tan destacable** como con la métrica $MeanIR$, siendo el mejor algoritmo (de nuevo, en términos tan solo de reducción de $SCUMBLE$) en una ocasión, para el MLD `cal500`. Para el resto, se mantiene como el segundo o tercero, disminuyendo, salvo en una ocasión, el valor de $SCUMBLE$ del MLD original. Este caso concreto ocurre con el MLD `scene`, en el que tan solo dos algoritmos son capaces de reducir el $SCUMBLE$ original.

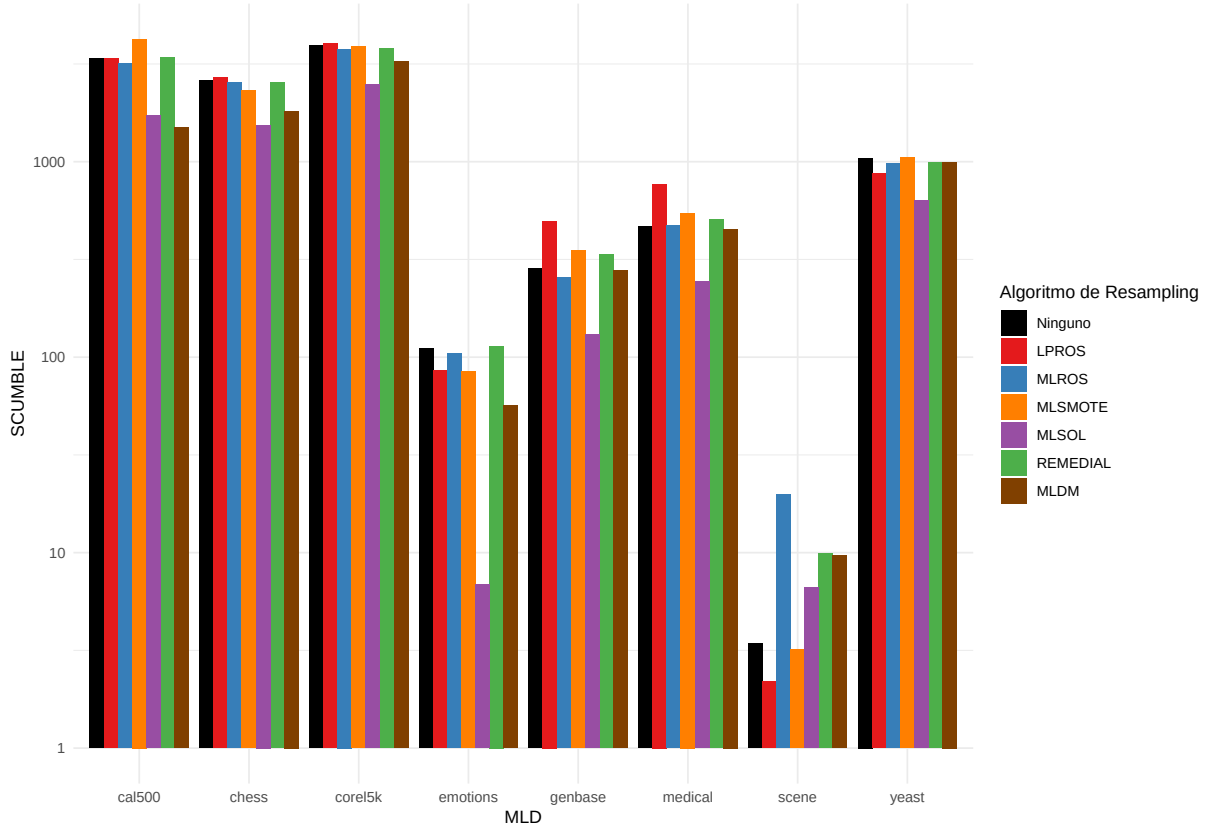

Figura 5.2: SCUMBLE al aplicar algoritmos, escala logarítmica, tras multiplicar por mil.

El hecho de que el $SCUMBLE$ no disminuya tanto al aplicar MLD era esperable, teniendo en cuenta el funcionamiento del algoritmo. Por otra parte, es lógico que disminuya el $MeanIR$. El motivo es que MLDM, salvo en su variante general, produce nuevas instancias a partir de aquellas que presentaban etiquetas minoritarias. Por ese motivo, **todas las muestras generadas cuentan con etiquetas minoritarias**, de modo que el $MeanIR$ disminuye. Sin embargo, las mencionadas instancias con las que se entrena el o los modelos de difusión, **también pueden incluir etiquetas mayoritarias**. Por

ese motivo, las nuevas instancias sintéticas seguirán la distribución aprendida y también podrán incluir etiquetas mayoritarias, haciendo que el $SCUMBLE$ no disminuya en gran medida, pudiendo incluso aumentar con respecto al MLD original.

En definitiva, los resultados demuestran el buen funcionamiento de MLDM, obteniendo resultados mejores o, al menos, similares al resto de algoritmos, siendo competente en cuanto a la reducción del desbalanceo. Sin embargo, la calidad de un algoritmo de remuestreo no reside únicamente en la capacidad de reducir el nivel de desbalanceo del conjunto de datos original, sino que su ejecución debe llevar consigo una **mejora en el rendimiento de los clasificadores** que se apliquen al mismo. Este aspecto se analiza a continuación.

5.6.2. Impacto en la clasificación

Reducir el grado de desbalanceo de acuerdo a las métricas de descripción del conjunto de datos no es suficiente, pues los algoritmos de remuestreo pueden, en dicho proceso, incluir ruido en los datos, haciendo que el rendimiento de los algoritmos de clasificación empeore. Por este motivo, se ha incluido, como parte de la experimentación, la ejecución de clasificadores antes y después de aplicar diferentes algoritmos de remuestreo, incluido MLDM. En este apartado se analizan los resultados obtenidos.

De nuevo, para facilitar la visualización con respecto a las tablas recogidas anteriormente, se ha construido una serie de gráficas en las que se puede observar, de manera visual, el impacto que ha tenido la ejecución de cada algoritmo de remuestreo en el rendimiento de los distintos clasificadores que componen el experimento en los diferentes MLD, según las métricas presentadas en el apartado 5.1.2.

A la vista de las gráficas recogidas en las Figuras 5.3, 5.4, 5.5, 5.5 y 5.7 se puede apreciar que el algoritmo MLDM tiene un rendimiento que **iguala e incluso supera** en varias ocasiones a sus competidores. Destaca que, en aquellos casos en los que no es el mejor, se mantiene próximo al resto de algoritmos, no quedando nunca en último lugar, y siendo en la gran mayoría de las ocasiones **uno de los tres mejores métodos**.

Como observaciones generales, cabe destacar que, para la métrica $Hamming\ Loss$, los resultados no son buenos, teniendo en cuenta que se trata de una métrica a minimizar. Por otro lado, para la métrica $One\ Error$, los resultados de todos los clasificadores con la mayoría de MLD son muy buenos. En consecuencia, los valores para estas métricas **no son tan determinantes del rendimiento de los algoritmos**.

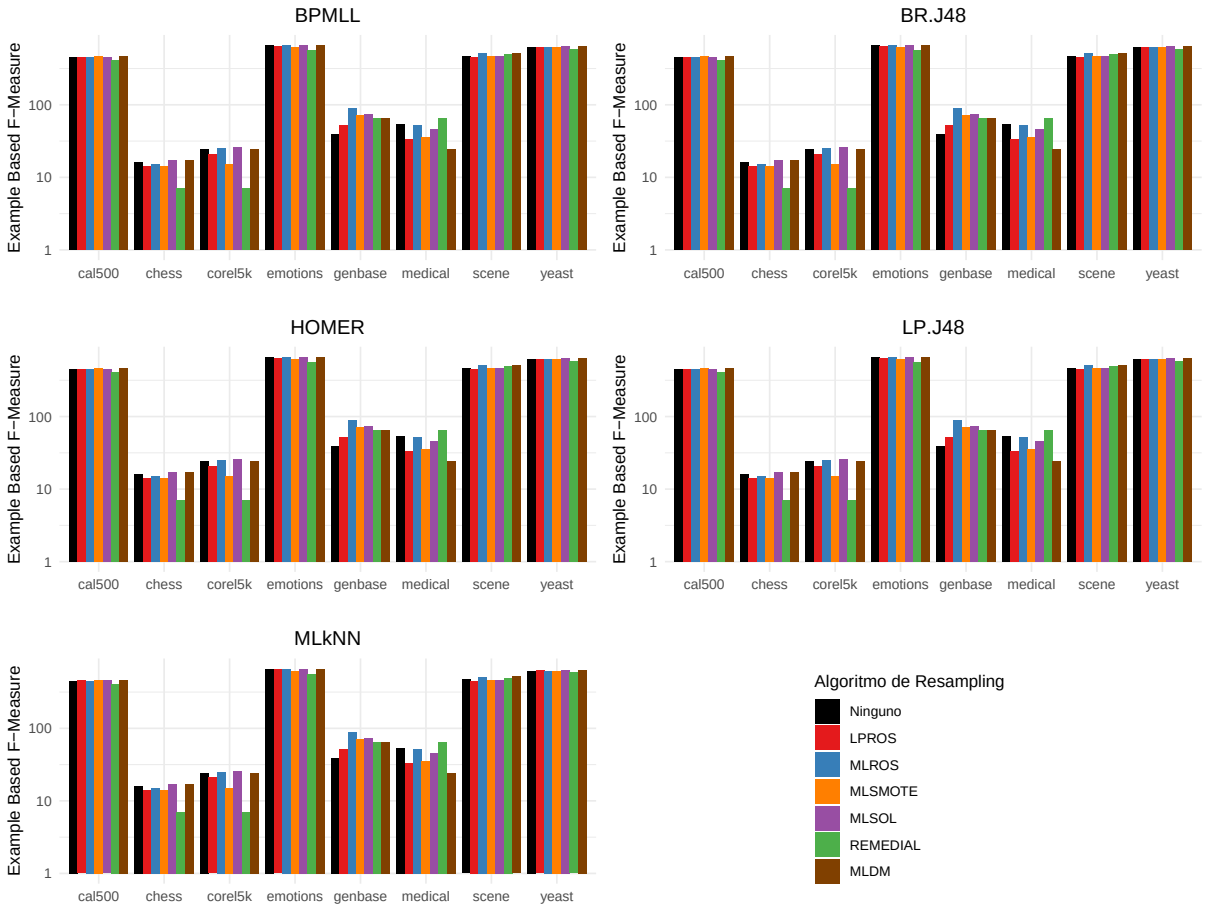

Figura 5.3: F-Measure basada en ejemplos al aplicar algoritmos, escala logarítmica, tras multiplicar por mil.

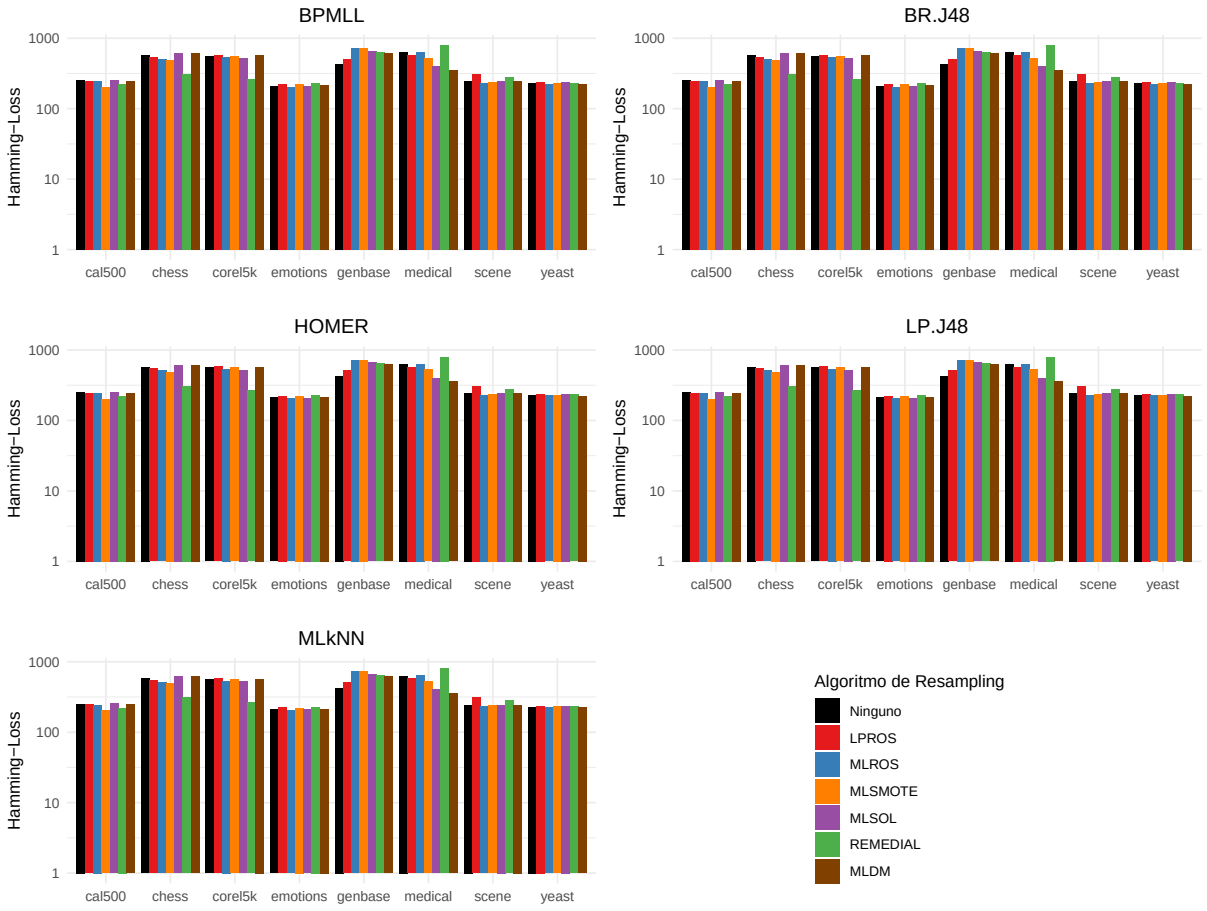

Figura 5.4: Hamming-Loss al aplicar algoritmos, escala logarítmica, tras multiplicar por mil.

De igual modo, también hay MLD para los que se obtienen mejores resultados que para otros, pudiendo esto deberse a características como su dimensionalidad y nivel de desbalanceo previo a la aplicación de los algoritmos, o bien por una mayor presencia de patrones que relacionen los atributos con las etiquetas. Estos MLD son `emotions`, `scene` y `yeast`. Todos ellos tienen de especial que **la mayoría de sus atributos son numéricos**. En cuanto a los clasificadores empleados, como se puede observar, no hay grandes diferencias de rendimiento, siendo las gráficas para una misma métrica parecidas entre sí.

Analizando el rendimiento de los algoritmos, se puede apreciar que no hay uno que funcione mucho mejor que el resto para todas las métricas y MLD, de igual modo que no hay ninguno que destaque por su mal funcionamiento. En distintas situaciones, unos son mejores que otros, pero no hay una dominancia de ninguno de ellos. Quizás puedan destacarse MLROS y MLSOL por su consistencia, manteniéndose siempre entre los tres mejores, **además del propio MLDM**.

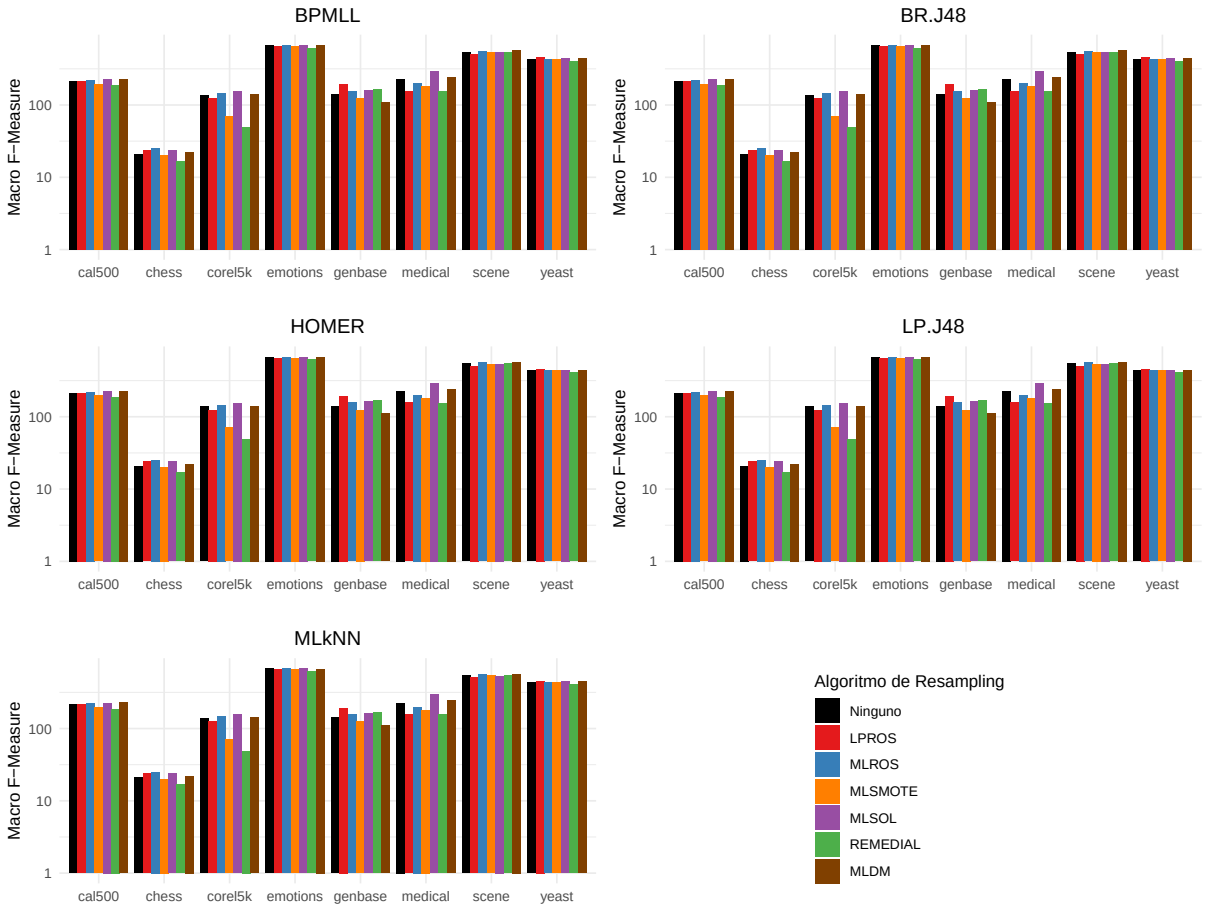

Figura 5.5: Macro F-Measure basada en ejemplos al aplicar algoritmos, escala logarítmica, tras multiplicar por mil.

Centrando la atención en MLDM, se pueden distinguir aquellos MLD en los que obtiene mejores resultados. Estos son `cal500`, `emotions`, `scene` y `yeast`. Por otro lado, aquellos en los que obtiene peores valores para las distintas métricas son `genbase` y `medical`. Se puede observar un claro patrón, y es que **MLDM funciona mejor sobre aquellos MLD que tienen un mayor número de atributos numéricos**. Este hecho es lógico, teniendo en cuenta que la estructura de MLDM es una red neuronal, y los modelos con dicha arquitectura, por lo general, trabajan mejor con valores numéricos.

Por otro lado, para MLD en los que la mayoría de atributos son categóricos, otros algoritmos como MLSOL son mejores, al emplear métricas especiales en el cálculo de distancias. Sin embargo, MLDM no obtiene resultados muy bajos aún en MLD mayoritariamente discretos, sino que se mantiene al nivel de la mayoría de algoritmos, lo cual demuestra que la codificación *one-hot* permite buenos resultados.

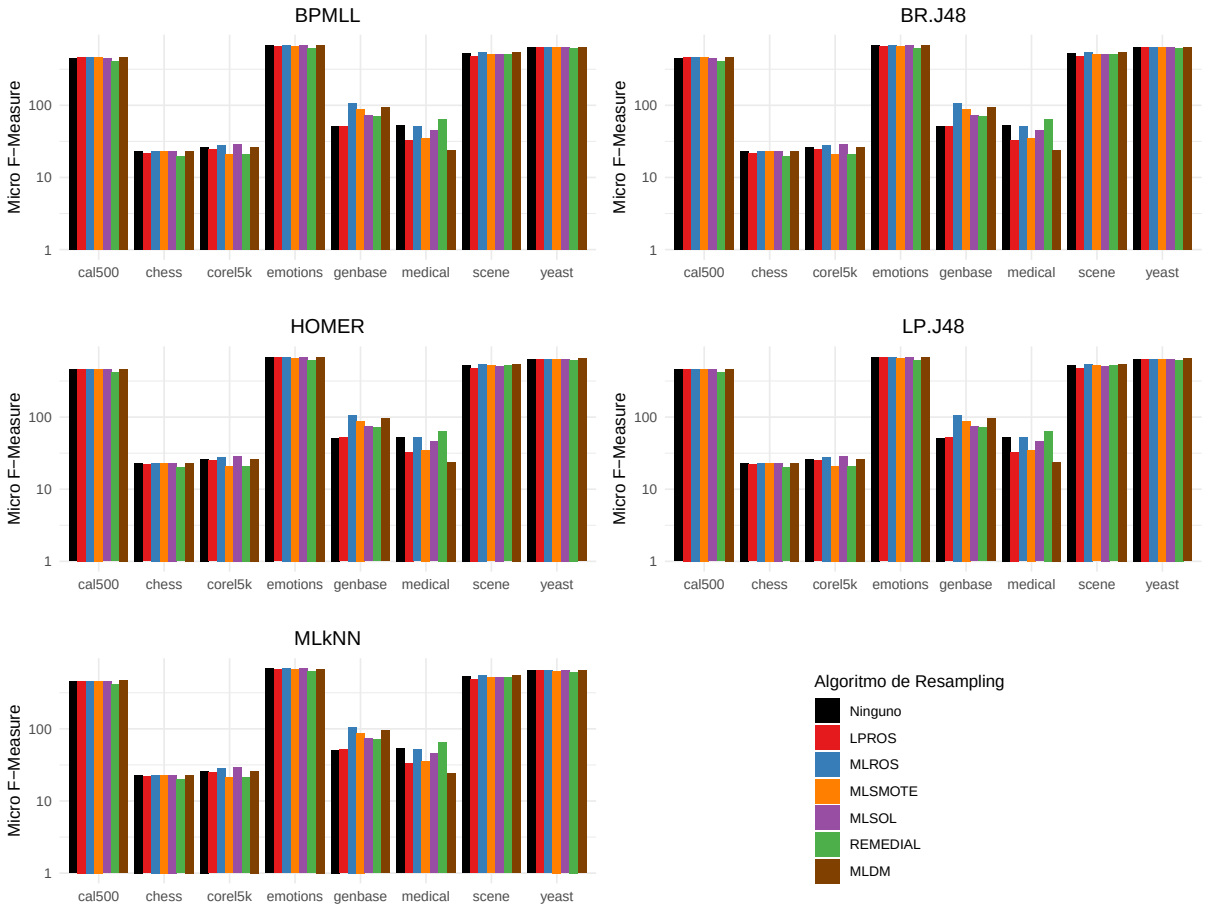

Figura 5.6: Micro F-Measure basada en ejemplos al aplicar algoritmos, escala logarítmica, tras multiplicar por mil.

5.6.3. Tiempo de ejecución

Como se puede apreciar en la Tabla 5.13, los algoritmos de sobremuestreo aleatorio, LPROS y MLROS, son muy rápidos, manteniéndose, por lo general, por debajo del segundo de ejecución, y tan solo superándolo en MLD de alta dimensionalidad, como `corel5k`, en el que alcanzan los cuatro segundos. Lo mismo ocurre con el método REMEDIAL, que tiene unos tiempos de ejecución similares. Se trata de los algoritmos más sencillos pero, como se ha visto anteriormente, pueden obtener resultados adecuados.

El algoritmo más lento es MLSOL, al ser, junto con MLDM, el más complejo. Para MLD con baja dimensionalidad, como `emotions`, el tiempo de ejecución es de unos cien segundos. Sin embargo, el aumento de la dimensionalidad del MLD al que se aplica el método conlleva un gran incremento del tiempo de ejecución, ya que MLSOL implica un proceso de cálculo de vecinos más cercanos para todas las instancias. Este es el motivo por el que, para MLD con un mayor número de instancias y atributos,

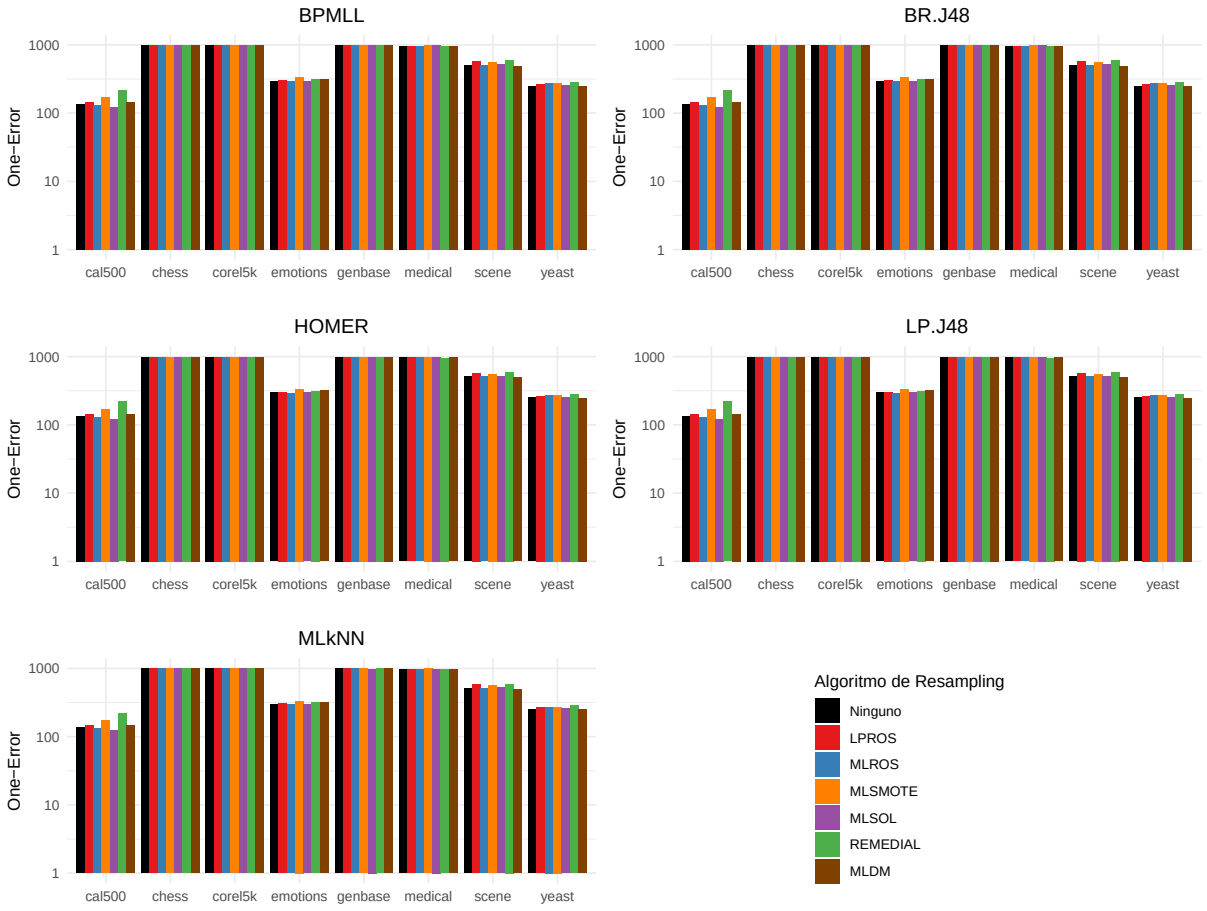

Figura 5.7: One-Error al aplicar algoritmos, escala logarítmica, tras multiplicar por mil.

especialmente categóricos (al ser el cálculo de la distancia más complejo que para los numéricos), el tiempo de ejecución puede ascender a los cien mil segundos, como es el caso de `corel5k`. Por su parte, MLSMOTE, aunque también lleva a cabo cálculo de vecinos, no lo hace para todas las instancias del MLD, sino únicamente para aquellas con etiquetas minoritarias. Por tanto, los tiempos son bajos, del orden de los diez segundos para MLD de baja dimensionalidad, pero pueden llegar a los cientos, incluso miles, de segundos para MLD más grandes.

En cuanto a MLDM, **sus tiempos de ejecución varían poco entre MLD**, manteniéndose siempre entre los 200 y los 300 segundos. Por ese motivo, en MLD de baja dimensionalidad es uno de los algoritmos más lentos, pero conforme aumenta la dimensionalidad, al mantenerse el tiempo, **consigue ser mucho más rápido que MLSOL**, con tiempos similares a los de MLSMOTE.

A la vista de los tiempos de ejecución de cada algoritmo y al rendimiento de los mismos, podemos concluir que los enfoques básicos como LPROS, MLROS y REMEDIAL son ideales cuando se requieren tiempos de ejecución mínimos en MLD con alta

dimensionalidad. Sin embargo, si se buscan mejores resultados, manteniendo tiempos aceptables, MLDM es una excelente opción.

En definitiva, **los resultados sitúan a MLDM como un algoritmo a la altura de otros métodos del estado del arte**, siendo preferible a otros en algunos casos, debido a sus buenos resultados en un tiempo de ejecución moderado.

Capítulo 6

CONCLUSIONES

En este trabajo de fin de grado se ha llevado a cabo la **propuesta de un nuevo algoritmo** de sobremuestreo para conjuntos de datos multietiqueta, basado en modelos de difusión, llamado MLDM.

Para conseguir los objetivos propuestos, ha tenido lugar una **amplia revisión del estado del arte** en el ámbito del aprendizaje multietiqueta, métodos de remuestreo, modelos generativos de aprendizaje profundo y, en particular, modelos de difusión. Dicha revisión ha sido útil también para el **desarrollo de un paquete de R**, que recopila los algoritmos propuestos dentro del mencionado ámbito.

El paquete de R mencionado, llamado `mldr.resampling`, implementa **once algoritmos de remuestreo** para datos multietiqueta del estado del arte, mejorando su eficiencia mediante un sistema de caché para el cálculo de vecinos, la vectorización de operaciones, en lugar del uso de bucles, y la posibilidad de paralelización de la ejecución. El paquete **está disponible en el repositorio CRAN**, y su documentación [95] está sometida a un proceso de revisión para ser publicada en la revista *Neurocomputing*, que se encuentra en el **primer cuartil del JCR** (*journal citation reports*).

De igual manera, **se han estudiado los nuevos modelos generativos**, basados en arquitecturas de redes neuronales, usados mayoritariamente para generar imágenes sintéticas. Este estudio ha permitido concluir que los modelos de difusión son los que tienen un mejor rendimiento, al menos en imágenes. Por ese motivo, han sido estos tipos de modelos los que han sido seleccionados como base de la propuesta desarrollada.

Esta adaptación ha dado lugar a MLDM, un **algoritmo de remuestreo multietiqueta basado en modelos de difusión**, capaz de generar instancias sintéticas que

sigan la distribución de los datos originales, y presenten etiquetas minoritarias, para así reducir el nivel de desbalanceo del MLD.

El método MLDM cuenta con diversos parámetros que dan lugar a variaciones del mismo. Por tanto, también se ha llevado a cabo una evaluación de las mismas para elegir la mejor. Este análisis se ha basado en el rendimiento de cinco clasificadores multietiqueta del estado del arte antes y después de aplicar cada variación. El rendimiento de los clasificadores se ha estimado atendiendo a los valores de cinco métricas distintas.

Una vez seleccionada la mejor variante de MLDM, esta se ha comparado con los mejores algoritmos de sobremuestreo de datos multietiqueta del estado del arte. Los aspectos medidos han sido, aparte del rendimiento de clasificadores explicado en el párrafo anterior, las métricas de caracterización $MeanIR$ y $SCUMBLE$ antes y después de aplicar cada algoritmo, y el tiempo de ejecución de cada método.

El proceso de experimentación ha constatado el **buen funcionamiento de MLDM**, que obtiene resultados satisfactorios en tiempos de ejecución relativamente cortos y constantes. En resumen, se trata de una propuesta que **iguala e incluso mejora al resto algoritmos del estado del arte**.

6.1. Valoración personal

Esta memoria ha sido la culminación de un trabajo de dos años, que comenzó como una Beca Ícaro en la que me introduje en el amplio campo de la minería de datos, centrándome en la clasificación multietiqueta y, finalmente, explorando el vasto campo de los modelos generativos. Este trabajo continuó con una Beca de Colaboración del Ministerio de Educación y Formación Profesional, llevada a cabo con el grupo de investigación Sistemas inteligentes y Minería de Datos. Esta colaboración tuvo como resultado el desarrollo del ya referenciado paquete `mldr.resampling`, proceso en el cual pude aprender aún más sobre algoritmos de remuestreo multietiqueta. Gracias a este paquete, he tenido la oportunidad de estudiar las propuestas existentes para desbalanceo en clasificación multietiqueta e implementar las más importantes, incluyéndolas en el paquete. Finalmente, comencé el estudio de los modelos de difusión y el desarrollo del método MLDM, formando todo ello parte de mi trabajo fin de grado

A lo largo de estos dos años he aprendido muchísimo acerca del proceso de investigación. La documentación y el estudio de propuestas similares a la que se quiere llevar a cabo, las decisiones de diseño, la resolución de los problemas encontrados, la

experimentación y análisis de resultados, la documentación del proceso, etc. Se trata de un trabajo duro, a veces tedioso, pero muy satisfactorio cuando se consiguen los objetivos propuestos.

También he tenido la oportunidad de usar herramientas muy importantes en el ámbito de la ciencia de datos, en el que quiero orientar mi carrera profesional. Los lenguajes de programación `R` y `Python`, probablemente dos de los más usados en este campo, han sido elementos fundamentales en el desarrollo del paquete y el TFG. El *framework* PyTorch, que, junto con TensorFlow, es el más usado en el campo del aprendizaje profundo, se me ha presentado como una herramienta muy potente, de la cuál aún solo domino una pequeña parte.

Por otro lado, programas como Docker o Conda, que permiten la virtualización de entornos de trabajo, a mayor y menor escala, respectivamente, me han permitido llevar a cabo la experimentación asociada al trabajo en un servidor remoto de altas prestaciones, y me he apoyado en ellos para conseguir una mejor reproducibilidad de dicha experimentación. También he aprendido mucho sobre la tecnología CUDA, indispensable para el aprovechamiento de las capacidades del servidor; en concreto, de su tarjeta gráfica.

Por último, el proceso de documentación de un trabajo tan extenso como este, también ha supuesto una novedad para mí. Es la primera vez que escribo una memoria de más de cien páginas, acostumbrado a las escritas como parte de las asignaturas del grado. A este respecto, también debo destacar que he podido familiarizarme con el entorno LaTeX, que es una de las mejores alternativas para la escritura de documentos, en especial cuando estos cuentan con tantas tablas, ecuaciones y figuras. No me cabe duda de que se convertirá en una de las herramientas de uso habitual en mi trabajo, al igual que las mencionadas anteriormente.

En definitiva, el desarrollo de este trabajo ha supuesto para mí un reto que me ha llevado a aprender herramientas y técnicas que me servirán en el futuro. Igualmente, me ha permitido desenvolverme en un ambiente de trabajo también novedoso, colaborando con compañeros y profesores para resolver los problemas que se me han planteado. **Sin duda, en conjunto ha sido una experiencia muy gratificante y significativa para mí, y que recordaré siempre**.

6.2. Trabajo futuro

A pesar de que los resultados obtenidos han sido de gran calidad, aún se puede ahondar en esta línea de investigación, intentando encontrar la **aplicación óptima de los modelos de difusión a datos multietiqueta**.

Una de las líneas de trabajo abierto es el desarrollo de un modelo de difusión para datos multietiqueta **basado en otras propuestas** para datos tabulares publicadas durante la realización de este trabajo, que podrían conseguir mejores resultados. Un ejemplo es el modelo SOS [155], que toma un enfoque diferente, al tratarse de un modelo *Score based*, una variación de los modelos de difusión.

Por otro lado, en lugar de adaptar los modelos de difusión a MLD, otro enfoque puede ser el de **adaptar los MLD**, para que tengan formato de imagen y así poder usar un modelo de difusión clásico, sin necesidad de llevar a cabo modificaciones sobre el mismo. La transformación de datos tabulares en imágenes [156, 157] ha sido objeto de estudio, a raíz del buen funcionamiento de las redes convolucionales en la clasificación de imágenes. Dicha transformación tiene en cuenta correlaciones entre atributos, usándolas como guía para el posicionamiento de los valores en píxeles. De esta manera, atributos con alta correlación estarán en píxeles contiguos, y atributos poco correlacionados, en píxeles lejanos.

De igual manera, en lugar de tomar como base adaptaciones a datos tabulares de modelos de difusión ya existentes, se podría aprovechar el conocimiento adquirido acerca del funcionamiento de los modelos de difusión originales para diseñar una **adaptación desde cero**, teniendo en mente desde un primer momento a los datos multietiqueta como destinatarios de la adaptación.

No obstante, puesto que los resultados obtenidos han sido muy satisfactorios, el objetivo una vez finalizado este trabajo es **publicar la propuesta de MLDM en una revista científica**, al contar con una amplia experimentación que avala su buen funcionamiento.

Apéndice A

Configuración de la experimentación

Como se ha explicado en el Apartado 4.3.4, la experimentación se ha llevado a cabo en un **servidor con una gran capacidad computacional**. En este apéndice se detallan las operaciones llevadas a cabo sobre el mismo, a fin de posibilitar la ejecución de los diferentes algoritmos, tanto de sobremuestreo como de clasificación, que conforman la experimentación de este trabajo.

A.1. Paralelización del código R en la máquina

La complejidad computacional de muchos de los algoritmos implementados en el mencionado paquete `mldr.resampling`, que han sido ejecutados para la obtención de resultados de cara a una comparación con el método propuesto, era elevada, especialmente en aquellos que implicaban **cálculo de vecinos**.

Debido a ello, surge el interés de recurrir a la **paralelización del código** de los algoritmos, para así aprovechar los 40 hilos con los que cuenta el servidor en el que se llevó a cabo la ejecución. No obstante, este proceso trajo consigo varias complicaciones, las cuales se detallan en este apartado.

El paso más directo es la edición del código en `R` del paquete, a fin de llevar a cabo la mencionada paralelización. Para ello, se recurrió al paquete `parallel`. Dicho paquete incluye las funciones `mclapply` y `parLapply`, que permiten la ejecución de distintas iteraciones de una función en procesos diferentes. La primera únicamente funciona en sistemas Unix, mientras que la segunda también funciona en Windows.

Una vez paralelizado el código, se lanzó una ejecución en la máquina, la cual **únicamente aprovechó la capacidad de uno de los 40 hilos del procesador**; es decir, no se estaba paralelizando. Este hecho se evidenció al ejecutar el programa `nmon`, que muestra, entre otros parámetros, el porcentaje de uso de la CPU. La salida de dicho programa se recoge en la Figura A.1. Como se puede apreciar en dicha figura, tan solo el hilo 37 está siendo utilizado por el programa en ejecución.

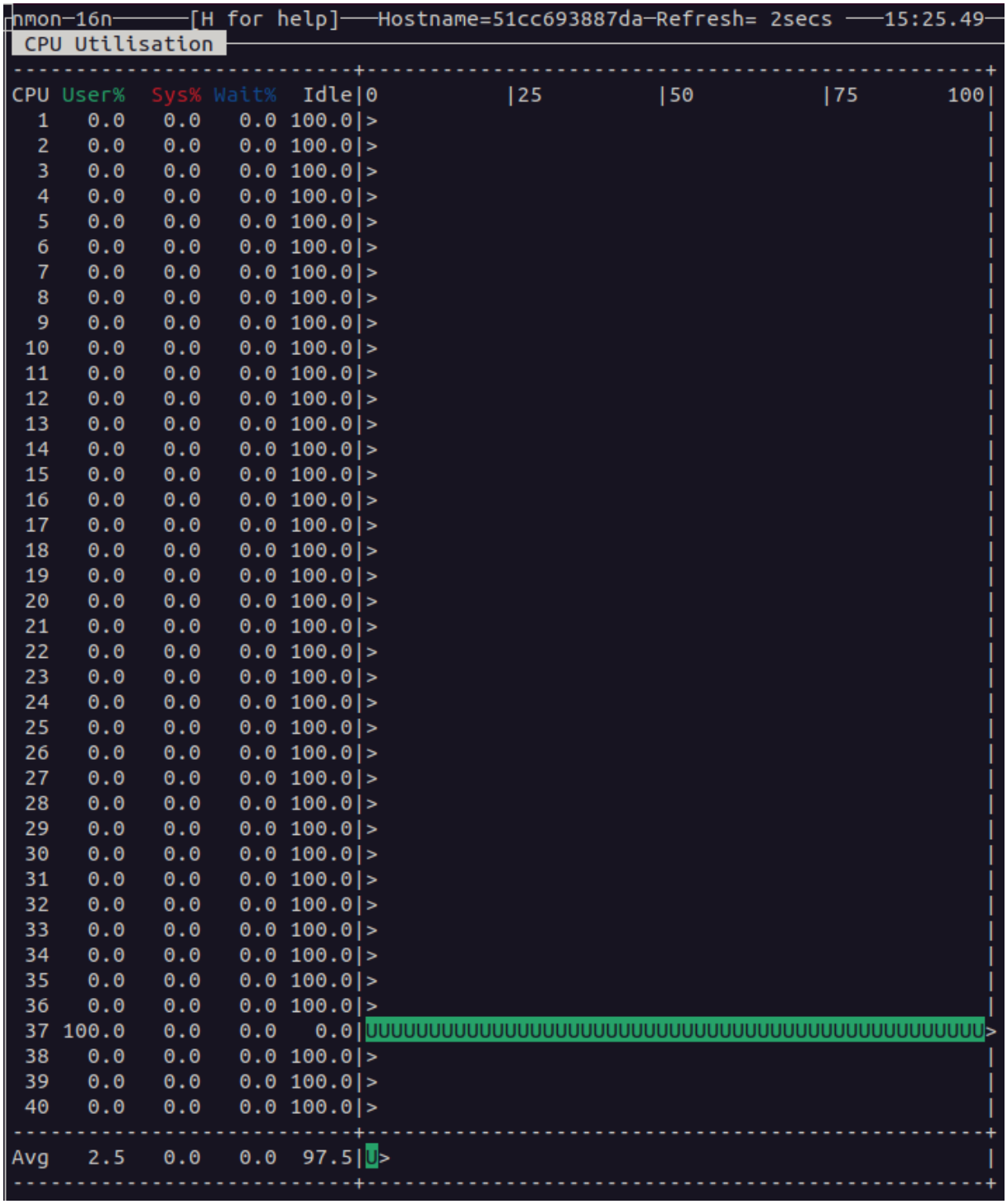

Figura A.1: Salida del programa nmon con la imagen de docker r-base.

Como se ha indicado en el Apartado 4.3.4, **la máquina tiene instalado Docker** [158]. En concreto, la ejecución se llevó a cabo sobre un contenedor construido a partir de la imagen `r-base` [159], concebida para la ejecución de código en `R`. No obstante, dicha imagen **no contemplaba el uso de los mecanismos de paralelización** previamente mencionados.

Investigando sobre este tema, se descubrió la imagen de Docker `r-parallel` [160], desarrollada por la misma organización que `r-base`: *rocker*; y que, como su nombre indica, **sí que fue creada con el propósito de llevar a cabo ejecuciones paralelas** en `R`. Solo había un problema, y es que la imagen dejó de tener soporte por parte de los desarrolladores hace tres años. La consecuencia de este hecho fue un **desfase del contenedor** generado, siendo imposible actualizar o instalar software debido a problemas con dependencias cruzadas.

La solución al problema pasó por examinar qué diferencias existían entre las dos imágenes, en cuanto a paquetes de `R` y dependencias de sistema operativo. Esto fue posible gracias a que las imágenes de Docker están definidas por un archivo llamado *Dockerfile*, en el que se especifica qué comandos se ejecutan sobre un contenedor base o sobre el propio sistema operativo (en este caso, sobre la mencionada imagen `r-base`). Gracias a dicho archivo, se pudieron obtener los comandos a ejecutar. Una vez ejecutados, **el contenedor estaba preparado para llevar a cabo computación paralela**, como se puede apreciar en la Figura A.2. De este modo, se consiguió reducir los tiempos de los experimentos en gran medida.

A.2. Instalación de dependencias para el funcionamiento de MLDM

Una vez ejecutados los algoritmos de remuestreo implementados en el paquete `mldr.resamling`, fue momento de ejecutar también el algoritmo MLDM, propuesto en este trabajo. Como se ha mencionado en el apartado 4.2.3, dicho algoritmo se ha implementado en el lenguaje de programación `Python`, usando el *framework* PyTorch.

Dado que se trata de una adaptación de un modelo anterior, se han respetado las dependencias del mismo. Debido a ello, la configuración para hacer posible la experimentación no ha sido trivial, al entrar en juego aspectos como los **controladores de la tarjeta gráfica de Nvidi**a, necesarios para el funcionamiento del algoritmo debido a la dependencia entre PyTorch y la herramienta CUDA [140].

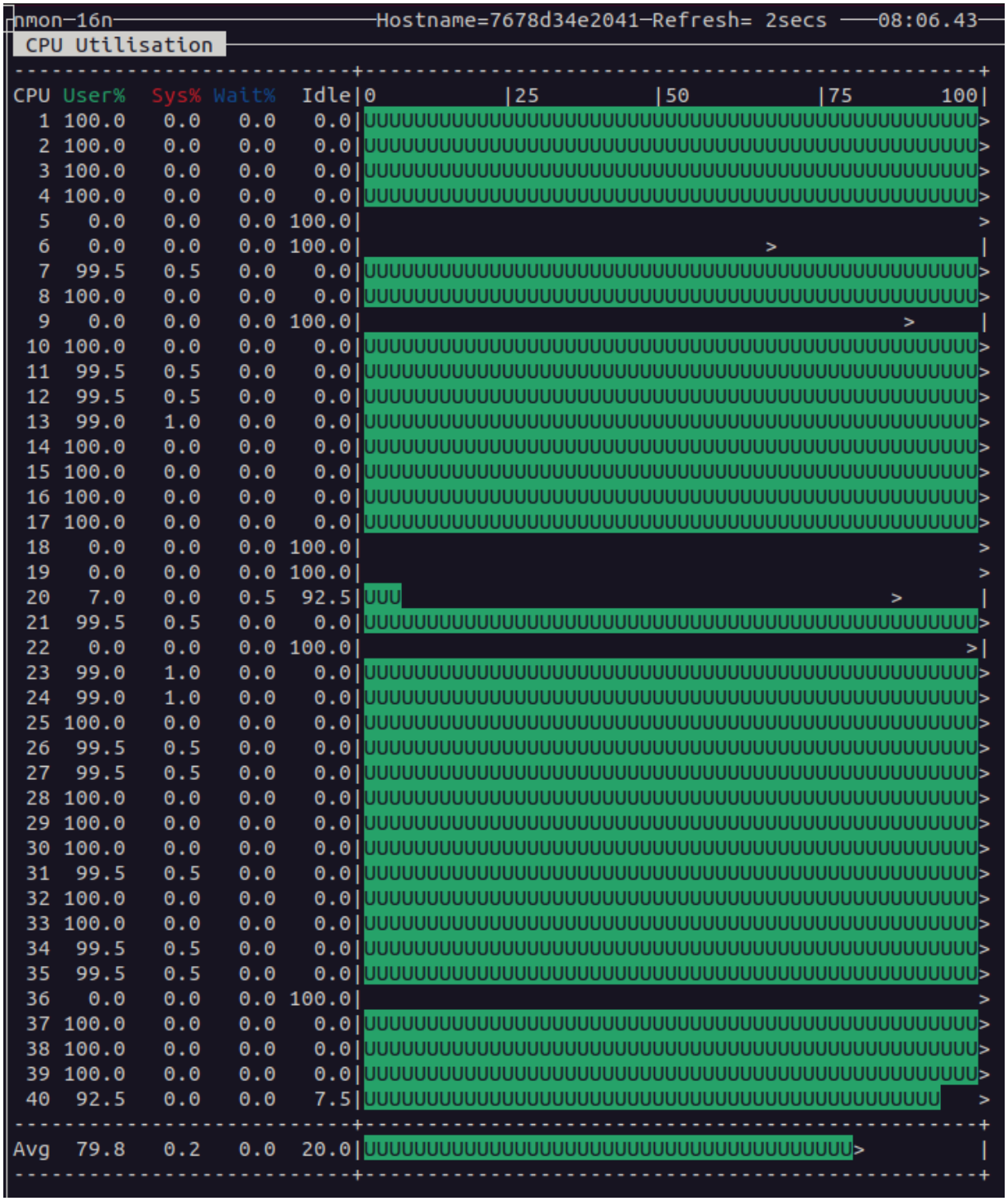

Figura A.2: Salida del programa nmon con la imagen de docker r-parallel.

En primer lugar, se intentaron instalar las distintas dependencias en una máquina que ya contaba con `Python` y CUDA, en versiones diferentes a la requerida por el programa. Como era de esperar, el código y muchas de las dependencias **no eran compatibles** con dichas versiones, por lo que se intentó cambiar a una versión más antigua de CUDA. Sin embargo, **este proceso es muy complejo**, por la mencionada influencia de los controladores de Nvidia.

Por ese motivo, se optó por una **instalación desde cero en una imagen de Docker**. Esto facilitó el manejo de CUDA, al instalar desde el principio los controladores y la versión requerida por el código del algoritmo. Por otro lado, para gestionar `Python` y sus dependencias, incluyendo el propio PyTorch, se creó un entorno virtual de Conda, de modo que tanto la versión de `Python` como la de las dependencias eran las correctas. Este paso podría ser omitido en favor de la instalación de las dependencias sobre la propia imagen de Docker, pero se consideró que un entorno virtual permitiría una mejor gestión de dichas dependencias.

De este modo, para ejecutar el algoritmo en el servidor **tan solo se tuvo que exportar la imagen de Docker creada e instalarla en dicho servidor**, dando lugar a un contenedor con todas las dependencias instaladas y listo para llevar a cabo la ejecución.

A.3. Ejecución del framework MULAN en la máquina

Una vez obtenidos los MLD generados por los métodos de sobremuestreo, se llevó a cabo la ejecución de los algoritmos de clasificación, que permitieron comparar el comportamiento de los modelos mencionados anteriormente. Dichos algoritmos están recopilados en la herramienta MULAN [132], una biblioteca de `Java` especializada en MLC.

De este modo, para la ejecución de los clasificadores, era necesario contar con una máquina con `Java` instalado. De nuevo, **se recurrió a una imagen de Docker**, que se instaló en el servidor mencionado en el apartado anterior. En concreto, tras investigar acerca de las posibles imágenes ya creadas, se optó por usar aquellas del equipo Eclipse Temurin [161].

Sin embargo, el proceso no fue tan sencillo como descargar la última versión, sino que, para que MULAN pudiera funcionar correctamente, fue necesario buscar una imagen que tuviera instalada la versión 8 de `Java`, **en su versión de desarrollo**, para po-

der compilar el script de ejecución. En concreto, se eligió la imagen *eclipse-temurin:8-jdk-alpine*.

Una vez descargada y desplegada la imagen de Docker, se usó un *script* de `Java` [1] en el que se automatiza el proceso de carga del MLD, ejecución de clasificadores y obtención de métricas.

[1] Código en GitHub: https://github.com/fcharte/SM-MLC/tree/master/RunMLClassifier

Bibliografía

[1] Francisco Charte Ojeda. *Nuevos métodos híbridos de computación flexible para clasificación multietiqueta*. Tesis doctoral, Departamento de Ciencias de la Computación e Inteligencia Artificial. Universidad de Granada, 05 2015.

[2] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37, Mar. 1996. doi: 10.1609/aimag.v17i3.1230. URL https://ojs.aaai.org/index.php/aimagazine/article/view/1230.

[3] Shichao Zhang, Chengqi Zhang, and Qiang Yang. Data preparation for data mining. *Applied Artificial Intelligence*, 17:375–381, 05 2003. doi: 10.1080/713827180.

[4] Ratnadeep Deshmukh and Vaishali Wangikar. Data cleaning: Current approaches and issues. pages 61–66, 01 2011.

[5] Jerzy W Grzymala-Busse and Witold J Grzymala-Busse. Handling missing attribute values. In *Data Mining and Knowledge Discovery Handbook*, 2010.

[6] Shivani Gupta and Atul Gupta. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474, 01 2019. doi: 10.1016/j.procs.2019.11.146.

[7] Denis Cousineau and Sylvain Chartier. Outliers detection and treatment: A review. *International Journal of Psychological Research*, 3, 06 2010. doi: 10.21500/20112084.844.

[8] Ying Yang, Geoffrey I Webb, and Xindong Wu. *Discretization Methods*, pages 113–130. Springer US, Boston, MA, 2005. ISBN 978-0-387-25465-4. doi: 10.1007/0-387-25465-X_6. URL https://doi.org/10.1007/0-387-25465-X_6.

[9] Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524, 2020. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2019.105524. URL https://www.sciencedirect.com/science/article/pii/S1568494619302947.

[10] Bouchlaghem, Younes, Akhiat, Yassine, and Amjad, Souad. Feature selection: A review and comparative study. *E3S Web Conf.*, 351:01046, 2022. doi: 10.1051/e3sconf/202235101046. URL https://doi.org/10.1051/e3sconf/202235101046.

[11] J Arturo Olvera-López, J Ariel Carrasco-Ochoa, J Francisco Martínez-Trinidad, and Josef Kittler. A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143, Aug 2010. ISSN 1573-7462. doi: 10.1007/s10462-010-9165-y. URL https://doi.org/10.1007/s10462-010-9165-y.

[12] Vladimir Nasteski. An overview of the supervised machine learning methods. *HORIZONS.B*, 4:51–62, 12 2017. doi: 10.20544/HORIZONS.B.04.1.17.P05.

[13] Zoubin Ghahramani. *Unsupervised Learning*, pages 72–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-28650-9. doi: 10.1007/978-3-540-28650-9_5. URL https://doi.org/10.1007/978-3-540-28650-9_5.

[14] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. In *Introduction to Semi-Supervised Learning*, 2009.

[15] Rui Nian, Jinfeng Liu, and Biao Huang. A review on reinforcement learning: Introduction and applications in industrial process control. *Computers Chemical Engineering*, 139:106886, 2020. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2020.106886. URL https://www.sciencedirect.com/science/article/pii/S0098135420300557.

[16] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. URL https://arxiv.org/abs/2204.06125. Fecha de último acceso: 5-3-2023.

[17] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165. Fecha de último acceso: 5-3-2023.

[18] Pedro Antonio Gutiérrez and Salvador García. Current prospects on ordinal and monotonic classification. *Progress in Artificial Intelligence*, 5(3):171–179, 2016.

[19] Norman R Draper and Harry Smith. *Applied Regression Analysis*. Wiley Series in Probability and Statistics. Wiley, 1998. ISBN 9780471170822. URL https://books.google.es/books?id=d6NsDwAAQBAJ.

[20] Tak-chung Fu. A review on time series data mining. *Eng. Appl. Artif. Intell.*, 24(1): 164–181, feb 2011. ISSN 0952-1976. doi: 10.1016/j.engappai.2010.09.007. URL https://doi.org/10.1016/j.engappai.2010.09.007.

[21] Lior Rokach. *A survey of Clustering Algorithms*, pages 269–298. Springer US, Boston, MA, 2010. ISBN 978-0-387-09823-4. doi: 10.1007/978-0-387-09823-4_14. URL https://doi.org/10.1007/978-0-387-09823-4_14.

[22] Frank Höppner. *Association Rules*, pages 299–319. Springer US, Boston, MA, 2010. ISBN 978-0-387-09823-4. doi: 10.1007/978-0-387-09823-4_15. URL https://doi.org/10.1007/978-0-387-09823-4_15.

[23] Stefan Edelkamp, Stefan Schroedl, and Sven Koenig. *Heuristic Search: Theory and Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. ISBN 0123725127.

[24] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5): 8091–8126, Feb 2021. ISSN 1573-7721. doi: 10.1007/s11042-020-10139-6. URL https://doi.org/10.1007/s11042-020-10139-6.

[25] John R Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, Mar 1986. ISSN 1573-0565. doi: 10.1007/BF00116251. URL https://doi.org/10.1007/BF00116251.

[26] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995. ISSN 1573-0565. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018.

[27] Thomas M Cover and Peter E Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 13:21–27, 1967.

[28] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[29] Bart Goethals. *Frequent Set Mining*, pages 377–397. Springer US, Boston, MA, 2005. ISBN 978-0-387-25465-4. doi: 10.1007/0-387-25465-X_17. URL https://doi.org/10.1007/0-387-25465-X_17.

[30] Norio Shiratori, Goutam Chakraborty, and Kenji Sugawara. Flexible computing: basic concepts, design and application. In *Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 152–159, 1995. doi: 10.1109/FTDCS.1995.524980.

[31] Arthur P Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society: Series B (Methodological)*, 30(2):205–232, 1968.

[32] Anders Krogh. What are artificial neural networks? *Nature biotechnology*, 26(2): 195–197, 2008.

[33] Gisele L Pappa and Alex A Freitas. *Evolutionary Algorithms*, pages 47–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-02541-9. doi: 10.1007/978-3-642-02541-9_3. URL https://doi.org/10.1007/978-3-642-02541-9_3.

[34] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[35] Lotfi A Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(65)90241-X. URL https://www.sciencedirect.com/science/article/pii/S001999586590241X.

[36] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook, 2nd ed*. 01 2010. ISBN 9780387098227.

[37] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. *Mining Multi-label Data*, pages 667–685. Springer US, Boston, MA, 2010. ISBN 978-0-387-09823-4. doi: 10.1007/978-0-387-09823-4_34. URL https://doi.org/10.1007/978-0-387-09823-4_34.

[38] Min-Ling Zhang, Yu-Kun Li, Xu-Ying Liu, and Xin Geng. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12(2):191–202, Apr 2018. ISSN 2095-2236. doi: 10.1007/s11704-017-7031-7. URL https://doi.org/10.1007/s11704-017-7031-7.

[39] Wimukthi Madhusanka. Multi-label classification, Aug 2021. URL https://wimukthimadhusanka85.medium.com/multi-label-classification-e259896182da. Fecha de último acceso: 17-12-2022.

[40] Francisco Herrera, Francisco Charte, Antonio J Rivera, and María J del Jesus. *Multilabel Classification: Problem Analysis, Metrics and Techniques*. Springer, 2016. ISBN 978-3-319-41110-1. doi: 10.1007/978-3-319-41111-8.

[41] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

[42] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[43] Sanjay Yadav and Sanyam Shukla. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In *2016 IEEE 6th International conference on advanced computing (IACC)*, pages 78–83. IEEE, 2016.

[44] Michael W Browne. Cross-validation methods. *Journal of mathematical psychology*, 44(1):108–132, 2000.

[45] Tzu-Tsung Wong. Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. *Pattern Recognition*, 48(9):2839–2846, 2015.

[46] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[47] Mehrdad Fatourechi, Rabab K Ward, Steven G Mason, Jane Huggins, Alois Schlögl, and Gary E Birch. Comparison of evaluation metrics in classification applications with imbalanced datasets. In *2008 Seventh International Conference on Machine Learning and Applications*, pages 777–782, 2008. doi: 10.1109/ICMLA.2008.34.

[48] Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, pages 22–30, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24775-3.

[49] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In Joost N Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Machine Learning: ECML 2007*, pages 406–417, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74958-5.

[50] Francisco Charte. A comprehensive and didactic review on multilabel learning software tools. *IEEE Access*, 8:50330–50354, 2020.

[51] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intell. Data Anal.*, 6:429–449, 11 2002. doi: 10.3233/IDA-2002-6504.

[52] Alberto Fernández, Victoria López, Mikel Galar, María J del Jesus, and Francisco Herrera. Analysing the classification of imbalanced data-sets with multiple classes: Binarization techniques and ad-hoc approaches. *Knowledge-Based*

Systems, 42:97–110, 2013. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2013.01.018. URL https://www.sciencedirect.com/science/article/pii/S0950705113000300.

[53] Minlong Lin, Ke Tang, and Xin Yao. Dynamic sampling approach to training neural networks for multiclass imbalance classification. *IEEE Transactions on Neural Networks and Learning Systems*, 24(4):647 – 660, 2013. doi: 10.1109/TNNLS.2012.2228231. Cited by: 122.

[54] Albert Orriols-Puig and Ester Bernadó-Mansilla. Evolutionary rule-based systems for imbalanced data sets. *Soft Comput.*, 13:213–225, 02 2009. doi: 10.1007/s00500-008-0319-7.

[55] Jonathan Ortigosa-Hernández, Iñaki Inza, and Jose A Lozano. Measuring the class-imbalance extent of multi-class problems. *Pattern Recognition Letters*, 98:32–38, 2017. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2017.08.002. URL https://www.sciencedirect.com/science/article/pii/S016786551730257X.

[56] Rui Zhu, Ziyu Wang, Zhanyu Ma, Guijin Wang, and Jing-Hao Xue. Lrid: A new metric of multi-class imbalance degree based on likelihood-ratio test. *Pattern Recognition Letters*, 116:36–42, 2018. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2018.09.012. URL https://www.sciencedirect.com/science/article/pii/S0167865518305907.

[57] Rui Zhu, Yiwen Guo, and Jing-Hao Xue. Adjusting the imbalance ratio by the dimensionality of imbalanced data. *Pattern Recognition Letters*, 133:217–223, 2020. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2020.03.004. URL https://www.sciencedirect.com/science/article/pii/S0167865520300829.

[58] Philip Sedgwick. Pearson's correlation coefficient. *BMJ*, 345:e4483–e4483, 07 2012. doi: 10.1136/bmj.e4483.

[59] Francisco Charte, Antonio Rivera, María José del Jesus, and Francisco Herrera. A first approach to deal with imbalance in multi-label datasets. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 150–160. Springer, 2013. doi: 10.1007/978-3-642-40846-5_16.

[60] Francisco Charte, Antonio Rivera Rivas, María J Del Jesus, and Francisco Herrera. Concurrence among imbalanced labels and its influence on multilabel resampling algorithms. 06 2014. ISBN 978-3-319-07616-4. doi: 10.1007/978-3-319-07617-1_10.

[61] Anthony B Atkinson. On the measurement of inequality. *Journal of Economic Theory*, 2(3):244–263, 1970. ISSN 0022-0531. doi: https://doi.org/10.1016/0022-0531(70)90039-6. URL https://www.sciencedirect.com/science/article/pii/0022053170900396.

[62] Victoria López, Alberto Fernández, Salvador García, Vasile Palade, and Francisco Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, 2013. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2013.07.007. URL https://www.sciencedirect.com/science/article/pii/S0020025513005124.

[63] Li Yijing, Guo Haixiang, Liu Xiao, Li Yanan, and Li Jinling. Adapted ensemble classification algorithm based on multiple classifier system and feature selection for classifying multi-class imbalanced data. *Knowledge-Based Systems*, 94:88–104, 2016. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2015.11.013. URL https://www.sciencedirect.com/science/article/pii/S0950705115004566.

[64] Shin Ando. Classifying imbalanced data in distance-based feature space. *Knowledge and Information Systems*, 46(3):707–730, Mar 2016. ISSN 0219-3116. doi: 10.1007/s10115-015-0846-3. URL https://doi.org/10.1007/s10115-015-0846-3.

[65] Zhiwen Liu, Hongrui Cao, Xuefeng Chen, Zhengjia He, and Zhongjie Shen. Multi-fault classification based on wavelet svm with pso algorithm to analyze vibration signals from rolling element bearings. *Neurocomputing*, 99:399–410, 2013. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2012.07.019. URL https://www.sciencedirect.com/science/article/pii/S0925231212005875.

[66] Shounak Datta and Swagatam Das. Near-bayesian support vector machines for imbalanced data classification with equal or unequal misclassification costs. *Neural Networks*, 70:39–52, 2015. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2015.06.005. URL https://www.sciencedirect.com/science/article/pii/S0893608015001264.

[67] María D Pérez-Godoy, Antonio J Rivera, Cristóbal J Carmona, and María J del Jesus. Training algorithms for radial basis function networks to tackle learning processes with imbalanced data-sets. *Applied Soft Computing*, 25:26–39, 2014. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2014.09.011. URL https://www.sciencedirect.com/science/article/pii/S1568494614004529.

[68] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, page 155–164, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581131437. doi: 10.1145/312129.312220. URL https://doi.org/10.1145/312129.312220.

[69] Jason Brownlee. Undersampling algorithms for imbalanced classification, Jan 2021. URL https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/. Fecha de último acceso: 15-01-2023.

[70] Satwik Mishra. Handling imbalanced data: Smote vs. random undersampling. *Int. Res. J Eng. Technol*, 4(8):317–320, 2017.

[71] Inderjeet Mani and Jianping Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of workshop on learning from imbalanced datasets*, volume 126, pages 1–7. ICML, 2003.

[72] Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968. doi: 10.1109/TIT.1968.1054155.

[73] Ivan Tomek. Two modifications of cnn. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(11):769–772, 1976. doi: 10.1109/TSMC.1976.4309452.

[74] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(3):408–421, 1972. doi: 10.1109/TSMC.1972.4309137.

[75] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. *Fourteenth International Conference on Machine Learning*, 06 2000.

[76] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In Silvana Quaglini, Pedro Barahona, and Steen Andreassen, editors, *Artificial Intelligence in Medicine*, pages 63–66, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-48229-1.

[77] Wei-Chao Lin, Chih-Fong Tsai, Ya-Han Hu, and Jing-Shang Jhang. Clustering-based undersampling in class-imbalanced data. *Information Sciences*, 409-410:17–26, 2017. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2017.05.008. URL https://www.sciencedirect.com/science/article/pii/S0020025517307235.

[78] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. Machine learning with oversampling and undersampling techniques: overview study and experimental results. In *2020 11th international conference on information and communication systems (ICICS)*, pages 243–248. IEEE, 2020.

[79] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16: 321–357, 2002. ISSN 1076-9757. doi: https://doi.org/10.1613/jair.953. URL https://www.jair.org/index.php/jair/article/view/10302.

[80] Lin Yu, Huang Xun, and Xu Kai. Research on extreme risk warning for financial markets based on ru-smote-svm. *Forecasting*, 4, 2013.

[81] Min Zeng, Beiji Zou, Faran Wei, Xiyao Liu, and Lei Wang. Effective prediction of three common diseases by combining smote with tomek links technique for imbalanced medical data. In *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, pages 225–228. IEEE, 2016.

[82] Xu Han, Runbang Cui, Yanfei Lan, Yanzhe Kang, Jiang Deng, and Ning Jia. A gaussian mixture model based combined resampling algorithm for classification of imbalanced credit data sets. *International Journal of Machine Learning and Cybernetics*, 10(12):3687–3699, Dec 2019. ISSN 1868-808X. doi: 10.1007/s13042-019-00953-2. URL https://doi.org/10.1007/s13042-019-00953-2.

[83] Gilles Cohen, Mélanie Hilario, Hugo Sax, Stéphane Hugonnet, and Antoine Geissbuhler. Learning from imbalanced data in surveillance of nosocomial infection. *Artificial Intelligence in Medicine*, 37(1):7–18, 2006. ISSN 0933-3657. doi: https://doi.org/10.1016/j.artmed.2005.03.002. URL https://www.sciencedirect.com/science/article/pii/S0933365705000850. Intelligent Data Analysis in Medicine.

[84] Vicente García, Josep S Sánchez, and Ramón A Mollineda. On the effectiveness of preprocessing methods when dealing with different levels of class imbalance. *Knowledge-Based Systems*, 25(1):13–21, 2012. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2011.06.013. URL https://www.sciencedirect.com/science/article/pii/S0950705111001286. Special Issue on New Trends in Data Mining.

[85] Leo Liberti, Carlile Lavor, Nelson Maculan, and Antonio Mucherino. Euclidean distance geometry and applications. *SIAM Review*, 56, 05 2012. doi: 10.1137/120875909.

[86] Craig Stanfill and David Waltz. Toward memory-based reasoning. *Commun. ACM*, 29(12):1213–1228, dec 1986. ISSN 0001-0782. doi: 10.1145/7902.7906. URL https://doi.org/10.1145/7902.7906.

[87] Hui Han, Wen-Yuan Wang, and Bing-Huan Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In De-Shuang Huang, Xiao-Ping Zhang, and Guang-Bin Huang, editors, *Advances in Intelligent Computing*, pages 878–887, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31902-3.

[88] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho, editors, *Advances in Knowledge Discovery and Data Mining*, pages 475–482, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-01307-2.

[89] Chumphol Bunkhumpornpat, Krung Sinapiromsaran, and Chidchanok Lursinsap. Dbsmote: Density-based synthetic minority over-sampling technique. *Applied Intelligence*, 36(3):664–684, Apr 2012. ISSN 1573-7497. doi: 10.1007/s10489-011-0287-y. URL https://doi.org/10.1007/s10489-011-0287-y.

[90] José A Sáez, Julián Luengo, Jerzy Stefanowski, and Francisco Herrera. Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, 291:184–203, 2015. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2014.08.051. URL https://www.sciencedirect.com/science/article/pii/S0020025514008561.

[91] Jaedong Lee, Noo-ri Kim, and Jee-Hyong Lee. An over-sampling technique with rejection for imbalanced class learning. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, IMCOM '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450333771. doi: 10.1145/2701126.2701181. URL https://doi.org/10.1145/2701126.2701181.

[92] Michal Koziarski and Michal Wozniak. Ccr: A combined cleaning and resampling algorithm for imbalanced data classification. *International Journal of Applied Mathematics and Computer Science*, 27(4):727–736, 2017. doi: doi:10.1515/amcs-2017-0050. URL https://doi.org/10.1515/amcs-2017-0050.

[93] György Kovács. An empirical comparison and evaluation of minority over-sampling techniques on a large number of imbalanced datasets. *Applied Soft*

Computing, 83:105662, 2019. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2019.105662. URL https://www.sciencedirect.com/science/article/pii/S1568494619304429.

[94] Adane Nega Tarekegn, Mario Giacobini, and Krzysztof Michalak. A review of methods for imbalanced multi-label classification. *Pattern Recognition*, 118: 107965, 2021.

[95] Antonio J Rivera, Miguel A Dávila, David Elizondo, María J del Jesus, and Francisco Charte. mldr.resampling: Efficient reference implementations of multilabel resampling algorithms, 2023. URL https://doi.org/10.48550/arXiv.2305.17152.

[96] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. Addressing imbalance in multilabel classification: Measures and random resampling algorithms. *Neurocomputing*, 163:3–16, 2015. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2014.08.091. URL https://www.sciencedirect.com/science/article/pii/S0925231215004269. Recent Advancements in Hybrid Artificial Intelligence Systems and its Application to Real-World Problems Progress in Intelligent Systems Mining Humanistic Data.

[97] Payel Sadhukhan and Sarbani Palit. Reverse-nearest neighborhood based oversampling for imbalanced, multi-label datasets. *Pattern Recognition Letters*, 125:813–820, 2019. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2019.08.009. URL https://www.sciencedirect.com/science/article/pii/S0167865519302193.

[98] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. Mlsmote: Approaching imbalanced multilabel learning through synthetic instance generation. *Knowledge-Based Systems*, 89:385–397, 2015. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2015.07.019. URL https://www.sciencedirect.com/science/article/pii/S0950705115002737.

[99] Bin Liu, Konstantinos Blekas, and Grigorios Tsoumakas. Multi-label sampling based on local label imbalance. *Pattern Recognition*, 122:108294, 2022.

[100] Francisco Charte, Antonio Rivera Rivas, María J Del Jesus, and Francisco Herrera. Resampling multilabel datasets by decoupling highly imbalanced labels. volume 9121, 06 2015. ISBN 978-3-319-19643-5. doi: 10.1007/978-3-319-19644-2_41.

[101] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. Remedial-hwr: Tackling multilabel imbalance through label decoupling

and data resampling hybridization. *Neurocomputing*, 326-327:110–122, 2019. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2017.01.118. URL https://www.sciencedirect.com/science/article/pii/S0925231217315187.

[102] José A Sáez, Bartosz Krawczyk, and Michal Wozniak. Analyzing the oversampling of different classes and types of examples in multi-class imbalanced datasets. *Pattern Recognition*, 57:164–178, 2016. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.03.012. URL https://www.sciencedirect.com/science/article/pii/S0031320316001072.

[103] Jungang Xu, Hui Li, and Shilong Zhou. An overview of deep generative models. *IETE Technical Review*, 32:131–139, 12 2014. doi: 10.1080/02564602.2014.987328.

[104] Lawrence K Saul, Tommi Jaakkola, and Michael I Jordan. Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4:61–76, 1996.

[105] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, 2006.

[106] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[107] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. *Neural Networks: Tricks of the Trade: Second Edition*, pages 599–619, 2012.

[108] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.

[109] Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 693–700. JMLR Workshop and Conference Proceedings, 2010.

[110] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 026268053X.

[111] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[112] Fernando Sancho Caparrini. Variational autoencoder, Mar 2020. URL http://www.cs.us.es/~fsancho/?e=232#Ref2. Fecha de último acceso: 5-3-2023.

[113] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL https://arxiv.org/abs/1312.6114. Fecha de último acceso: 5-3-2023.

[114] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[115] Alex Honchar. Gans beyond generation: 7 alternative use cases, Oct 2018. URL https://alexrachnog.medium.com/gans-beyond-generation-7-alternative-use-cases-725c60ba95e8.

[116] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G Willcocks. Deep generative modelling: A comparative review of VAEs,GANs, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, nov 2022. doi: 10.1109/tpami.2021.3116668. URL https://doi.org/10.1109%2Ftpami.2021.3116668.

[117] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J Goodfellow. Adversarial autoencoders. *ArXiv*, abs/1511.05644, 2015.

[118] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1530–1538, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/rezende15.html.

[119] Lilian Weng. Flow-based deep generative models, Oct 2018. URL https://lilianweng.github.io/posts/2018-10-13-flow-models/. Fecha de último acceso: 6-3-2023.

[120] Douglas A Reynolds et al. Gaussian mixture models. *Encyclopedia of biometrics*, 741(659-663), 2009.

[121] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[122] Oscar Contreras Carrasco. Gaussian mixture models explained, Feb 2020. URL https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95. Fecha de último acceso: 6-3-2023.

[123] Yuxi Xie, Min Qiu, Haibo Zhang, Lizhi Peng, and Zhenxiang Chen. Gaussian distribution based oversampling for imbalanced data classification. *IEEE*

Transactions on Knowledge and Data Engineering, 34(2):667–679, 2022. doi: 10.1109/TKDE.2020.2985965.

[124] Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. URL https://arxiv.org/abs/1503.03585. Fecha de último acceso: 6-3-2023.

[125] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

[126] Rafid Siddiqui. Diffusion models made easy, May 2022. URL https://towardsdatascience.com/diffusion-models-made-easy-8414298ce4da. Fecha de último acceso: 6-3-2023.

[127] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. URL https://arxiv.org/abs/2112.10752. Fecha de último acceso: 6-3-2023.

[128] Francisco Charte, Antonio J Rivera, David Charte, María J del Jesus, and Francisco Herrera. Tips, guidelines and tools for managing multi-label datasets: The mldr.datasets r package and the cometa data repository. *Neurocomputing*, 2018. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2018.02.011.

[129] David Charte, Francisco Charte, and Antonio J Rivera. *R Ultimate Multilabel Dataset Repository*. Universidad de Jaén, Jaén, España, 2019. URL https://cran.r-project.org/web/packages/mldr.datasets/index.html.

[130] Francisco Charte and David Charte. Working with multilabel datasets in R: The mldr package. *The R Journal*, 7(2):149–162, December 2015. URL https://journal.r-project.org/archive/2015-2/charte-charte.pdf.

[131] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[132] Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.

[133] Francisco Charte, Antonio J Rivera, María J del Jesus, and Francisco Herrera. Mlenn: a first approach to heuristic multilabel undersampling. In *Intelligent Data Engineering and Automated Learning–IDEAL 2014: 15th International Conference, Salamanca, Spain, September 10-12, 2014. Proceedings 15*, pages 1–9. Springer, 2014.

[134] Rodolfo M Pereira, Yandre MG Costa, and Carlos N Silla Jr. Mltl: A multi-label approach for the tomek link undersampling algorithm. *Neurocomputing*, 383:95–105, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.11.076. URL https://www.sciencedirect.com/science/article/pii/S0925231219316790.

[135] Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. *arXiv preprint arXiv:2209.15421*, 2022.

[136] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34:12454–12465, 2021.

[137] Jason Brownlee. How to use quantile transforms for machine learning. https://machinelearningmastery.com/quantile-transforms-for-machine-learning/. Fecha de último acceso: 11-04-2023.

[138] Kedar Potdar, Taher S Pardawala, and Chinmay D Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International journal of computer applications*, 175(4):7–9, 2017.

[139] Junyan Chen, Frédéric Dubut, Jason (Zengzhong) Li, and Rangan Majumder. Make every feature binary: A 135b parameter sparse neural network for massively improved search relevance, Aug 2021. URL https://www.microsoft.com/en-us/research/blog/make-every-feature-binary-a-135b-parameter-sparse-neural-network-for-massively- Fecha de último acceso: 14-06-2023.

[140] Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang, and Vasily Volkov. Parallel computing experiences with cuda. *IEEE micro*, 28(4):13–27, 2008.

[141] Orhan G Yalçin. Top 5 deep learning frameworks to watch in 2021. https://towardsdatascience.com/top-5-deep-learning-frameworks-to-watch-in-2021-and-why-tensorflow-98d8d6667351 Fecha de último acceso: 11-04-2023.

[142] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[143] Google. Google-trends. https://trends.google.com/trends/explore?q=pytorch,tensorflow. Fecha de último acceso: 11-04-2023.

[144] Douglas Turnbull, Luke Barrington, David Torres, and Gert Lanckriet. Semantic annotation and retrieval of music and sound effects. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):467–476, 2008.

[145] Pinar Duygulu, Kobus Barnard, João FG de Freitas, and David A Forsyth. Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary. In *Computer Vision, ECCV 2002*, volume 2353 of *LNCS*, pages 97–112. 2002.

[146] Alicja Wieczorkowska, Piotr Synak, and Zbigniew Ra's. Multi-label classification of emotions in music. In *Intelligent Information Processing and Web Mining*, volume 35, chapter 30, pages 307–315. 2006.

[147] Sotiris Diplaris, Grigorios Tsoumakas, Pericles Mitkas, and Ioannis Vlahavas. Protein classification with multiple algorithms. In *Proc. 10th Panhellenic Conference on Informatics, Volos, Greece, PCI05*, pages 448–456, 2005.

[148] Koby Crammer, Mark Dredze, Kuzman Ganchev, Partha P Talukdar, and Steven Carroll. Automatic code assignment to medical text. In *Proc. Workshop on Biological, Translational, and Clinical Language Processing, Prague, Czech Republic, BioNLP07*, pages 129–136, 2007.

[149] Matthew Boutell, Jiebo Luo, Xipeng Shen, and Cristopher Brown. Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771, 2004.

[150] Francisco Charte, Antonio J Rivera, Maria J del Jesus, and Francisco Herrera. Quinta: A question tagging assistant to improve the answering ratio in electronic forums. In *EUROCON 2015 - International Conference on Computer as a Tool (EUROCON), IEEE*, pages 1–6, Sept 2015.

[151] Andre Elisseeff and Jason Weston. A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems*, volume 14, pages 681–687, 2001.

[152] Zhou Zhi-Hua Zhang, Min-Ling. Multi-label neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 18:1338–1351, 2006.

[153] 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.

[154] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recogn.*, 40(7):2038–2048, 2007. ISSN 0031-3203.

[155] Jayoung Kim, Chaejeong Lee, Yehjin Shin, Sewon Park, Minjung Kim, Noseong Park, and Jihoon Cho. Sos: Score-based oversampling for tabular data. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, page 762–772, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539454. URL https://doi.org/10.1145/3534678.3539454.

[156] Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith A Boroevich, and Tatsuhiko Tsunoda. Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific reports*, 9(1):11399, 2019.

[157] Yitan Zhu, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A Evrard, James H Doroshow, and Rick L Stevens. Converting tabular data into images for deep learning with convolutional neural networks. *Scientific reports*, 11(1):11325, 2021.

[158] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[159] The Rocker Community. r-base - official image - docker. https://hub.docker.com/_/r-base, . Fecha de último acceso: 09-05-2023.

[160] The Rocker Community. r-parallel - official image - docker. https://hub.docker.com/r/rocker/r-parallel, . Fecha de último acceso: 09-05-2023.

[161] Eclipse Temurin. eclipse-temurin. https://hub.docker.com/_/eclipse-temurin. Fecha de último acceso: 14-06-2023.