Universidad Central Marta Abreu de Las Villas

Facultad de Matemática, Física y Computación

Licenciatura en Ciencia de la Computación

TRABAJO DE DIPLOMA

Desarrollo de clasificadores ensamblados robustos ante el problema de clases desbalanceadas en la clasificación multinstancia

Autor: César Carlos Calderón Muro

Tutor: Dr. Dánel Sánchez Tarragó

Santa Clara

2015

Agradecimientos

- **A mi mamá, por ser la persona más importante en mi vida, por dedicarme la suya, por estar siempre a mi lado a través de todos estos años. Gracias por ayudarme y nunca dudar, sin ti hubiese sido imposible.**
- **A mi papá, por sus consejos de siempre, por ser ejemplo de dedicación y superación. Gracias.**
- **A mis amigos, los que estudiaron conmigo (CAP) y los que no. Siempre han sido un gran apoyo y buena compañía. Juntos hemos pasado momentos difíciles y de alegría, momentos inolvidables.**
- **A mi tutor por servir de guía y de imprescindible ayuda. Gracias.**
- **A todos los profesores y trabajadores en general, principalmente de MFC, que me han formado espiritual y profesionalmente. Gracias.**

Resumen

Dentro del campo de estudio del aprendizaje automático, la clasificación multinstancia tiene como objetivo construir, a partir de un conjunto de ejemplos, un modelo matemático que permita clasificar objetos descritos por múltiples vectores de atributos. La clasificación sufre de varios problemas que afectan su desempeño siendo el problema de clases desbalanceadas el tratado en el presente trabajo.

El problema de clases desbalanceadas ocurre cuando hay mucha diferencia en el tamaño de las clases y provoca que el modelo de aprendizaje inducido a partir de los datos no represente adecuadamente el concepto que se pretende aprender y consecuentemente incurra en muchos errores de clasificación. En la literatura existen varias formas de tratar este problema en el escenario simple-instancia, siendo la incorporación de técnicas de muestreo dentro de clasificadores ensamblados una de las más populares y eficaces.

Los clasificadores ensamblados entrenan múltiples clasificadores base y combinan sus predicciones para clasificar una instancia de la cual no se conoce su clase. Estos presentan generalmente mejor capacidad predictiva que clasificadores individuales, siendo los algoritmos Bagging y Boosting dos de los más representativos.

Debido a que el problema de clases desbalanceadas en la clasificación multinstancia es un área joven a la cual se le ha prestado poca atención y a que se conocen las ventajas utilizar clasificadores ensamblados, en la tesis se proponen varios algoritmos a partir de los algoritmos Bagging y Boosting que incorporan en su funcionamiento técnicas de muestreo para tratar el problema de clases desbalanceadas en el escenario multinstancia. Las pruebas experimentales validadas por métodos estadísticos mostraron que las soluciones propuestas mejoran significativamente la calidad de la clasificación y son competitivas con otras soluciones existentes en la literatura.

Palabras claves: clasificación,multinstancia, ensamble, desbalanceadas, Bagging, Boosting, muestreo.

Abstract

Inside of machine learning, the multi-instance clasification has as his main objective to build, from a set of examples, a mathematical model that allows to classify objects that are described by multiples attributes vectors. The classification suffers some troubles that affects his performance, being the imbalanced class problem the main objective of this papper.

The imbalanced class problem occurs when exist a great difference in the class distributions and causes that the induced learning model from the set of examples do not present in a good way the concept that aims to learn and incur in many classification errors. Many ways to treat the imbalanced class problem has been found in the simple-instance context, being the introduction of sampling technics inside ensemble methods one of the most popular and effective.

The ensemble methods train multiple base classifiers and combine his predictions to classify an unknow instance. This algorithms generally present a better predictive capacity than individual classifiers, being the Bagging and Boosting algorithms two of most representative.

Because the imbalanced class problem inside the multi-instance classification is a young area that has received little attention and the advantages that presents the use of ensemble methods, the thesis propose a serie of algorithms from Bagging and Boosting that incorporates sampling methods in his body to treat the imbalanced class problem in the multi-instance context. The experimental tests validated by statistical methods shown that the solutions proposed improve significantly the quality of classification and they are competitive solutions with others of the state of art.

Keywords: multi-instance, classification, ensemble, imbalanced, Bagging, Boosting, sampling.

Índice general

Índice de figuras

Índice de cuadros

Introducción

Una de las principales tareas que busca resolver el aprendizaje automático es la clasificación. En este tipo de problemas se parte de un conjunto de objetos de entrenamiento, cada uno descrito por un vector de valores (cada valor describe un atributo) y una etiqueta de clase. Los objetos de entrenamiento se toman como ejemplo para inducir un modelo matemático capaz de predecir dichas etiquetas de clase de nuevos objetos que aparezcan sin clasificación. Los problemas de clasificación tradicional han sido enfocados principalmente a objetos que están descritos por un solo vector de atributos, identificándolos como tradicionales.

Dichos problemas son relativamente sencillos de modelar, pero pueden aparecer formas distintas de presentarse, surgiendo así la clasificación multinstancia que se diferencia a grandes rasgos en que los objetos son descritos por varios vectores, cada vector representa una instancia del objeto y cada objeto de clasificación se conoce comúnmente como bolsa, asociándose así, a cada bolsa, una etiqueta de clase.

A pesar de los avances obtenidos en el desarrollo y aplicación de métodos de clasificación multinstancia todavía es un área muy joven dentro del aprendizaje automático, y tiene aún muchos aspectos sin resolver, siendo uno de ellos el que se ocupa en la presente investigación: el problema de las clases desbalanceadas.

El problema de clases desbalanceadas surge en los problemas de clasificación tradicionales cuando la diferencia en el tamaño de las clases es muy acusado, provocando un deterioro importante de la calidad de la clasificación. La clasificación multinstancia también sufre el problema de las clases desbalanceadas.

Para dar solución al problema de clases desbalanceadas en el escenario simple-instancia se han seguido principalmente dos líneas de trabajo: introducir diferentes técnicas de muestreo para balancear las distribuciones de la clases y los algoritmos basados en costo. Las técnicas de muestreo principales con el submuestreo de la clase mayoritaria y el sobremuestreo de la clase minoritaria. Los algoritmos sensibles al costo se basan en una matriz de costos asociada a los errores de clasificación, tratando de minimizar el costo de la clasificación general.

Una de las formas de introducir las técnicas de muestreo para dar solución al problema de clases desbalanceadas es utilizando clasificadores ensamblados. Un clasificador ensamblado consiste a grandes rasgos en entrenar múltiples versiones de un clasificador base y combinar sus decisiones para resolver un problema.

Existen diversas formas de combinar los clasificadores base, siendo los algoritmos Bagging y Boosting dos de las más conocidos y populares debido a su eficacia. Bagging y Boosting han sido utilizados tanto en la clasificación simple-instancia como en la multinstancia.

El problema de clases desbalanceadas también está presente en el escenario multinstancia. Para resolverlo se han utilizado algoritmos basados en Boosting para introducir soluciones basadas en costo. Estas soluciones han mejorado los resultados de problemas de clasificación multinstancia en conjuntos desbalanceados, pero presentan la desventaja de que por lo general no se conoce a priori la matriz de costos para un conjunto de datos dado. Otra solución para la clasificación multinstancia se presenta en [44]. Allí se introduce el método de prepocesamiento de datos MISMOTE el cual realiza un sobremuestreo de la clase minoritaria mediante la introducción de ejemplos sintéticos. MISMOTE ha dado buenos resultados y puede ser utilizado con cualquier algoritmo de clasicación multinstancia tradicional.

Se conoce que los clasificadores ensamblados presentan frecuentemente mejor capacidad predictiva que los clasificadores individuales. Es posible aplicar MISMOTE a los datos multinstancia desbalanceados y luego entrenar un clasificador ensamblado, sin embargo hay estudios que muestran que aplicar el preprocesamiento en cada etapa de un ensamble es más efectivo que aplicar el preprocesamiento antes de entrenar el ensamble. Teniendo en cuanto lo planteado anteriormente surge la siguiente **pregunta de investigación:**

£Es posible desarrollar algoritmos basados en clasificadores ensamblados (bagging y boosting), que incorporen soluciones al problema de clases desbalanceadas basadas en muestreo, que mejoren significativamente la calidad de la clasificación multinstancia?

Este tipo de solución supondría una ventaja sobre los ensambles basados en costo, puesto que como se explicó anteriormente no sería necesario conocer a priori la matriz de costo para un determinado problema.

Para dar solución al problema científico se plantea como **objetivo general** de esta tesis:

Verificar que es posible desarrollar clasificadores ensamblados multinstancia que, integrando en su funcionamiento soluciones para el problema de clases desbalanceadas basadas en muestreo, mejoren significativamente la calidad de la clasificación multinstancia con clases desbalanceadas.

Dado que los principales tipos de ensambles son los bagging y los boosting, planteamos los siguientes **objetivos específicos**:

1. *Desarrollar clasificadores ensamblados basados en el algoritmo Bagging que integren en su funcionamiento técnicas de muestreo.*

2. *Desarrollar clasificadores ensamblados basados en el algoritmo Boosting que integren en su funcionamiento técnicas de muestreo.*

3. *Comparar los ensambles desarrollados entre sí y con otras soluciones al problema de desbalance de clases en la clasicación multinstancia para de-*

terminar las mejores soluciones y verificar la introducción de mejoras significativas en la calidad de la clasificación.

A partir de estos objetivos específicos nos planteamos responder a las siguientes preguntas:

Preguntas específicas:

£Cuál de los clasificadores ensamblados desarrollados basados en Bagging es el más eficaz?

£Cuál de los clasificadores ensamblados desarrollados basados en Boosting es el más eficaz?

£Cuál de los algoritmos más eficaces desarrollados, basados en Bagging y Boosting, presenta mejor desempeño?

£Son algunos de los algoritmos desarrollados mejores que las soluciones del estado del arte para el problema de clasificación multinstancia con clases desbalanceadas?

Estructura del documento:

La tesis está estructurada en tres capítulos. El primer capítulo se divide en dos secciones, en la primera se explica en qué consiste el aprendizaje automático así como una de sus principales tareas: la clasificación. Luego se realiza una formalización de la clasificación multinstancia, se muestra por qué es importante y se mencionan, junto con una breve descripción, algunos algoritmos utilizados para dar solución a problemas de clasificación multinstancia. En la segunda sección del capítulo 1 se describe el problema de clases desbalanceadas, las medidas de desempeño adecuadas a utilizar además de explicar una serie de soluciones a este problema en ambos escenarios, el simple-instancia y el multinstancia.

En el segundo capítulo se comienza explicando detalladamente el funcionamiento de los algoritmos usados como base, dando paso así a la descripción de los clasificadores ensamblados desarrollados por el autor.

En un tercer capítulo se describe el marco experimental, se hacen las comparaciones pertinentes entre los algoritmos desarrollados por el autor y los presentes en la bibliografía y se presentan los resultados obtenidos. Posteriormente el documento termina con las conclusiones y las recomendaciones del autor.

Capítulo 1

Fundamentación teórica

En este capítulo se aborda todo lo referente a la base teórica que fundamenta esta investigación. Se procede a realizar una revisión bibliográfica, primero, para presentar los fundamentos de la clasificación multinstancia. Luego, se formaliza la noción de clasificación multinstancia y se introduce la notación matemática que será usada en este informe. Por último, se procede a describir el problema de las clases desbalanceadas y cómo este afecta el desempeño de la clasificación, además de describir las soluciones propuestas hasta el momento para el problema de las clases desbalanceadas encontradas después de la revisión bibliográfica.

1.1. Contexto y formalización de la clasificación multinstancia

En esta sección se introduce el marco contextual de la investigación, haciendo un recorrido desde el aprendizaje automático, pasando por el aprendizaje multinstancia, hasta la clasificación multinstancia.

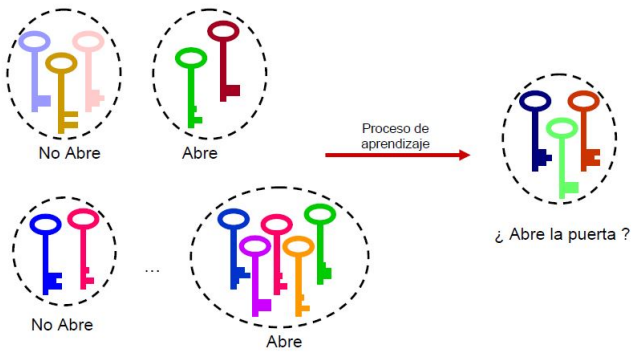


Figura 1.1.1: Ejemplo de clasificación multinstancia

La imagen anterior es un ejemplo de problema multinstancia. En dicho caso, se tienen distintos grupos de llaves en los cuales se encuentra o no una llave que abre determinada puerta y se desea saber si en la combinación de la derecha hay alguna llave que pueda abrir dicha puerta.

El objetivo de la clasificación multinstancia es construir un modelo a partir de las bolsas (conjuntos de llaves en el ejemplo) de entrenamiento que permita predecir las etiquetas de clase de nuevas bolsas. Esta es una de las tareas que ha recibido mayor atención en el aprendizaje multinstancia debido a su gran aplicabilidad en los más variados campos de la actividad humana.

1.1.1. Definición de aprendizaje automático

El tema de la investigación, la clasificación multinstancia, está contenido en el marco general del aprendizaje automático. El aprendizaje automático puede definirse de la siguiente manera: <<Un programa de computadora aprende de la experiencia E respecto a alguna clase de tarea T y una medida del desempeño D, si su desempeño en la tarea T, medido a través de D, aumenta con la experiencia E>>. Es necesario identificar estos tres elementos para plantear correctamente un problema de aprendizaje: el tipo de tarea a realizar por el aprendiz (T), la medida del desempeño que se debe mejorar (D) y la fuente de experiencia (E).

Un ejemplo del propio [34] ilustra esta idea: ((Un programa de computadora que aprenda a jugar damas puede mejorar su desempeño medido por su habilidad para ganar [D] en la clase de tarea que consiste en jugar el juego de damas [T], a través de la experiencia [E] obtenida por jugar contra el mismo.)) Los tres elementos mencionados determinan el proceso de aprendizaje en sí, y el hecho de que este proceso sea llevado a cabo por medio de un programa de computadora explica el empleo del adjetivo automático.

1.1.2. Tipos de aprendizaje automático

Existen varios tipos de aprendizaje automático, pero los más populares son el supervisado y el no supervisado. De los dos mencionados anteriormente, el más ampliamente utilizado y al mismo tiempo el más estudiado es el aprendizaje supervisado. En este tipo de aprendizaje, los algoritmos requieren el suministro de un conjunto de ejemplos, cada ejemplo descrito por un conjunto de atributos y una etiqueta asociada que corresponde a alguna propiedad importante o decisión relativa al ejemplo. La tarea del algoritmo de aprendizaje es construir un modelo que genere predicciones precisas de las etiquetas de futuros ejemplos.

Uno de los ejemplos que mejor puede ilustrar el concepto anterior es el reconocimiento facial, el cual está presente en muchas de las tecnologías que usamos en la actualidad y es formalizado de la siguiente forma: Dado un álbum digital de fotos de cientos de fotografías digitales, identificar aquellas fotos que incluyen a una persona dada. Un modelo de este proceso de decisión permitiría organizar las fotos por persona. Algunas cámaras y programas como iPhoto tienen esta capacidad.

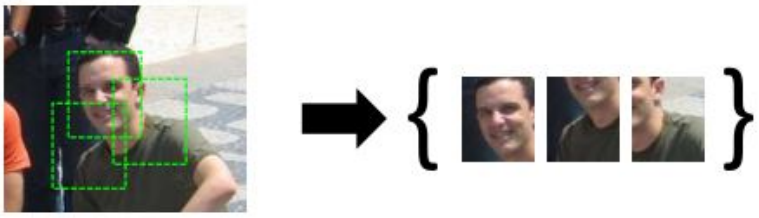


Figura 1.1.2: Reconocimiento facial como ejemplo de aprendizaje automático.

El aprendizaje no supervisado es un tipo de aprendizaje automático donde los ejemplos no presentan etiquetas de clase. Una forma de aprendizaje no supervisado, llamado agrupamiento (clustering en inglés) intenta dividir el conjunto de datos dado en grupos de instancias relacionadas. El grado de relación se mide típicamente utilizando una medida de proximidad, tal como la distancia Euclideana entre vectores de atributos.

El aprendizaje semisupervisado se aplica a conjuntos de datos donde algunas instancias están etiquetadas y otras no. Por tanto, puede considerarse que es un tipo intermedio entre el aprendizaje supervisado y el aprendizaje no supervisado. El etiquetado de los datos es con frecuencia un proceso manual y costoso, mientras que los datos sin etiquetar pueden ser adquiridos automáticamente y relativamente a bajo costo en algunos dominios de problema. Como en el caso supervisado, el objetivo es inferir una regla para predecir las etiquetas de instancias individuales. En algunos problemas se puede mejorar la exactitud de la clasificación/regresión tomando provecho de grandes fuentes de datos sin etiquetar. Los tipos de aprendizaje descritos anteriormente trabajan con datos organizados en forma de una tabla de plana (i.e. las instancias en las filas y los atributos en las columnas), pero existen muchos problemas de aprendizaje que tienen una estructura relacional, es decir, que están formados por varias tablas que se relacionan entre sí a través de atributos claves (similar a las bases de datos relacionales). Este tipo de problemas de aprendizaje se conoce como aprendizaje relacional, y se contrapone con los problemas que involucran una tabla plana, a los cuales se les llama problemas de aprendizaje atributo-valor. La programación lógica inductiva es un tipo de aprendizaje automático que, mediante lenguajes de programación como Prolog, se basa en la lógica de primer orden para representar y resolver problemas relacionales.

1.1.3. Formalización del aprendizaje supervisado estándar o simple-instancia

Para formalizar la notación que se va a utilizar en el presente trabajo, se va a utilizar la misma que [44]. Básicamente, una instancia es un par (x, y) donde $x =< x_1, x_2, , x_N >$ que pertenece a X es un vector de N atributos y y que pertenece a Y es la etiqueta de decisión de la instancia. El espacio N dimensional

X de donde x toma valores se conoce como espacio de instancias o espacio de atributos, y Y es el conjunto de etiquetas de decisión. La Tabla 1.1 muestra la forma en que se representa la información en el aprendizaje supervisado. Cada ejemplo x_i es descrito por un vector de atributos y una etiqueta de decisión y_i.

Ejemplos	Atributo 1	Atributo 2	...	Atributo N	Etiqueta
x_1	x_{11}	x_{12}	...	x_{1n}	y_1
x_2	x_{21}	x_{22}	...	x_{2n}	y_2
...	...	...	...	...	...
x_m	x_{m1}	x_{m2}	...	x_{mn}	y_m

Cuadro 1.1: Representación de la información en el aprendizaje automático estándar

El objetivo del aprendizaje supervisado es hallar una función $f(x) = y$, o sea, encontrar una función que logre encontrar el valor de la columna etiqueta de nuevas instancias de las que se desconozca dicho valor. Todo este proceso a partir de un conjunto de instancias de entrenamiento D. Cuando el valor de la clase es de tipo nominal, este proceso se conoce como clasificación y cuando es de tipo numérico se conoce como regresión. Dado un conjunto de ejemplos de entrenamiento D a partir de los cuales aprender, un algoritmo de aprendizaje supervisado devuelve un modelo de los datos $h(x)$, el cual se pretende que sea la mejor aproximación a $f(x)$. Dicho modelo se conoce como hipótesis o descripción del concepto y es el utilizado en la etapa de prueba.

1.1.4. Aprendizaje multinstancia

Como se planteó en la sección anterior, en el aprendizaje supervisado cada instancia es un par (x, y) donde x es un vector de valores y y es la etiqueta asociada a cada vector. Aquí reside la principal diferencia entre el aprendizaje supervisado simple-instancia y el multinstancia, debido a que en este último cada instancia u objeto está descrito ya no solo por un vector de valores, sino por varios vectores, convirtiéndose así cada instancia en una bolsa (bag) y asociándosele a esta el valor de una etiqueta de clase. Es válido aclarar que el valor de dicha etiqueta está asociada a cada bolsa, no a las instancias que la componen. El término bolsa evoca la imagen de un saco que contiene las instancias sin orden ni concierto, pudiendo estar incluso repetida la misma instancia dentro de la bolsa.

A partir de un conjunto de bolsas de entrenamiento, el objetivo del aprendizaje multinstancia es el mismo que el del aprendizaje supervisado estándar: hallar una función $f(x) = y$, o sea, encontrar una función que logre encontrar el valor de la etiqueta de clase de nuevas bolsas de las que se desconozca dicho valor.

El aprendizaje multinstancia fue definido por Dietterich en [8]. Allí encontramos el primer ejemplo: la predicción de actividad de fármacos. En este caso, el objeto de entrada es una molécula y el resultado es la medida en que se adhiere

a una molécula mucho más grande. Una buena molécula del fármaco se adherirá con fuerza al lugar deseado, mientras que una mala no se unirá en una buena medida. Este es un caso de aprendizaje multinstancia debido a que la conformación de una misma molécula puede estar dada de varias formas, debido a la rotación de sus enlaces como se puede observar en la figura 1.3.

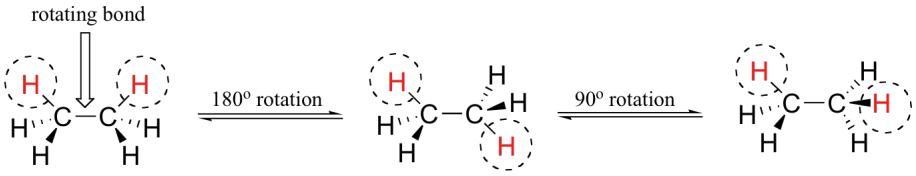
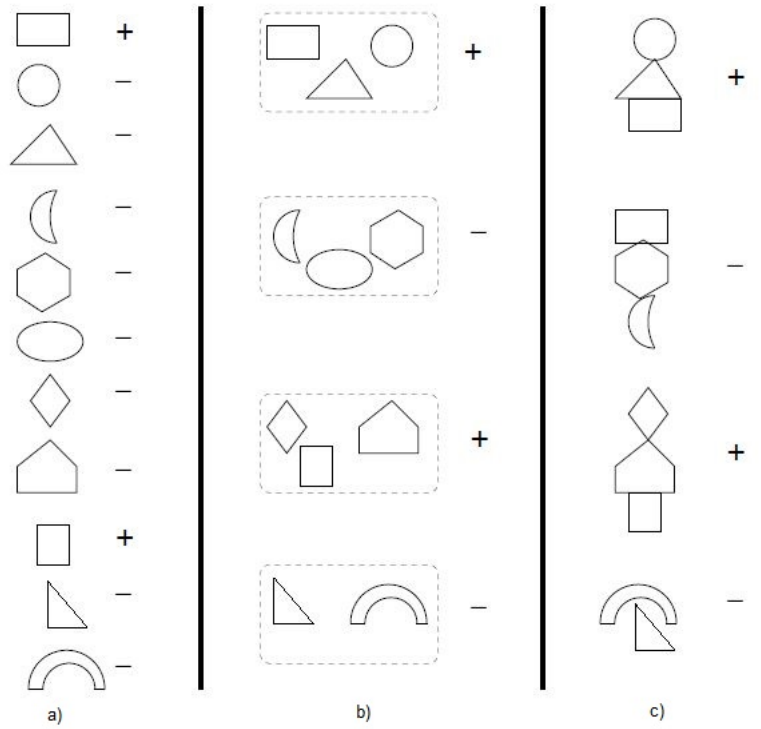


Figura 1.1.3: Ejemplo de las moléculas en el aprendizaje multinstancia

Aplicar el aprendizaje automático en la predicción de la actividad de fármacos podría ser un gran beneficio para dicho proceso. Si los químicos pudieran obtener un modelo tri-dimensional de los requerimientos de la actividad de las drogas, serían capaces de designar mejores.

Este es un caso de aprendizaje multinstancia donde cada molécula emula una bolsa y las diferentes conformaciones de la misma serían las instancias pertenecientes a dicha bolsa.

Desde el punto de vista teórico el aprendizaje multinstancia ocupa un lugar intermedio entre el aprendizaje supervisado estándar y el aprendizaje relacional de primer orden. El aprendizaje supervisado estándar es un caso especial del aprendizaje multinstancia y a la vez este último es un caso especial del aprendizaje relacional de primero orden. Según [40]el aprendizaje multinstancia es un paso clave en la transición entre el aprendizaje supervisado estándar y el relacional de primer orden. En la figura 1.4 se puede observar lo explicado anteriormente.



a) Aprendizaje supervisado simple-instancia b)Aprendizaje supervisado multinstancia c) Aprendizaje relacional de primer orden.

Figura 1.1.4: Relación entre aprendizaje simple-instancia, multinstancia y relacional de primer orden.

1.1.5. Formalización de la clasificación multinstancia

Como se hizo anteriormente, para formalizar en este caso la clasificación multinstancia se va a utilizar la misma forma que en [44]. Un ejemplo en la clasificación multinstancia es un par (X, y) donde X= $\{x_1, x_2, ..., x_T\}$ que pertenece N^x es un multiconjunto de T instancias y y es la etiqueta de clase asociada a cada bolsa X. Multiconjunto es la palabra que en jerga matemática equivale a bolsa. Se dice que es un multiconjunto y no simplemente un conjunto porque el multiconjunto, a diferencia de este, permite tener instancias repetidas.

A diferencia del aprendizaje supervisado estándar las instancias x_i no presentan etiquetas de clase, presentándolas solo las bolsas X_j que son conformadas por dichas instancias. El conjunto Y es el formado por las etiquetas de clase asociadas a cada bolsa. En un ambiente estándar[1] se asume que una bolsa es etiquetada como positiva si y solo si tiene al menos una instancia positiva (+),

[1]Existen varias hipótesis para clasificar una bolsa. Para más información remitirse a [44].

en otro caso, la bolsa es clasificada de forma negativa (-), aunque de forma general, Y puede estar compuesto por más de dos elementos. En este caso se dice que el problema es de clasificación multiclase. En la tabla 1.2 se puede observar lo descrito anteriormente.

Ejemplo	Instancias	Atributo 1	Atributo 2	...	Atributo N	Etiqueta
X_1	$x_{1,1}$	$x_{1,1,1}$	$x_{1,1,2}$	...	$x_{1,1,N}$	y_1
	...	...	...	...	...	
	x_{1,T_1}	$x_{1,T_1,1}$	$x_{1,T_1,2}$	...	$x_{1,T_1,N}$	
...	...	...	...	...	...	...
X_M	$x_{M,1}$	$x_{M,1,1}$	$x_{M,1,2}$	...	$x_{M,1,N}$	y_M
	...	...	...	...	...	
	x_{M,T_M}	$x_{M,T_M,1}$	$x_{M,T_M,2}$	...	$x_{M,T_M,N}$	

Cuadro 1.2: Representación de la información en el aprendizaje automático

La clasificación multinstancia consta de dos etapas: la etapa de entrenamiento y la etapa de prueba. En la etapa de entrenamiento se pretende encontrar $f(X)=y$, basado, al igual que la clasificación simple-instancia, en un conjunto de instancias de entrenamiento D, pero en este caso conformado por bolsas en vez de instancias simples. Un algoritmo de aprendizaje multinstancia construye a partir de D un clasificador o modelo de datos $h(x)$ el cual es una hipótesis de $f(x)$ y es el utilizado en la etapa de prueba para clasificar ejemplos sin etiqueta de clase.

1.1.6. Importancia y aplicaciones de la clasificación multinstancia

En esta sección se tratarán una serie de aplicaciones de la clasificación mutinstancia muy interesantes y mejor, vigentes en la actualidad, quedando así demostrada su importancia.

El aprendizaje multinstancia es una rama muy joven del aprendizaje automático, sin embargo ha despertado un gran interés entre los investigadores y profesionales debido a su capacidad para modelar problemas de aprendizaje caracterizados por la presencia de ambigüedad en sus datos, y también porque ha venido a ser un eslabón que permite enlazar el aprendizaje proposicional con el relacional. El debut del aprendizaje multinstancia fue con los algoritmos de rectángulos paralelos a los ejes para la predicción de la actividad farmacológica. Hoy en día existen decenas de métodos de solución y siguen apareciendo nuevas propuestas que mejoran aspectos específicos del problema. Igualmente, ha crecido significativamente el número y la variedad de los campos de aplicación que se han beneficiado con el aprendizaje multinstancia. A continuación se mencionarán una serie de aplicaciones de la clasificación multinstancia.

Recuperación de imágenes basadas en su contenido es uno de los campos donde más se ha utilizado la clasificación multinstancia según [1, 39]. En este caso una bolsa es emulada por una imagen, mientras que las instancias son sus

segmentos obtenidos mediante alguna técnica de segmentación. Por ejemplo, en el reconocimiento facial, es necesario que el algoritmo de clasificación aprenda de los rasgos distintivos de una serie de imágenes faciales segmentadas de entrenamiento.

Otro de los dominios de aplicación donde se ha utilizado con muy buenos resultados es la *categorización de textos*. Similar al ejemplo anterior, un documento puede constar de múltiples secciones de diferentes tópicos. Ray y Craven [39] aplicaron un método basado en SVM MIL a este problema, dónde cada documento representa una bolsa y pequeños pasajes de los documentos de 50 palabras de longitud representan las instancias asociadas.

Además se ha introducido la clasificación multinstancia en el trabajo con *series de tiempo* [35], *identificar familias de proteínas* entre otros muchos ejemplos.

1.1.7. Algoritmos de clasificación multinstancia

En esta sección se expondrán algunos de los algoritmos más importantes de clasificación multinstancia, mencionándolos por cada grupo descrito por [44] y describiendo uno por grupo, debido a que existen muchos algoritmos con distintos enfoques y centrarse en todos sería prácticamente imposible y no es un objetivo de este trabajo.

1.1.7.1. Algoritmos ad-hoc

En el primer grupo, los *Algoritmos ad-hoc* están los algoritmos de rectángulos paralelos a los ejes (APR) [8], los algoritmos basados en el enfoque de densidad diversa [30, 31] y el algoritmo GMIL [43] basado en resultados teóricos del reconocimiento de patrones geométricos.

En este acápite se va a describir los algoritmos de Densidad Diversa debido a su gran dominio de aplicación y además debido a que fue el primer modelo probabilístico desarrollado para el aprendizaje multinstancia. Fue propuesto originalmente por [32], los cuales definieron la densidad diversa como un punto que es Íuna medida de cuantas bolsas positivas tienen instancias cercanas a ese punto y cuan lejos las instancias de las bolsas negativas estánÍ . La idea básica es hallar un punto en el espacio de búsqueda con la mayor Densidad Diversa. En otras palabras, un punto candidato es uno que este cerca de la mayor cantidad de bolsas positivas posibles y a la vez lejos de tantas bolsas negativas como sea posible.

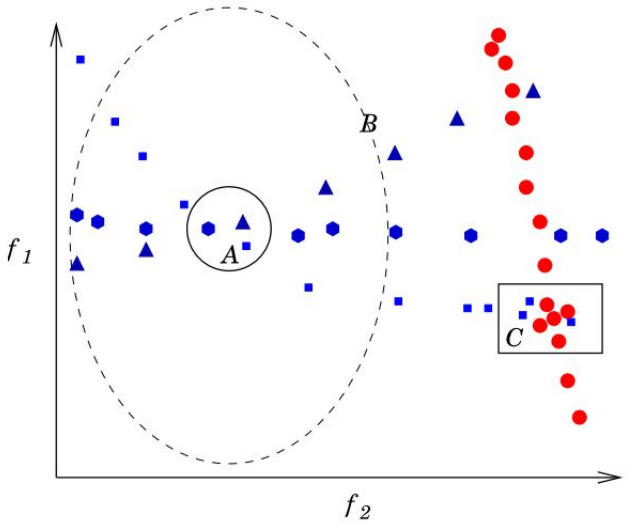


Figura 1.1.5: En esta imagen se pueden observar 3 bolsas positivas y una negativa, siendo sus instancias representadas por las figuras geométricas y descritas por dos atributos: f1 y f2. La bolsa que contiene los círculos rojos es la negativa. La región C es una región de densidad alta porque varias instancias pertenecen a esa región. La región A es de alta ÍDensidad DiversaÍ porque varias instancias de diferentes bolsas positivas pertenecen a esa región y no hay instancias de bolsas negativas cerca. La región B muestra un concepto que puede ser aprendido si el algoritmo de aprendizaje asume que todas las instancias en las bolsas positivas son positivas.

1.1.7.2. Algoritmos tradicionales adaptados al aprendizaje multinstancia

Es lógico pensar en adaptar un algoritmos existente de clasificación simple-instancia para usarlo en la clasificación mutinstancia, siendo este un enfoque muy utilizado en la actualidad y es el usado por los autores de la presente investigación como se verá mas adelante; debido a esto haremos énfasis en esta técnica.

El adaptar los algoritmos tradicionales a la clasificación multinstancia trae muchas ventajas debido a que existen algoritmos muy ÍfuertesÍ en el escenario monoistancia que serían fáciles de adaptar con solo cambios menores en el algoritmo, ahorrando tiempo de diseño y de desarrollo. Los algoritmos de aprendizaje supervisado que han sido adaptados al escenario multinstancia incluyen el k-vecinos más cercano, los arboles de decisión, las maquinas de soporte vectorial, la regresión logística y la ampliación (boosting), entre otros.

En este caso el algoritmo que se seleccionó para abundar en la forma en que se ideó fue el de *ampliación* (de ahora en adelante *boosting*). Se seleccionó este algoritmo debido a que como se pudo observar en la introducción es uno de los

objetivos de este proyecto el desarrollar clasificadores a partir de los algoritmos *bagging* y *boosting*.

Boosting es un un algoritmo muy popular y poderoso en el aprendizaje automático estadístico. Una gran cantidad de variantes de *boosting* están presentes en la literatura probando ser efectivos en muchas aplicaciones, [52, 13]. La idea detrás de este algoritmo es tomar un algoritmo de aprendizaje simple, entrenar varias veces el clasificador y luego combinarlos. Mayormente la combinación es hecha mediante una suma «pesada», siendo definido el clasificador de la siguiente manera: $h(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$ donde h_t es el clasificador débil o base y α_t son los pesos positivos. En el capítulo 2 se abunda en este algoritmo.

[49]propuso una variante del boosting para el escenario multinstancia. Este es similar a los modelos de estimación probabilística AdaBoost y LogitBoost [14].

1.1.7.3. Algoritmos envoltorios

Otro método utilizado en la clasificación multinstancia es el de construir algoritmos generales que apliquen un clasificador simple-instancia arbitrario a los datos multinstancia. En este caso, el algoritmo de clasificación simple-instancia no cambia en absoluto, en lugar de ello se aplica algún tipo de filtro para que los datos de forma multinstancia puedan ser utilizados por el algoritmo simple-instancia. La salida de este proceso es usada para las predicciones.

Los algoritmos envoltorios multinstancia pueden aplicarse tanto a conceptos multinstancia basados en instancia como en los basados en metadatos. Para los conceptos multinstancia basados en instancias, la hipótesis es que las instancias están etiquetadas a través de algún proceso, y estas etiquetas determinan las etiquetas de clase a nivel de bolsa. Los algoritmos envoltorios que aprenden conceptos basados en instancias usualmente aplican el aprendizaje simple-instancia directamente a las instancias dentro de las bolsas de entrenamiento. En el momento de la predicción las etiquetas a nivel de bolsa son asignadas teniendo en cuenta la hipótesis multinstancia que haya asumido, por ejemplo la hipótesis multinstancia estándar.

En contraste, se asume que las etiquetas de clase a nivel de bolsa para los conceptos multinstancia basados en metadatos son determinadas por un proceso en algún espacio de atributos simple-instancia que consiste en metadatos extraídos de las bolsas. Los algoritmos envoltorios que aprenden este tipo de conceptos usualmente mapean cada bolsa en el espacio de atributos basado en metadatos a través de alguna transformación, y entonces construyen un modelo simple-instancia en el espacio de atributos transformado. Cada instancia en el espacio de atributos basado en los metadatos corresponde a una bolsa. Entonces, el clasificador base simple-instancia puede proporcionar directamente la etiquetas de clase a nivel de bolsa.

De este grupo de algoritmos se seleccionó para su breve descripción el algoritmo MILES (*Multiple-Instance Learning via Embedded Instance Selection*) [7] debido a que se basa en el marco de densidad diversa analizado anteriormente.

MILES incrusta bolsas en un espacio de atributo simple-instancia, y aplica un algoritmo de soporte vectorial con norma 1 al conjunto de datos transformado. Sin embargo, en lugar del Support Vector Machine norma 1, podrían usarse otros algoritmos de aprendizaje simple-instancia, por tanto clasicamos este algoritmo como un enfoque envoltorio.

MILES usa una hipótesis simétrica, donde se permiten múltiples puntos objetivos que pueden estar relacionados lo mismo a bolsas positivas como a bolsas negativas. Bajo esta hipótesis, y usando el estimador causa-más-probable del marco densidad diversa, Chen et al. denen una medida que estima la probabilidad de que un punto x sea un punto objetivo dada una bolsa $X_i$sin importar la etiqueta de clase de la bolsa:

$$Pr\left(x|X_i\right) \propto s\left(x|X_i\right) = \max_j \exp\left(-\frac{[\![x_{ij} - x]\!]^2}{\sigma^2}\right) \tag{1.1.1}$$

donde x_{ij} son las instancias dentro de la bolsa X_i, y σ es un factor escala predenido. Observe que $s(x \mid X_i)$ puede considerarse como una medida de similitud entre una bolsa y una instancia, la cual es determinada por la instancia x y la instancia más cercana dentro de la bolsa X_i.

Para encontrar los puntos objetivos MILES usa la hipótesis de que los puntos objetivos pueden aproximarse por las instancias dentro de las bolsas. En otras palabras, cada instancia es candidata para un punto objetivo. Los candidatos son representados como atributos en un espacio de atributos basado en instancia F_x. Cada bolsa en el conjunto de entrenamiento se mapea en F_x por medio de la transformación

$$m\left(X_i\right) = \left[s\left(x^1 \mid X_i\right), s\left(x^2 \mid X_i\right), ..., s\left(x^n \mid X_i\right)\right]^T$$

donde $x_i \;\; X$ es una instancia del conjunto X de todas las instancias en todas las bolsas de entrenamiento. Si se añaden las etiquetas de clase $y \;\; Y$, el espacio ($F_x \mid Y$) es un espacio de atributos simple-instancia al cual pueden aplicarse los algoritmos del aprendizaje supervisado tradicional. La salida de este clasicador puede usarse para asignar etiquetas de clase a nivel de bolsa para datos futuros.

1.2. Problema de clases desbalanceadas y sus soluciones en el escenario multinstancia

El problema de clases desbalanceadas también afecta la clasificación multinstancia, sin embargo este ha recibido poca atención hasta el momento. Después de una revisión bibliográfica intensa se han encontrado solo dos soluciones para este problema en la clasificación multinstancia: los algoritmos amplificadores sensibles al costo [50] y el MISMOTE [44]. Dichos métodos se van a evaluar con mayor énfasis más adelante. A continuación se explica el problema de las clases desbalanceadas y se exponen las soluciones para este problema encontradas en la bibliografía.

1.2.1. Formalización del problema de clases desbalanceadas

Muchos de los algoritmos de aprendizaje automático asumen que el conjunto de datos está balanceado. Sin embargo este no es siempre el caso en problemas del mundo real donde una clase puede estar representada por un gran número de ejemplos, mientras la otra está representada por solo unos pocos. Este es conocido como el problema de las clases desbalanceadas, el cual se presenta como un obstáculo frecuente en la clasificación, particularmente, altos niveles de error en la clasificación de los ejemplos de la clase minoritaria. Este problema es significativo porque generalmente la clase minoritaria es la más importante.

En años recientes, el problema de clases desbalanceadas ha emergido como uno de los retos en la comunidad de minería de datos [53]. Ha cobrado tal importancia debido a que está presente en muchos problemas de clasificación del mundo real. Por ejemplo, alguna de las aplicaciones conocidas que sufren de este problema son, la detección de fallos [54],[57], la detección de anomalías [24],[47], diagnósticos médicos [33], almacenamiento de e-mails [3], reconocimiento facial [29], o la detección de filtración de petróleo [26], entre muchas otras.

Para ilustrar este problema se presenta un problema de clasificación biomédica [19]. Considérese el conjunto «Mammography Data Set», una colección de imágenes adquirida de una serie de exámenes de mamas realizado sobre diferentes pacientes, que ha sido a su vez utilizado en el análisis de algoritmos dirigidos al problema en cuestión. Analizando las imágenes en sentido binario (no necesariamente tienen que ser dos clases, existe también el caso multiclase [46]), correspondiendo las etiquetas Positiva a una persona con cáncer, mientras que la etiqueta Negativa corresponde a una persona saludable. Se conoce que las personas con cáncer son mucho menores en cantidad a las personas saludables, de hecho, en este conjunto de datos se observan 10923 casos Negativos (mayoría) y 260 casos Positivos (minoría). En este caso al aplicar una serie de algoritmos para la clasificación se observa que la precisión cuando se trata de la clase mayoritaria es cercana al 100 por ciento, mientras que la de la clase minoritaria es oscila entre 0 y 10 por ciento.

Aunque comúnmente se conoce este problema como el de clases desbalanceadas, en realidad la diferencia entre el tamaño de las clases no es la esencia del problema, sino una condición que favorece la manifestación de otros problemas que subyacen en segundo plano. Entre las causas primarias que dan lugar a los efectos observados tras el problema de las clases desbalanceadas, algunos de los más conocidos son los siguientes:

- *Uso de medidas del desempeño inadecuadas para guiar el proceso de aprendizaje:* Los algoritmos de aprendizaje que usen de forma explícita o implícita heurísticas basadas en maximizar medidas del desempeño tales como la exactitud de la clasificación (accuracy) tenderán a ignorar los ejemplos de la clase minoritaria. Como estas medidas tratan los aciertos de ejemplos positivos de igual manera que los aciertos de ejemplos negativos, al ser estos últimos mucho más abundantes, los aciertos de ejemplos positivos

pasan a ser irrelevantes. Consideremos un conjunto de datos desbalanceado, que tenga, digamos, un ejemplo positivo por cada 100 negativos. Un clasificador que trate de maximizar el accuracy de su regla de clasificación podrá obtener una exactitud del 99 % con solo ignorar los ejemplos de la clase positiva, clasificando todos los ejemplos como negativos.

- *Pequeños tamaños de la muestra:* Para que el aprendizaje sea efectivo se necesitan conjuntos de ejemplos representativos de cada clase. Si el número de ejemplos de la clase minoritaria es muy bajo el desempeño de la clasificación puede afectarse por falta de representatividad [23].
- *Solapamiento entre clases:* Las clases pueden estar (en alguna medida) solapadas por una pobre definición intrínseca de los conceptos que ellas representan, o porque los atributos seleccionados no son suficientes para diferenciarlas adecuadamente [2]. Naturalmente, mientras mayor es el solapamiento entre las clases mas difícil se hace su separación. Sin embargo, las clases desbalanceadas amplifican el efecto: menores niveles de solapamiento son suficientes para observar un marcado deterioro del desempeño de la clasificación [37, 17]
- *Presencia de subconceptos en la clase minoritaria:* La presencia de subconceptos es otro factor que aumenta la complejidad de un problema de clasificación debido a que implica una descripción más larga de la frontera de decisión. Las fronteras de decisión más largas requieren más ejemplos para su especificación. Por eso, aunque los subconceptos pueden aparecer en cualquier clase, cuando están en la clase minoritaria, puede ser particularmente difícil su identificación por estar más esparcida la muestra [19]

Usualmente el desbalance entre el tamaño de dos clases es caracterizado mediante la tasa de desbalance (IR por su acrónimo en inglés *Imbalanced Ratio*), que se define como el cociente del número de ejemplos en la clase mayoritaria entre el número de ejemplos en la clase minoritaria [18, 36]. Mientras mayor es el IR, mayor es el desbalance entre las clases. Un elevado IR esta asociado con una mayor dificultad para su clasificación. Sin embargo, no puede decirse explícitamente a partir de que valor de IR es apreciable el deterioro de la clasificación, debido al carácter heterogéneo de la complejidad de los conjuntos de datos. La mayoría de los algoritmos de aprendizaje tradicionales no son apropiados para datos con clases desbalanceadas, generando modelos de clasificación subóptimos en estos casos, i.e., una buena cobertura de los ejemplos de la clase mayoritaria, pero frecuentes errores en la clasificación de los ejemplos de la clase minoritaria. Actualmente es ampliamente reconocida la necesidad de emplear medidas robustas para evaluar la calidad de la clasificación cuando las clases están desbalanceadas, i.e., medidas que no sean falseadas por la prevalencia de una clase, y cuantifiquen fielmente cuán bueno es un clasificador, ya sea identificando ejemplos de la clase minoritaria, o identificando ejemplos de ambas clases, como mejor convenga a la aplicación. En la sección siguiente se presentan algunas de estas medidas, las cuales son usadas en este estudio.

	Clasificado como positivo	**Clasificado como Negativos**
Realmente Positivo (AP)	Verdaderos Positivos (TP)	Falsos Negativos (FN)
Realmente Negativo (AN)	Falsos Positivos (FP)	Verdaderos Negativos (TN)

Cuadro 1.3: Matriz de confusión para un problema binario

1.2.2. Cómo medir el desempeño de la clasificación cuando las clases están desbalanceadas

Para medir el desempeño de la clasificación cuando se presenta el problema de clases desbalanceadas se va a utilizar las medidas empleadas por [44]. A continuación se mencionan y se explican de forma sintetizada, siendo cada una útil en determinado campo de aplicación.

- *Precision y recall:* provienen del campo de recuperación de información. *Precision* es una medida de exactitud ya que da la fracción de ejemplos clasificados como positivos que son realmente positivos; mientras que el *recall* o exactitud de la clase positiva es una medida de completitud puesto que da la fracción de ejemplos de la clase positiva clasificados correctamente. Estas medidas se centran en la clase Positiva o minoritaria.

- F_1: proviene también del campo de recuperación de la información. Se dene como la media armónica de *precision* y *recall*. Por lo general, el objetivo principal de un algoritmo de aprendizaje es maximizar el *recall*, sin sacricar la *precision*. Sin embargo, ambos objetivos entran en conicto frecuentemente, y es difícil decidir el mejor entre dos algoritmos si uno aventaja al otro en *precision* y el otro aventaja al primero en *recall*. F1 incorpora el equilibrio costo-benecio de *recall* y *precision*, dando igual peso a ambas medidas, para evaluar el desempeño de un clasicador a través de un solo número. Al igual que *precision* y *recall*, F1 se enfoca solo en la clase positiva.

- *AUC:* es el área bajo una curva ROC[2]. La curva ROC describe equilibrios entre beneficios (TP) y costos (FP) a lo largo de un rango de umbrales de un modelo de clasicación. Los grácos ROC son consistentes para un problema dado aún si la distribución de instancias positivas y negativas está altamente desbalanceada [38, 11]. El AUC permite medir el desempeño de un clasicador para evaluar cuál modelo es en promedio mejor usando un único valor numérico obtenido a partir de la curva ROC. Su valor representa la probabilidad de que un ejemplo positivo seleccionado aleatoriamente sea evaluado (correctamente) con más seguridad que un ejemplo negativo seleccionado aleatoriamente. AUC tiene un sólido signicado estadístico ya que es equivalente a la prueba de rangos de Wilcoxon. A diferencia de

[2]siglas del término inglés Receiver Operating Characteristic, ya que originalmente se utilizó para caracterizar el funcionamiento de los radares. Mientras que AUC son las siglas de Area Under the (ROC) Curve.

las medidas anteriores, AUC toma en cuenta la identicación correcta de ambas clases.

- GMean: Fue sugerida por Kubat en [26] como las medias geométricas del recall de todas las clases planteada por la fórmula 1.2.1. Como todo valor del recall que representa el desempeño de la clasificación de una clase especifica es igualmente contado, el GMean es capaz de medir el rendimiento balanceado entre las clases de la salida del proceso de clasificación.

$$G - Mean = \left(\prod_{i=1}^{k} R_i \right)^{1/k} \tag{1.2.1}$$

1.2.3. Soluciones existentes al problema de las clases desbalanceadas en la clasificación simple-instancia

Según la literatura especializada [45, 28, 16, 19], el problema de las clases desbalanceadas en la clasificación simple-instancia se puede enfrentar de cuatro formas generales:

1. Soluciones a nivel de algoritmo: Los algoritmos tradicionales de aprendizaje de clasicadores son adaptados para favorecer el aprendizaje de la clase pequeña. Se ha probado que estas soluciones a nivel de algoritmo son efectivas para ciertos algoritmos de aprendizaje de clasicadores y en ciertos dominios de aplicaciones, sin embargo son difciles de extender a otros contextos porque requieren un profundo conocimiento del algoritmo de aprendizaje y del por qué fallan cuando la distribución de clases de los datos disponibles está desbalanceada. Por este motivo, son preferibles las soluciones en las que la estrategia de aprendizaje subyacente permanece sin cambios.

2. Soluciones sensibles al costo: Se basan en los diferentes costos asociados a los errores de clasicacion en las clases o incluso en ejemplos especícos, ya que el costo del error en la clasicación de un ejemplo de la clase más pequeña es, por lo general, más elevado. La estrategia en este tipo de solución es minimizar el costo de la clasicación general. La principal desventaja del aprendizaje sensible al costo es que esta basado en una matriz de costo para diferentes tipos de errores o ejemplos. Sin embargo, muchas veces no se dispone de una matriz de costo para un conjunto de datos dado.

3. Soluciones de remuestreo: Alteran la distribución de clases mediante un remuestreo del espacio de datos buscando el rebalance de las clases. Las formas más sencillas de remuestreo son el submuestreo aleatorio de la clase mayoritaria y el sobremuestreo aleatorio de la clase minoritaria. Estas estrategias pueden conducir a la pérdida de información y el sobreajuste, respectivamente. Para tratar de evitar estos inconvenientes se han usado algunas heurísticas para seleccionar deliberadamente los ejemplos que son eliminados, en el llamado submuestreo informado de la clase mayoritaria,

y aquellos que son remuestreados en el llamado sobremuestreo informado de la clase minoritaria.

4. Soluciones de clasificadores ensamblados o ensambles: Los clasificadores ensamblados construyen múltiples clasicadores a partir de los datos originales y luego agregan sus predicciones cuando clasican ejemplos desconocidos. Los ensambles pueden ser de tipo remuestreo o sensibles al costo. Los ensambles de remuestreo se basan en la aplicación de una técnica de remuestreo a una parte de los datos antes del entrenamiento de cada uno de los clasicadores que forman parte del ensamble. En los ensambles sensibles al costo el peso de los ejemplos se calcula tomando en cuenta, no solo la frecuencia de las clasicaciones erradas, sino también los costos de los errores.

1.2.3.1. Variantes de las soluciones de clasificadores ensamblados basadas en muestreo

Existen disímiles estrategias basadas en muestreo para tratar el problema de las clases desbalanceadas mediante clasificadores ensamblados, varias de las cuales son usadas en el presente trabajo investigativo:

1. Submuestrear aleatoreamente la clase mayoritaria (negativa): es una técnica no heurística que pretende balancear los tamaños de las clases mediante la eliminación al azar de ejemplos de la clase negativa. Su mayor inconveniente es que puede descartar datos muy útiles, que pueden ser importantes en el proceso de inducción.

2. Sobremuestrear aleatoriamente la clase minoritaria (positiva): este método trata de balancear las distribuciones de las clases, duplicando aleatoriamente instancias de la clase positiva. Como es lógico tiende a introducir redundancia en los datos pues hace copias exactas de las instancias existentes.

3. Sobremuestreo sintético de la clase minoritaria (positiva): la idea principal de esta estrategia es crear nuevas instancias mediante la interpolación de instancias semejantes existentes. Este proceso se lleva a cabo mediante algoritmos como SMOTE [6], MSMOTE [21], entre otros.

4. Combinación de sobremuestreo y submuestro.

Las técnicas descritas con anterioridad son también aplicables al escenario multinstancia. En la presente investigación se utilizó la estrategia de submuestreo aleatorio de la clase mayoritaria, tal y como se explicó anteriormente, el sobremuestreo con ejemplos sintéticos de la clase minoritaria, utilizando en este caso el algoritmo MISMOTE [44] y la combinación del submuestreo aleatorio de la clase mayoritaria y el sobremuestreo con ejemplos sintéticos de la clase minoritaria. Estas técnicas son descritas a lo largo del capítulo 2.

Como aclaración es necesario decir que cuando se utiliza la palabra «balancear» no se habla de una equivalencia exacta entre el tamaño de las clases, de

hecho, en [10, 9] se muestra que el tamaño final del remuestreo no resulta necesariamente en la misma cantidad de instancias en cada clase. La distribución óptima depende del dominio de entrada entre otros factores y no es fácil de estimar a priori.

1.2.4. Soluciones al problema de clases desbalanceadas en la clasificación multinstancia

En esta sección se van a describir dos soluciones para tratar el problema de clases desbalanceadas. La primera es muy popular en este campo y es un antecedente directo de los desarrollados en este trabajo mientras que el segundo es utilizado dentro de los algoritmos planteados para generar ejemplos sintéticos a partir de la clase minoritaria y así balancear las cantidades de cada clase.

1.2.4.1. Algoritmos AdaBoost sensibles al costo

Wang et al. [50] estudiaron el uso de algoritmos amplicadores sensibles al costo en conjuntos de datos multinstancia desbalanceados. Los métodos amplicadores son algoritmos ensambles secuenciales que recalculan el peso de los datos después de cada iteración, de forma que el próximo clasicador débil en ser entrenado se centre más en los ejemplos que fueron mal clasicados la vez anterior. A describe brevemente el amplicador AdaBoost[41] debido a que será tratado más adelanto con mayor énfasis.

El amplicador AdaBoost usa un algoritmo de aprendizaje base (también llamado débil) que acepta como entrada un conjunto de ejemplos de entrenamiento T junto con una distribución de pesos D. Dada esa entrada, el algoritmo de aprendizaje base calcula una hipótesis base (o débil) h. El signo de $h(x)$ es interpretado como la etiqueta que debe ser asignada a x. La idea de la amplicación es usar el algoritmo de aprendizaje base para formar una regla de predicción de gran exactitud al invocar el algoritmo de aprendizaje base repetidamente en diferentes distribuciones sobre los ejemplos de entrenamiento. En cada iteración el peso de los ejemplos correcta (o incorrectamente) clasicados por h es disminuido (o aumentado).

Especícamente, el peso $D^{t+1}(i)$ del ejemplo $x_i$en la iteración $t+1$ se calcula mediante las fórmulas (1,2,2) y (1,2,3)

$$D^{t+1}(i) = \frac{D^t(i) K_t(x_i, y_i)}{Z_t} \tag{1.2.2}$$

$$K_t(x_i, y_i) = \exp(-a_t y_i h_t(x_i)) \tag{1.2.3}$$

$Z_t$es un factor de normalización escogido de forma tal que D^{t+1}sea una distribución . La hipótesis nal del amplicador, dada por la Fórmula (1,2,4), es el signo de la suma ponderada de las predicciones de los T clasicadores base.

$$H(x) = sign\left(\sum_{t=1}^{T} a_t h_t(x)\right) \tag{1.2.4}$$

Los amplicadores pueden usarse con cualquier algoritmo de aprendizaje base siempre que este pueda manejar pesos en los ejemplos. Si se usa un algoritmo de aprendizaje multinstancia como algoritmo de aprendizaje base, entonces el amplicador será capaz de aprender a partir de conjuntos de datos multinstancia (en lugar de conjuntos de datos simpleinstancia). La estrategia propuesta en [50] es usar un clasicador multinstancia como algoritmo de aprendizaje base de un ensamble amplicador *sensible al costo*. Ellos incluyen en su estudio diversas variantes de ensambles amplicadores sensibles al costo. Cada variante introduce, de una forma diferente, información de costo dentro de la fórmula de actualización de pesos de AdaBoost. La Tabla 1.4 presenta las cuatro mejores variantes de AdaBoost sensible al costo reportadas por [50]. El costo del error de la clasicación del ejemplo X_i es representado por C_i. Para cada variante, la fórmula que le corresponde en la Tabla 1.4 sustituye a la Fórmula (1,2,2)en el cálculo de los pesos.

Nombre de la variante	Modicación en la fórmula de actualización de peso
Ab1	$K_t(x_i, y_i)$=$C_i \exp\left(-\alpha_t y_i h_t\left(X_i\right)\right)$
Ab2	$K_t(x_i, y_i)$=$C_i \exp\left(-C_i \alpha_t y_i h_t\left(X_i\right)\right)$
Ab3	$K_t(x_i, y_i)$=$C_i \exp\left(-C_i^2 \alpha_t y_i h_t\left(X_i\right)\right)$

Cuadro 1.4: Tres variantes de AdaBoost sensible al costo usadas para clasicación multinstancia en problemas con clases desbalanceadas.

Observe que las x_i de la Fórmula (1,2,2) , que representan las instancias, han sido sustituidas en la tabla por X_i, que representan bolsas. En [50] asumen que todos los ejemplos de una misma clase tienen igual costo. Como la matriz de costo puede ser escalada al multiplicar cada elemento de la matriz por una constante positiva, la búsqueda de la matriz óptima se reduce a encontrar la proporción adecuada entre el costo para la clase positiva y el costo para la clase negativa. La proporción se busca iterativamente tratando de maximizar una medida de desempeño.

En [50] se demuestra que el uso de amplificadores sensibles al costo mejora el desempeño de la clasificación cuando el conjunto de datos está desbalanceado.

En otro trabajo [56] abordan los problemas multinstancia y multietiqueta al aplicar directamente un SVM en un marco de regularización. En el aprendizaje MIME cada ejemplo de entrenamiento, además de contener muchas instancias (sin etiquetas), también pueden ser asignados a múltiples etiquetas de clase. El aprendizaje multinstancia puede considerarse como un caso particular del aprendizaje MIME donde el conjunto de etiquetas asignadas a cada bolsa contiene solo un elemento. El algoritmo D-MimlSvm que ellos proponen incorpora un mecanismo para tratar con el desbalance de clases. Basados en el método de rescalamiento [55] , ellos estiman la proporción de desbalance para cada etiqueta de clase y ponderan la función de pérdida de acuerdo a estas proporciones. Aunque la comparación hecha en [56] entre D-MimlSvm y otros métodos MIME (que no tienen en cuenta el problema del debalance de clases), reporta mejoras para D-MimlSvm, no queda claro hasta qué punto las mejoras observadas se

deben al mecanismo incluido para tratar el desbalance de clases o a las diferencias en las estrategias de aprendizaje. Observe que esta es una solución a nivel de algoritmo, por tanto con aplicabilidad limitada.

1.2.4.2. Algoritmo MISMOTE

El algoritmo MISMOTE (*Multi-Instance Synthetic Minority Over-sampling TEchnique*) [44] hace un sobremuestreo sintético de la clase minoritaria. Este es uno de los métodos de muestreo que se usan como base para desarrollar los ensambles que se proponen en la tesis.

La esencia del algoritmo MISMOTE consiste en tomar cada ejemplo de la clase minoritaria e introducir ejemplos sintéticos a lo largo del segmento de línea que une a algunos de los k vecinos más cercanos de la clase minoritaria. Más detalladamente, a partir de un ejemplo de entrenamiento P, se genera un nuevo ejemplo sintético S seleccionando aleatoriamente uno de los k vecinos más cercanos de P, que llamaremos Q, e interpolándolos a ambos P y Q. Se define el tamaño de una bolsa sintética S como el promedio de los tamaños de las $k!$ bolsas más cercanas a la bolsa de entrenamiento que da lugar a S. El número $k!$ de vecinos más cercanos dene el alcance del concepto de localidad en el espacio de bolsa y, aunque no necesariamente tiene que ser el mismo número k de vecinos más cercanos a partir de los cuales se selecciona la bolsa involucrada en la interpolación, en [44] se hace $k! = k$.

Cada instancia sintética dentro la bolsa generada se obtiene muestreando aleatoriamente una instancia de P y otra de Q, e interpolándolas. El algoritmo se basa solamente en la información de las T bolsas de entrenamiento $D=\{X_1, ..., X_T\}$ de la clase minoritaria para generar bolsas sintéticas. Se asume que para cada bolsa original se generan N bolsas sintéticas, aunque realmente esto solo sucede cuando el número total de bolsas a ser generadas es un múltiplo entero de T. En otro caso, después de repartir equitativamente la responsabilidad de generar las bolsas sintéticas entre las T bolsas de entrenamiento, el resto de las bolsas sintéticas serán generadas por bolsas de entrenamiento seleccionadas aleatoriamente (sin remplazo). Por ejemplo, si se tienen 10 bolsas de entrenamiento y se desean generar 25 bolsas sintéticas, se generarán dos bolsas sintéticas a partir de cada bolsa de entrenamiento y además otras 5 bolsas sintéticas serán generadas por 5 bolsas de entrenamiento seleccionadas al azar.

El algoritmo itera sobre las T bolsas de entrenamiento de la clase minoritaria. Se determinan los k vecinos más cercanos de la clase minoritaria para cada bolsa de entrenamiento X_i, utilizando la distancia de Hausdor, y se calcula el tamaño que será asignado a las bolsas sintéticas generadas a partir de X_i. El número k de vecinos más cercanos es un parámetro del algoritmo. Luego se procede a generar las bolsas sintéticas usando la interpolación de bolsas definida.

El método preserva la distribución local del tamaño de la bolsa y es independiente de cualquier hipótesis multinstancia. En [44] se demuestra la efectividad y competitividad del nuevo método de preprocesamiento.

Capítulo 2

Clasificadores ensamblados con técnicas de muestreo en la clasificación multinstancia

En el presente capítulo se pasará a explicar los algoritmos ensamblados desarrollados primero a partir de Bagging y luego de Boosting, partiendo desde los algoritmos originales.

El objetivo principal de la metodología utilizada por los clasificadores ensamblados es tratar de mejorar el desempeño de un algoritmo de clasificación entrenando varios clasificadores y combinando sus predicciones para obtener un clasificador más preciso que cada uno de ellos por separado. La idea básica es que después de construir cada clasificador con el conjunto de datos de entrenamiento, agrupar sus predicciones cuando aparezcan instancias con etiqueta de clase desconocidas. Esta idea surge para mejorar la habilidad de generalización: cada clasificador individual comete errores, pero debido a que son diferentes, los ejemplos clasificados incorrectamente no necesariamente son los mismos [25].

A pesar de no haber sido demostrado teóricamente, la diversidad entre los clasificadores individuales es crucial para formar un clasificador ensamblado, como se puede observar en la literatura. [48, 27]. Existen varias maneras para alcanzar la diversidad requerida, pero se recomienda que el clasificador base debe ser *débil*; un clasificador es llamado *débil* cuando pocos cambios en el conjunto de datos de entrenamiento producen grandes cambios en el modelo inducido. A continuación se explican los algoritmos Bagging y Boosting.

2.1. Algoritmos basados en Bagging

A continuación se pasa a describir el algoritmo Bagging en la sección 2.1.1, después en la sección 2.1.2 se presenta dicho algoritmo adaptado para introducir técnicas de muestreo y en las secciones posteriores se describen los algoritmos

desarrollados por el autor.

2.1.1. Algoritmo Bagging

Mimbrean [5] introdujo el concepto de agrupamiento aleatorio para construir clasificadores ensamblados. En la figura 2.1.1 se puede observar una representación esquemática del algoritmo Bagging. En una primera etapa se tiene el conjunto de datos de entrada D=$\{(X_1, y_1), ..., (X_M, y_M)\}$. Luego se aplica un muestreo aleatorio con reemplazamiento, también conocido como bootstrapping. Al aplicar bootstrapping se obtiene una variación del conjunto inicial D, que constituye los datos de entrenamiento utilizados para entrenar cada clasificador base h_t. El proceso de muestreo y entrenamiento es repetido T veces, formando el clasificador ensamblado H.

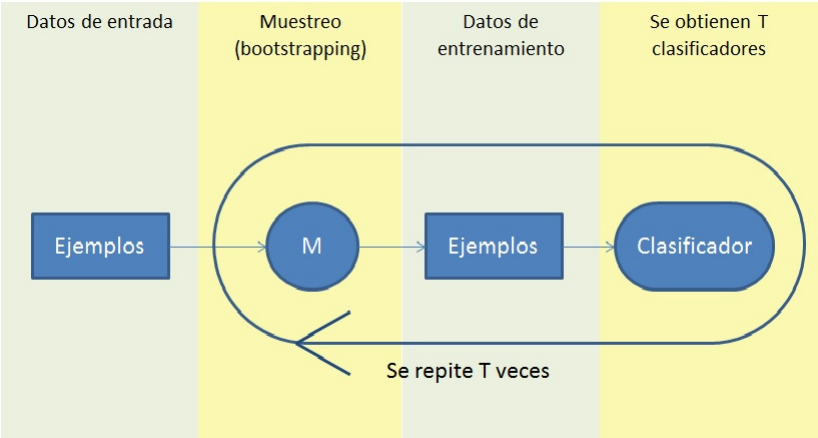


Figura 2.1.1: Bagging

En la etapa de prueba, cuando aparece una instancia de la cuál se desconoce su etiqueta se presenta a cada clasificador h_t, los cuales agregan su voto por mayoría para clasificar dicha instancia ($H(x)$=$sign(\sum_{t=1}^{T} h_t(x))$).

A continuación se explica el algoritmo Bootstrap (algoritmo 1) usado para el remuestreo con reemplazamiento. En cada iteración t (iteraciones que dependen del tamaño del submuestreo aleatorio) de este algoritmo, se toma un número aleatorio r que se encuentre en el rango del tamaño del conjunto de datos y se añade la instancia ubicada en ese índice del conjunto a $D^{'}$ que es el conjunto

resultante.

Input:
tamaño n del bootstrap
conjunto $D = \{X_1, \ldots, X_M\}$ de entrenamiento
Output:
conjunto D' de entrenamiento remuestreado por bootstrap

```
1 D' ← ∅ inicializar conjunto de ejemplos remuestreados
2 for i ← 1 to n do
3 |   r ← número aleatorio entre 1 y M
4 |   añadir X_r a D'
5 return D'
```

Algorithm 1: Bootstrap

2.1.2. Propuestas de Bagging con técnicas de muestreo

La figura 2.1.2 muestra la idea general de la propuesta para incorporar técnicas de muestreo al algoritmo Bagging original con el objetivo de tratar el problema de clases desbalanceadas. En este caso, luego de entrar el conjunto de entrenamiento al algoritmo, es necesario segregarlo en dos conjuntos: el de los ejemplos positivos y el de los ejemplos negativos. El paso de segregar el conjunto por clases no se hace en el Bagging original, el cuál trata todos los ejemplos por igual.

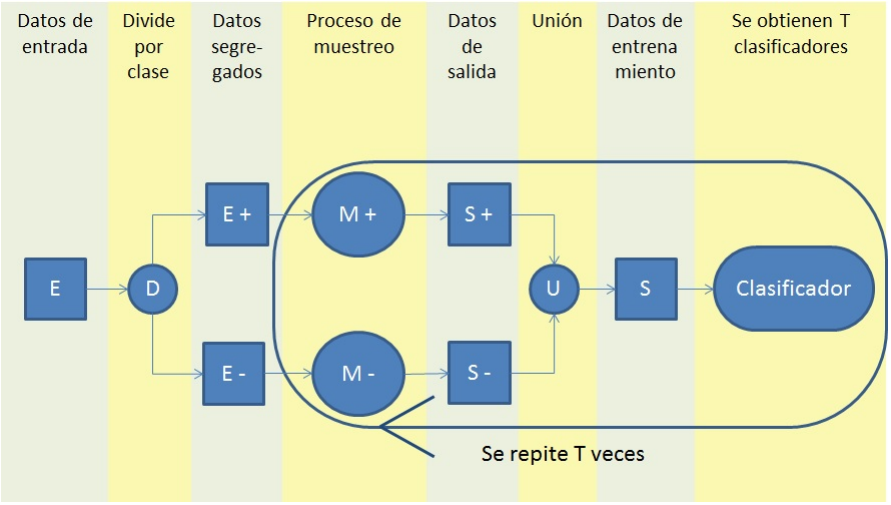


Figura 2.1.2: Bagging con técnicas de muestreo

Después se aplica el proceso de muestreo diferenciado por clases para tratar de equilibrar las distribuciones de las mismas, utilizando en esta etapa tres variantes: el submuestreo de la clase mayoritaria, el sobremuestreo con ejemplos

sintéticos de la clase minoritaria y la combinación de las mismas. Luego se unen los dos conjuntos para formar el que sirve de entrada para entrenar cada clasificador base. La etapa de prueba no sufre cambios en cuanto al Bagging clásico, por tanto no cambia en ninguno de los algoritmos desarrollados en esta investigación.

Luego de explicar cómo se puede adaptar el algoritmo Bagging clásico para introducirle técnicas de muestreo, se pasará a describir los algoritmos MIRUBagging (sección 2.1.3) que utiliza el submuestreo aleatorio de la clase mayoritaria, MISOBagging (sección 2.1.4) que introduce el sobremuestreo con ejemplos sintéticos de la clase positiva y el MISORUBagging (sección 2.1.4) que combina las dos técnicas utilizadas por los algoritmos mencionados anteriormente, el submuestreo aleatorio de la clase negativa y el sobremuestreo con ejemplos sintéticos de la clase positiva.

2.1.3. MIRUBagging

MIRUBagging, acrónimo de *Multi-Instance Random Undersampling Bagging*, es un ensamble de clasificadores que incorpora una solución para enfrentar el problema de clases desbalanceadas descrito por el algoritmo 2. Consiste en un submuestreo aleatorio (o bootstrapping) de la clase negativa (mayoritaria) en cada iteración del algoritmo bagging.

En los pasos 1 y 2 se hace la segregación de los datos para aplicar la técnica de muestreo sobre los datos divididos por clase.

El parámetro γ es el por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase positiva. Por ejemplo, si el conjunto de entrada tiene 40 ejemplos positivos ($p = 40$) y se escoge $\gamma = 50\,\%$ entonces se muestrearán 40 ejemplos de la clase negativa ($n' = 40$), de forma que cada clase contiene la mitad de ejemplos del total. Mientra que si se escoge $\gamma = 40\,\%$ entonces se muestrearán 60 ejemplos de la clase negativa ($n' = 60$).

El parámetro σ permite introducir cierta variabilidad en el por ciento de ejemplos positivos n' en cada iteración del algoritmo, de forma que n' es realmente una variable aleatoria con distribución $N(\gamma, \sigma)$ (líneas 7-9). Si $\sigma = 0$ no habrá ninguna variación, y n' será en cada iteración como fue descrito en el párrafo anterior (línea 4).

En cada iteración se aplica bootstrapping a la clase positiva (línea 10) generándose el mismo número de ejemplos de la clase positiva original, de forma que una parte de los ejemplos incluidos en la nueva clase positiva estarán duplicados. Se aplica bootstrapping sobre la clase positiva para introducir la variabilidad necesaria para el algoritmo Bagging aunque se mantengan el mismo número de ejemplos originales .

La selección aleatoria de los ejemplos de la clase negativa se hace con *bootstrapping* con tamaño $n^{'}$ (línea 11) .

En la línea 12 se unen los datos segregados en el conjunto D_t que sirve de entrada para entrenar el clasificador base $h_t$en la línea 13.

Como este algoritmo realiza un submuestreo de la clase mayoritaria o negativa, la variable n' representa el valor que equilibra los tamaños de las dos

clases, reduciendo a n' el tamaño de la clase negativa.

Input:

- $D = \{X_1, \ldots, X_M\}$ conjunto de ejemplos de entrenamiento
- T número de iteraciones (def: $T = 10$)
- γ por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase minoritaria (def: $\gamma = 50$)
- σ grado de perturbación de γ (def: $\sigma = 5$)
- I algoritmo de clasificación de aprendizaje débil

Output:

- $H(X) = \text{sign}\left(\sum_{t=1}^{T} h_t(X)\right)$ clasificador ensamblado donde $h_t(X) \in [-1, 1]$ son los clasificadores inducidos

```
1  D^P ← conjunto de ejemplos positivos
2  D^N ← conjunto de ejemplos negativos
3  p ← |D^P| número de ejemplos positivos de entrada
4  if σ = 0 then n' ← p(100/γ − 1) número de ejemplos negativos de
   salida
5  for t ← 1 to T do
6      if σ > 0 then
7          χ ← N(0,1) variable aleatoria con distribución normal
8          Γ ← γ + σχ
9          n' ← p(100/Γ − 1)
10     D_t^P ← Bootstrap (p, D^P)
11     D_t^N ← Bootstrap (n', D^N)
12     D_t ← D_t^P ∪ D_t^N
13     h_t ← I(D_t)
```

Algorithm 2: MIRUBagging.

Después de realizar el remuestreo sobre los conjuntos de ejemplos positivos y negativos se procede a realizar la unión de dichos conjuntos (línea 12) y con el conjunto resultante es que se procede a entrenar el clasificador débil que recibe el anterior algoritmo como entrada (línea 13).

2.1.4. MISOBagging

MISOBagging, acrónimo de *Multi-Instance Synthetic Oversampling Bagging*, es un ensamble de clasificadores que incorpora una solución para enfrentar el problema de clases desbalanceadas descrito por el algoritmo 3. Consiste en un sobremuestreo con ejemplos sintéticos en la clase positiva (minoritaria) en cada

etapa del algoritmo bagging.

En los pasos 1 y 2 se hace la segregación de los datos para aplicar la técnica de muestreo sobre los datos divididos por clase.

El parámetro γ es el por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase positiva. Por ejemplo, si el conjunto de entrada tiene 20 ejemplos positivos y 40 negativos, y se escoge $\gamma = 50\,\%$, entonces se muestrearán 40 ejemplos de la clase positiva ($p' = 40$), de forma que cada clase contiene la mitad de ejemplos del total. Siguiendo con este ejemplo, los primeros 20 ejemplos positivos se obtienen por bootstrapping de los ejemplos de entrenamiento positivos (línea 11), mientras que el resto (los otros 20 en este ejemplo) se obtienen aplicando MISMOTE [44] a los ejemplos de entrenamiento positivos (línea 12).

El parámetro σ permite introducir cierta variabilidad en el por ciento de ejemplos positivos p' en cada iteración del algoritmo, de forma que p' es realmente una variable aleatoria con distribución $N(\gamma, \sigma)$ (líneas 8-10). Esto quiere decir que si se escoge $\gamma = 50\,\%$ y $\sigma = 5$, el por ciento de ejemplos positivos p' del total de ejemplos de entrenamiento se distribuirá normalmente alrededor del $50\,\%$ con una desviación estándar del $5\,\%$. Si $\sigma = 0$ no habrá ninguna variación, y p' será en cada iteración como el descrito en el párrafo anterior (línea 5).

En cada iteración se aplica bootstrapping a la clase negativa (línea 13) generándose el mismo número de ejemplos de la clase negativa original, de forma que una parte de los ejemplos incluidos en la nueva clase negativa estarán duplicados. Se aplica bootstrapping sobre la clase negativa para introducir la variabilidad necesaria para el algoritmo Bagging aunque se mantengan el mismo número de ejemplos originales .

En la línea 14 se unen los datos segregados en el conjunto D_t que sirve de entrada para entrenar el clasificador base $h_t$en la línea 15.

Como este algoritmo realiza un sobremuestreo con ejemplos sintéticos de la clase minoritaria o positiva, se aplica MISMOTE para, al incorporar a los ejemplos positivos del conjunto de entrenamiento original los ejemplos positivos

sintéticos generados, se logre equilibrar la clase positiva con la negativa.

Input:

- $D = \{X_1, \ldots, X_M\}$ conjunto de ejemplos de entrenamiento
- T número de iteraciones
- γ por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase minoritaria (def: $\gamma = 50$)
- σ grado de perturbación de γ (def: $\sigma = 5$)
- I algoritmo de clasificación de aprendizaje débil

Output:

- $H(X) = \text{sign}\left(\sum_{t=1}^{T} h_t(X)\right)$ clasificador ensamblado donde $h_t(X) \in [-1, 1]$ son los clasificadores inducidos

$D^P \leftarrow$ conjunto de ejemplos positivos
$D^N \leftarrow$ conjunto de ejemplos negativos
$p \leftarrow \left|D^P\right|$ número de ejemplos positivos de entrada
$n \leftarrow \left|D^N\right|$ número de ejemplos negativos de entrada
if $\sigma = 0$ **then** $p' \leftarrow \frac{n\gamma}{100-\gamma}$ número de ejemplos positivos de salida
for $t \leftarrow 1$ **to** T **do**
  if $\sigma > 0$ **then**
    $\chi \leftarrow N(0,1)$ variable aleatoria con distribución normal
    $\Gamma \leftarrow \gamma + \sigma\chi$
    $p' \leftarrow \frac{n\Gamma}{100-\Gamma}$
  $D_t^P \leftarrow$ `Bootstrap` (p, D^P)
  $S_t \leftarrow$ `MISMOTE` $(p' - p, D^P)$
  $D_t^N \leftarrow$ `Bootstrap` (n, D^N)
  $D_t \leftarrow D_t^P \cup S_t \cup D_t^N$
  $h_t \leftarrow I(D_t)$

Algorithm 3: MISOBagging

El algoritmo *MISMOTE* devuelve un conjunto de instancias generadas sintéticamente de tamaño $p^{'} - p$ en cada iteración, a partir de un conjunto pasado por parámetro.

2.1.5. MISORUBagging

MISORUBagging, acrónimo de *Multi-Instance Synthetic Oversampling Random Undersampling Bagging*, es un ensamble de clasificadores que incorpora una solución para enfrentar el problema de clases desbalanceadas descrito por el Algoritmo 4. Consiste en un sobremuestreo con ejemplos sintéticos en la clase

positiva (minoritaria) y al mismo tiempo un submuestreo aleatorio en la clase negativa (mayoritaria) en cada etapa del algoritmo bagging.

En los pasos 1 y 2 se hace la segregación de los datos para aplicar la técnica de muestreo sobre los datos divididos por clase.

En la línea 5 se calcula el número de ejemplos de entrenamiento de la clase negativa que se reduce mediante muestreo aleatorio en un por ciento indicado por el parámetro β. El muestreo se hace con remplazo (bootstrapping) en la línea 14.

El parámetro γ es el por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase positiva después de que el número de ejemplos de entrenamiento negativos fue reducido por submuestreo. El siguiente ejemplo muestra cómo funciona el algoritmo MISORUBagging. Sea un conjunto de entrada con 20 ejemplos positivos y 100 negativos, y se escogen los parámetros $\beta = 40\,\%$ y $\gamma = 50\,\%$. Entonces, el número de ejemplos de entrenamiento negativos se reduce al $40\,\%$, con lo cual quedan 40 ejemplos ($n' = 40$) y luego se muestrean 40 ejemplos de la clase positiva ($p' = 40$), de forma que cada clase contiene la mitad de ejemplos del total. Los primeros 20 ejemplos positivos se obtienen por bootstrapping de los ejemplos de entrenamiento positivos (línea 12), mientras que el resto (los otros 20 en este ejemplo) se obtienen aplicando MISMOTE (línea 13) a los ejemplos de entrenamiento positivos.

El parámetro σ permite introducir cierta variabilidad en el por ciento de ejemplos positivos p' en cada iteración del algoritmo, de forma que p' es realmente una variable aleatoria con distribución $N(\gamma, \sigma)$ (líneas 9-11). Esto quiere decir que si se escoge $\gamma = 50\,\%$ y $\sigma = 5$, el por ciento de ejemplos positivos p' del total de ejemplos de entrenamiento se distribuirá normalmente alrededor del $50\,\%$ con una desviación estándar del $5\,\%$. Si $\sigma = 0$ no habrá ninguna variación, y p' será en cada iteración como el descrito en el párrafo anterior (línea 6).

En la línea 15 se unen los datos segregados en el conjunto D_t que sirve de entrada para entrenar el clasificador base $h_t$en la línea 16.

Este algoritmo combina las técnicas de submuestreo de la clase mayoritaria y sobremuestreo de la minoritaria. Se submuestrea la clase negativa con un tamaño $n^{'}$, reduciendo así su tamaño y se sobremuestrea la clase positiva y se le incorporan los ejemplos sintéticos obtenidos con MISMOTE a los ejemplos positivos originales, logrando así un equilibrio de las distribuciones de las dos

clases.

Input:

- $D = \{X_1, \ldots, X_M\}$ conjunto de ejemplos de entrenamiento
- T número de iteraciones
- β por ciento deseado de reducción del número de ejemplos de entrenamiento de la clase mayoritaria
- γ por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase minoritaria (def: $\gamma = 50$)
- σ grado de perturbación de γ (def: $\sigma = 5$)
- I algoritmo de clasificación de aprendizaje débil

Output:

- $H(X) = \text{sign}\left(\sum_{t=1}^{T} h_t(X)\right)$ clasificador ensamblado donde $h_t(X) \in [-1, 1]$ son los clasificadores inducidos

$D^P \leftarrow$ conjunto de ejemplos positivos
$D^N \leftarrow$ conjunto de ejemplos negativos
$p \leftarrow \left|D^P\right|$ número de ejemplos positivos de entrada
$n \leftarrow \left|D^N\right|$ número de ejemplos negativos de entrada
$n' \leftarrow \frac{n\beta}{100}$ número de ejemplos negativos de salida
if $\sigma = 0$ **then** $p' \leftarrow \frac{n'\gamma}{100-\gamma}$ número de ejemplos positivos de salida
for $t \leftarrow 1$ **to** T **do**
  if $\sigma > 0$ **then**
    $\chi \leftarrow N(0, 1)$ variable aleatoria con distribución normal
    $\Gamma \leftarrow \gamma + \sigma\chi$
    $p' \leftarrow \frac{n'\Gamma}{100-\Gamma}$
  $D_t^P \leftarrow$ `Bootstrap` (p, D^P)
  $S_t \leftarrow$ `MISMOTE` $(p' - p, D^P)$
  $D_t^N \leftarrow$ `Bootstrap` (n', D^N)
  $D_t \leftarrow D_t^P \cup S_t \cup D_t^N$
  $h_t \leftarrow I(D_t)$

Algorithm 4: MISORUBagging

2.2. Algoritmos basados en Boosting

El algoritmo *boosting* fue planteado por Schapire en [42]. Schapire demostró que un clasificador débil (el cuál es ligeramente mejor que la suposición aleatoria) se puede convertir en uno fuerte en sentido de la *aproximación probabilística*

correcta (PAC por sus siglas en inglés).

Existen varios algoritmos representativos del algoritmos Boosting dentro de los que se encuentran el *AdaBoost,AdaBoostM1* [12] y el *AdaBoostM2* [41].

A continuación se describe el algoritmo AdaBoost original en la sección 2.2.1, después en la sección 2.2.2 se presenta dicho algoritmo adaptado para introducir técnicas de muestreo y en las secciones posteriores se describen los algoritmos desarrollados por el autor.

2.2.1. AdaBoost

AdaBoost, representado en la figura 2.2.1, parte de un conjunto de datos de entrenamiento D=$\{(X_1, y_1), ..., (X_M, y_M)\}$. Inicialmente se realiza un cálculo del peso de las instancias el cuál es igual para cada una. Después se pasa a entrenar cada clasificador base h_t que tiene como entrada el conjunto de datos D y su vector de pesos asociado D_t, para luego ser probado sobre todo el conjunto de datos de entrenamiento. Las etapas del cálculo de pesos y el de entrenamiento y prueba de cada clasificador base son repetidas T veces. En cada iteración t se actualiza el vector de pesos de forma tal que que aumente el peso de las instancias clasificadas erróneamente y que disminuya el peso de las instancias clasificadas correctamente en la iteración anterior. Así, en la próxima iteración t+*1* se hace énfasis en las instancias clasificadas de forma incorrecta. Para la actualización de los pesos se usa la pseudo-pérdida ϵ de cada clasificador h_t.

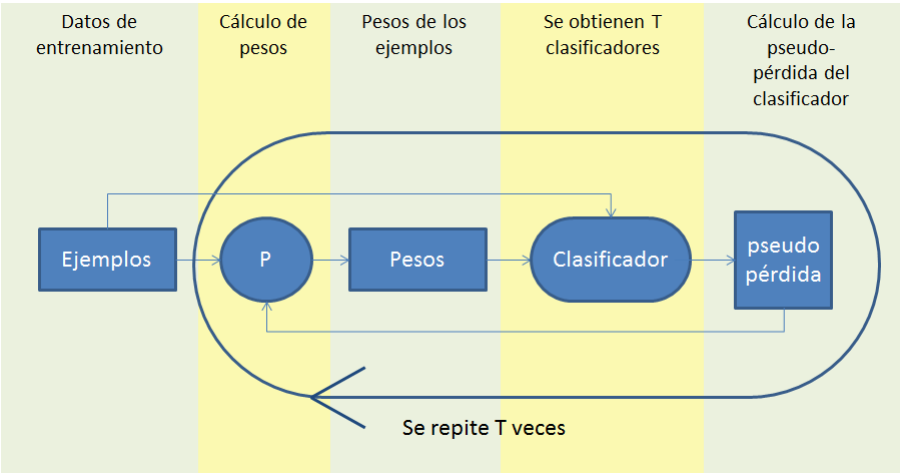


Figura 2.2.1: AdaBoost

Además, otro peso α_t es asignado a cada clasificador individual dependiendo de su precisión. α_t es después usado en la fase de prueba del clasificador ensamblado H; esto se hace con el objetivo de brindar mayor «confianza» a los clasificadores más precisos. Finalmente el clasificador ensamblado (*boosted*) es obtenido al sumar los votos de todos los clasificadores entrenados donde el

voto vale α_t unidades. AdaBoost es descrito por el algoritmo 5 y es explicado a continuación.

Input:
conjunto $D = \{(X_1, y_1), \ldots, (X_M, y_M)\}$ de ejemplos de entrenamiento con etiquetas de clase $y_i \in Y = 1, \ldots, k$
número T de iteraciones
algoritmo I de aprendizaje débil de clasificación

Output:
clasificador amplificado (boosted):

$$H(X) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t h_t(X, y)$$

1 $\mathcal{D}_1(i) \leftarrow 1/M$
2 **for** $t \leftarrow 1$ **to** T **do**
3 $\quad h_t \leftarrow I(D, \mathcal{D}_t)$
4 $\quad \epsilon \leftarrow \sum_{i, y_i \neq h_t(x_i)}^{M} \mathcal{D}_t(i)$
5 $\quad$ **if** $\epsilon > 0{,}5$ **then**
6 $\quad\quad T \leftarrow t - 1$
7 $\quad \alpha_t \leftarrow \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$
8 $\quad D_{t+1}(i) \leftarrow D_t(i) e^{(-\alpha_t h_t(x_i) y_i)}$
9 $\quad$ Normalizar $D_{t+1}(i)$

Algorithm 5: AdaBoost

En el paso 1 del algoritmo 6 se asigna el peso inicial para cada instancia del conjunto de entrenamiento. Dicho peso es igual para todos los ejemplos y tiene un valor de $1/M$ donde M es el número de ejemplos de entrenamiento. En cada iteración t se entrena un clasificador h_t con el conjunto de entrenamiento original y su vector de pesos asociados.

Posteriormente se calcula la pseudo-pérdida ϵ mediante la fórmula ubicada en la línea 4. ϵ es el error del clasificador actual y consiste en la suma de los pesos de las instancias que fueron clasificadas incorrectamente. Si $\epsilon_t$es mayor que 0.5 las pruebas son terminadas y T pasa a ser *t-1*. En otro caso se pasa a calcular el vector de los pesos para la próxima iteración.

Para calcular el vector de pesos es necesario calcular el factor α_t. Este factor es el peso asignado a cada clasificador base y es calculado mediante la fórmula en la línea 7. El vector de pesos se obtiene multiplicando los pesos de las instancias que fueron clasificadas correctamente por el factor α_t y luego se normaliza.

2.2.2. Propuestas de Boost con técnicas de muestreo

La figura 2.2.2 muestra la idea general de la propuesta para incorporar técnicas de muestreo al algoritmo AdaBoost original con el objetivo de tratar el problema de clases desbalanceadas. En este caso, luego de entrar el conjunto de entrenamiento al algoritmo, es necesario segregarlo en dos conjuntos: el de los

ejemplos positivos y el de los ejemplos negativos. El paso de segregar el conjunto por clases no se hace en el AdaBoost original, el cuál trata todos los ejemplos por igual.

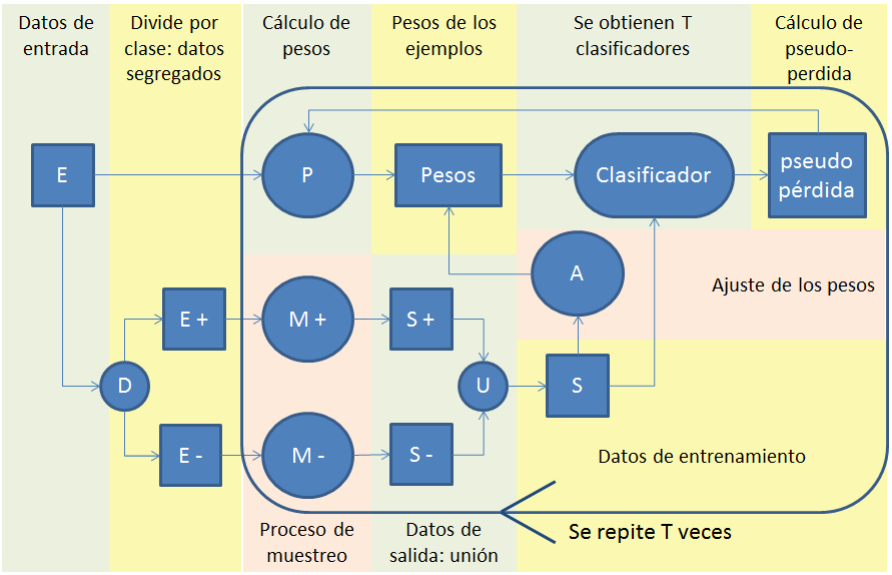


Figura 2.2.2: AdaBoost con técnicas de muestreo.

Después se aplica el proceso de muestreo diferenciado por clases para tratar de equilibrar las distribuciones de las mismas utilizando en esta etapa tres variantes: el submuestreo de la clase mayoritaria, el sobremuestreo con ejemplos sintéticos de la clase minoritaria y la combinación de las mismas.Luego se unen los dos conjuntos para formar el que sirve de entrada para entrenar cada clasificador base, junto con su vector de pesos asociado. Luego que de entrenarse cada clasificador es calculada la pseudo-pérdida y los pesos son recalculados como en el AdaBoost original. La etapa de prueba no sufre cambios en cuanto al AdaBoost clásico, por tanto no cambia en ninguno de los algoritmos desarrollados en esta investigación.

Luego de explicar cómo se puede adaptar el algoritmo AdaBoost clásico para introducirle técnicas de muestreo, se pasará a describir los algoritmos MIRUBoosting (sección 2.2.3) que utiliza el submuestreo aleatorio de la clase mayoritaria, MISOBoosting (sección 2.2.4) que introduce el sobremuestreo con ejemplos sintéticos de la clase positiva y el MISORUBoosting (sección 2.2.4) que combina las dos técnicas utilizadas por los algoritmos mencionados anteriormente, el submuestreo aleatorio de la clase negativa y el sobremuestreo con ejemplos sintéticos de la clase positiva.

2.2.3. MIRUBoost

MIRUBoost, acrónimo de *Multi-Instance Random Undersampling Boosting*, es un ensamble de clasificadores que incorpora una solución para enfrentar el problema de clases desbalanceadas descrito por el Algoritmo 6. Consiste en un submuestreo aleatorio (o bootstrapping) de la clase negativa (mayoritaria) en cada iteración del algoritmo boosting.

En los pasos 1 y 2 se hace la segregación de los datos para aplicar la técnica de muestreo sobre los datos divididos por clase. En el paso 4 se le asigna el peso inicial a cada ejemplo del conjunto de entrenamiento.

El parámetro γ es el por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase positiva. Por ejemplo, si el conjunto de entrada tiene 40 ejemplos positivos ($p = 40$) y se escoge $\gamma = 50\,\%$ entonces se muestrearán 40 ejemplos de la clase negativa ($n' = 40$), de forma que cada clase contiene la mitad de ejemplos del total. Mientra que si se escoge $\gamma = 40\,\%$ entonces se muestrearán 60 ejemplos de la clase negativa ($n' = 60$).

El parámetro σ permite introducir cierta variabilidad en el por ciento de ejemplos positivos n' en cada iteración del algoritmo, de forma que n' es realmente una variable aleatoria con distribución $N(\gamma, \sigma)$ (líneas 6-8).

En cada iteración se mantiene el número inicial de ejemplos de la clase positiva. La selección aleatoria de los ejemplos de la clase negativa se hace sin reemplazo, utilizando el algoritmo *Submuestrear* (línea 9).

En la línea 10 se unen los datos segregados en el conjunto D_t .

En la línea 11 se hace el recálculo de los pesos para el conjunto D_t. Para cada instancia, si ella se encuentra en el conjunto D_t se le asigna su peso multiplicado por un factor de normalización al vector de pesos $\mathcal{D}'_t$.

El conjunto D_t y su distribución de pesos $\mathcal{D}'_t$ sirven de entrada para entrenar el clasificador base $h_t$en la línea 12.

Como este algoritmo realiza un submuestreo de la clase mayoritaria o negativa, la variable n' representa el valor que equilibra los tamaños de las dos

clases, reduciendo a n' el tamaño de la clase negativa.

Input:

- $D = \{(X_1, y_1), \ldots, (X_M, y_M)\}$ conjunto de ejemplos de entrenamiento con etiquetas de clase $y_i \in Y = 1, \ldots, k$
- T número de iteraciones (def: $T = 10$)
- γ por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase minoritaria (def: $\gamma = 50$)
- σ grado de perturbación de γ (def: $\sigma = 5$)
- I algoritmo de clasificación de aprendizaje débil

Output:

- $H(X) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t h_t(X, y)$ clasificador ensamblado donde $h_t(X, y) \in [-1, 1]$ son los clasificadores inducidos

1 $D^P \leftarrow$ conjunto de ejemplos positivos
2 $D^N \leftarrow$ conjunto de ejemplos negativos
3 $p \leftarrow \left|D^P\right|$ número de ejemplos positivos de entrada
4 $\mathcal{D}_1(i) \leftarrow 1/M$
5 **for** $t \leftarrow 1$ **to** T **do**
6 $\quad \chi \leftarrow N(0, 1)$ variable aleatoria con distribución normal
7 $\quad \Gamma \leftarrow \gamma + \sigma\chi$
8 $\quad n' \leftarrow p\left(\frac{100}{\Gamma} - 1\right)$
9 $\quad D_t^N \leftarrow$ `Submuestrear` (n', D^N)
10 $\quad D_t \leftarrow D^P \cup D_t^N$
11 $\quad \mathcal{D}'_t(i) \leftarrow$recálculo de pesos
12 $\quad h_t \leftarrow I(D_t, \mathcal{D}'_t)$
13 $\quad \epsilon \leftarrow \sum_{i, y_i \neq h_t(x_i)}^{M} \mathcal{D}_t(i)$
14 $\quad$ **if** $\epsilon > 0{,}5$ **then**
15 $\quad\quad T \leftarrow t - 1$
16 $\quad\quad$ return
17 $\quad \alpha_t \leftarrow \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$
18 $\quad \mathcal{D}_{t+1}(i) \leftarrow \mathcal{D}_t(i)\, e^{(-\alpha_t h_t(x_i) y_i)}$
19 $\quad$ Normalizar $\mathcal{D}_{t+1}(i)$

Algorithm 6: MIRUBoost

Después de cada iteración se calcula la pseudo-pérdida (línea 13) y se recalculan los pesos de todos los ejemplos de entrenamiento multiplicando los pesos de las instancias que fueron clasificadas correctamente por el factor α_t (línea 18) y luego se normaliza (línea 19).

El algoritmo *Submuestrear* (algoritmo 7) es utilizado para realizar un submuestreo aleatorio sin reemplazo sobre un conjunto de datos.

Input:

- n tamaño del submuestreo
- $D = \{X_1, \ldots, X_M\}$ conjunto de ejemplos originales

Output:

- D' conjunto de ejemplos submuestreados

```
1 comprobar que M ≥ n, en caso contrario emitir un error
2 pool ← hacer una copia de D
3 D' ← ∅ inicializar conjunto de ejemplos submuestreados
4 for i ← 1 to n do
5 |   seleccionar aleatoriamente un ejemplo X de pool
6 |   añadir X a D'
7 |   remover a X de pool
8 return D'
```

Algorithm 7: Submuestrear.

El algoritmo anterior recibe como entrada un conjunto de instancias D y la cantidad de ejemplos n que se quiere obtener después de aplicar el submuestreo, el cuál no puede ser mayor que el total de instancias del conjunto a submuestrear. A continuación en cada iteración se selecciona aleatoriamente una instancia *X,* se añade al conjunto resultado $D^{'}$ y se elimina del conjunto *pool,* garantizando así el muestreo sin reemplazamiento.

2.2.4. MISOBoost

MISOBoost, acrónimo de *Multi-Instance Synthetic Oversampling Boosting*, es un ensamble de clasificadores que incorpora una solución para enfrentar el problema de clases desbalanceadas descrito por el Algoritmo 8. Consiste en un sobremuestreo aleatorio con ejemplos sintéticos en la clase positiva (minoritaria) en cada etapa del algoritmo boosting.

En los pasos 1 y 2 se hace la segregación de los datos para aplicar la técnica de muestreo sobre los datos divididos por clase. En el paso 5 se le asigna el peso inicial a cada ejemplo del conjunto de entrenamiento.

En cada iteración se aplica bootstrapping a la clase negativa (línea 10) generándose el mismo número de ejemplos de la clase negativa original, de forma que una parte de los ejemplos incluidos en la nueva clase negativa estarán duplicados. Se aplica bootstrapping sobre la clase negativa para introducir la variabilidad necesaria para el algoritmo Boosting aunque se mantengan el mismo número de ejemplos originales. En cada iteración se mantiene el número inicial de ejemplos de la clase positiva (línea 11).

El parámetro γ es el por ciento deseado del total de ejemplos de entrena-

miento a ser representado por la clase positiva. Por ejemplo, si el conjunto de entrada tiene 20 ejemplos positivos y 40 negativos, y se escoge $\gamma = 50\,\%$, entonces se muestrearán 40 ejemplos de la clase positiva ($p' = 40$), de forma que cada clase contiene la mitad de ejemplos del total. Siguiendo con este ejemplo, los primeros 20 ejemplos positivos se obtienen de los ejemplos de entrenamiento positivos originales (línea 11), mientras que el resto (los otros 20 en este ejemplo) se obtienen aplicando MISMOTE a los ejemplos de entrenamiento positivos (línea 12).

El parámetro σ permite introducir cierta variabilidad en el por ciento de ejemplos positivos p' en cada iteración del algoritmo, de forma que p' es realmente una variable aleatoria con distribución $N(\gamma, \sigma)$ (líneas 7-9). Esto quiere decir que si se escoge $\gamma = 50\,\%$ y $\sigma = 5$, el por ciento de ejemplos positivos p' del total de ejemplos de entrenamiento se distribuirá normalmente alrededor del $50\,\%$ con una desviación estándar del $5\,\%$.

En la línea 13 se unen los datos segregados en el conjunto D_t.

En la línea 14 se hace el recálculo de los pesos para el conjunto D_t. Para cada ejemplo obtenido a partir del conjunto de datos de entrenamiento inicial que está en D_t se le asigna su peso multiplicado por un factor de normalización al vector de pesos $\mathcal{D}'_t$. En el caso de los nuevos ejemplos sintéticos se les asigna un peso con valor $1/K$ donde K es el tamaño del conjunto D_t.

El conjunto D_t y su distribución de pesos $\mathcal{D}'_t$ sirven de entrada para entrenar el clasificador base $h_t$en la línea 15

Como este algoritmo realiza un sobremuestreo con ejemplos sintéticos de la clase minoritaria o positiva, se aplica MISMOTE para, al incorporar a los ejemplos positivos del conjunto de entrenamiento original los ejemplos positivos

sintéticos generados, se logre equilibrar la clase positiva con la negativa.

Input:

- $D = \{(X_1, y_1), \ldots, (X_M, y_M)\}$ conjunto de ejemplos de entrenamiento con etiquetas de clase $y_i \in Y = 1, \ldots, k$
- T número de iteraciones (def: $T = 10$)
- γ por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase minoritaria (def: $\gamma = 50$)
- σ grado de perturbación de γ (def: $\sigma = 5$)
- I algoritmo de clasificación de aprendizaje débil

Output:

- $H(X) = \arg\max_{y \in Y} \sum_{t=1}^{T} \alpha_t h_t(X, y)$ clasificador ensamblado donde $h_t(X, y) \in [-1, 1]$ son los clasificadores inducidos

$D^P \leftarrow$ conjunto de ejemplos positivos
$D^N \leftarrow$ conjunto de ejemplos negativos
$p \leftarrow \left|D^P\right|$ número de ejemplos positivos de entrada
$n \leftarrow \left|D^N\right|$ número de ejemplos negativos de entrada
$\mathcal{D}_1(i) \leftarrow 1/M$
for $t \leftarrow 1$ **to** T **do**
  $\chi \leftarrow N(0, 1)$ variable aleatoria con distribución normal
  $\Gamma \leftarrow \gamma + \sigma\chi$
  $p' \leftarrow \frac{n\Gamma}{100-\Gamma}$
  $D_t^N \leftarrow$ `Bootstrap` (n, D^N)
  $D_t^P \leftarrow D^P$
  $S_t \leftarrow$ `MISMOTE` $(p' - p, D^P)$
  $D_t \leftarrow D_t^P \cup D_t^N \cup S_t$
  $\mathcal{D}'_t(i) \leftarrow$recálculo de pesos
  $h_t \leftarrow I(D_t, \mathcal{D}'_t)$
  $\epsilon \leftarrow \sum_{i, y_i \neq h_t(x_i)}^{M} \mathcal{D}_t(i)$
  if $\epsilon > 0{,}5$ **then**
    $T \leftarrow t - 1$
    return
  $\alpha_t \leftarrow \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$
  $\mathcal{D}_{t+1}(i) \leftarrow \mathcal{D}_t(i)\, e^{(-\alpha_t h_t(x_i) y_i)}$
  Normalizar $\mathcal{D}_{t+1}(i)$

Algorithm 8: MISOBoosting

Después de cada iteración se calcula la pseudo-pérdida (línea 16) y se recalculan los pesos de todos los ejemplos de entrenamiento multiplicando los pesos de las instancias que fueron clasificadas correctamente por el factor α_t (línea 21) y luego se normaliza (línea 22).

2.2.5. MISORUBoost

MISORUBoost, acrónimo de *Multi-Instance Synthetic Oversampling Random Undersampling Boosting*, es un ensamble de clasificadores que incorpora una solución para enfrentar el problema de clases desbalanceadas descrito por el Algoritmo 9. Consiste en un sobremuestreo aleatorio con ejemplos sintéticos en la clase positiva (minoritaria) y al mismo tiempo un submuestreo aleatorio en la clase negativa (mayoritaria) en cada etapa del algoritmo boosting.

En los pasos 1 y 2 se hace la segregación de los datos para aplicar la técnica de muestreo sobre los datos divididos por clase. En el paso 6 se le asigna el peso inicial a cada ejemplo del conjunto de entrenamiento.

En la línea 5 se calcula el número de ejemplos de entrenamiento de la clase negativa que se reduce mediante muestreo aleatorio en un por ciento indicado por el parámetro β. El muestreo se hace con remplazo (bootstrapping) en la línea 13. En cada iteración del algoritmo se mantiene el número de ejemplos de la clase positiva (línea 11).

El parámetro γ es el por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase positiva después de que el número de ejemplos de entrenamiento negativos fue reducido por submuestreo. El siguiente ejemplo muestra cómo funciona el algoritmo MISORUBoosting. Sea un conjunto de entrada con 20 ejemplos positivos y 100 negativos, y se escogen los parámetros $\beta = 40\,\%$ y $\gamma = 50\,\%$. Entonces, el número de ejemplos de entrenamiento negativos se reduce al $40\,\%$, con lo cual quedan 40 ejemplos ($n' = 40$) y luego se muestrean 40 ejemplos de la clase positiva ($p' = 40$), de forma que cada clase contiene la mitad de ejemplos del total. Los primeros ejemplos positivos se obtienen manteniendo el número inicial de la clase positiva (línea 11), mientras que el resto (los otros 20 en este ejemplo) se obtienen aplicando MISMOTE (línea 12) a los ejemplos de entrenamiento positivos.

El parámetro σ permite introducir cierta variabilidad en el por ciento de ejemplos positivos p' en cada iteración del algoritmo, de forma que p' es realmente una variable aleatoria con distribución $N(\gamma, \sigma)$ (líneas 8-10). Esto quiere decir que si se escoge $\gamma = 50\,\%$ y $\sigma = 5$, el por ciento de ejemplos positivos p' del total de ejemplos de entrenamiento se distribuirá normalmente alrededor del $50\,\%$ con una desviación estándar del $5\,\%$.

En la línea 14 se unen los datos segregados en el conjunto D_t.

En la línea 15 se hace el recálculo de los pesos para el conjunto D_t. Para cada ejemplo obtenido a partir del conjunto de datos de entrenamiento inicial que está en D_t se le asigna su peso multiplicado por un factor de normalización al vector de pesos $\mathcal{D}'_t$. En el caso de los nuevos ejemplos sintéticos se les asigna un peso con valor $1/K$ donde K es el tamaño del conjunto D_t.

El conjunto D_t y su distribución de pesos $\mathcal{D}'_t$ sirven de entrada para entrenar el clasificador base $h_t$en la línea 16

Este algoritmo combina las técnicas de submuestreo de la clase mayoritaria y sobremuestreo de la minoritaria. Se submuestrea la clase negativa con un tamaño $n^{'}$, reduciendo así su tamaño y se sobremuestrea la clase positiva y se le incorporan los ejemplos sintéticos obtenidos con MISMOTE a los ejemplos positivos originales, logrando así un equilibrio de las distribuciones de las dos

clases.

Input:

- $D = \{(X_1, y_1), \ldots, (X_M, y_M)\}$ conjunto de ejemplos de entrenamiento con etiquetas de clase $y_i \in Y = 1, \ldots, k$
- T número de iteraciones (def: $T = 10$)
- γ por ciento deseado del total de ejemplos de entrenamiento a ser representado por la clase minoritaria (def: $\gamma = 50$)
- σ grado de perturbación de γ (def: $\sigma = 5$)
- β por ciento deseado de reducción del número de ejemplos de entrenamiento de la clase mayoritaria (def: $\beta = 0$)
- I algoritmo de clasificación de aprendizaje débil

Output:

- $H(X) = \arg \max_{y \in Y} \sum_{t=1}^{T} \alpha_t h_t(X, y)$ clasificador ensamblado donde $h_t(X, y) \in [-1, 1]$ son los clasificadores inducidos

$D^P \leftarrow$ conjunto de ejemplos positivos
$D^N \leftarrow$ conjunto de ejemplos negativos
$p \leftarrow \left|D^P\right|$ número de ejemplos positivos de entrada
$n \leftarrow \left|D^N\right|$ número de ejemplos negativos de entrada
$n' \leftarrow \frac{n\beta}{100}$ número de ejemplos negativos de salida
$\mathcal{D}_1(i) \leftarrow 1/M$
for $t \leftarrow 1$ **to** T **do**
  $\chi \leftarrow N(0, 1)$ variable aleatoria con distribución normal
  $\Gamma \leftarrow \gamma + \sigma\chi$
  $p' \leftarrow \frac{n'\Gamma}{100-\Gamma}$
  $D_t^P \leftarrow D^P$
  $S_t \leftarrow$ `MISMOTE` $(p' - p, D^P)$
  $D_t^N \leftarrow$ `Bootstrap` (n', D^N)
  $D_t \leftarrow D_t^P \cup D_t^N \cup S_t$
  $\mathcal{D}'_t(i) \leftarrow$recálculo de pesos
  $h_t \leftarrow I(D_t, \mathcal{D}'_t)$
  $\epsilon \leftarrow \sum_{i, y_i \neq h_t(x_i)}^{M} \mathcal{D}_t(i)$
  if $\epsilon > 0{,}5$ **then**
    $T \leftarrow t - 1$
    return
  $\alpha_t \leftarrow \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$
  $D_{t+1}(i) \leftarrow D_t(i) e^{(-\alpha_t h_t(x_i) y_i)}$
  Normalizar $D_{t+1}(i)$

Algorithm 9: MISORUBoost

Después de cada iteración se calcula la pseudo-pérdida (línea 17) y se recalculan los pesos de todos los ejemplos de entrenamiento multiplicando los pesos de las instancias que fueron clasificadas correctamente por el factor α_t(línea 22) y luego se normaliza (línea 23).

Capítulo 3

Experimentación y resultados

En este capítulo se describen los experimentos realizados para verificar la eficacia de los algoritmos propuestos en el capítulo anterior con los que se pretende dar respuesta a las preguntas de investigación del presente trabajo. El primero de estos experimentos busca descubrir cuál de los algoritmos basados en Bagging es el mejor, el segundo busca la misma meta pero con los algoritmos basados en Boosting; el tercer experimento tiene como objetivo comparar los dos algoritmos resultantes de las dos pruebas anteriores para quedarse con el más eficiente y la cuarta prueba compara los dos algoritmos más eficaces basados uno en Bagging y otro en Boosting con los algoritmos del estado del arte.

3.1. Elementos y métodos

Para presentar los elementos involucrados en el desarrollo de los experimentos, en la sección 3.1.1 se describen los conjuntos de datos utilizados para el estudio, en la 3.1.2 se explica brevemente el funcionamiento del clasificador base usado y en la sección 3.1.3 se explica cómo se evaluó el desempeño de la clasificación y las pruebas estadísticas realizadas para probar la significación de los resultados.

3.1.1. Datos

Los cuatro experimentos a realizar se efectuaron sobre 27 conjuntos de datos multinstancia desbalanceados, cada uno con dos clases. Los conjuntos de datos pertenecen a tres dominios de aplicación distintos: seis conjuntos de datos pertenecientes al problema de clasificación textual WIR (por sus siglas en inglés Web Index page Recommendation), 20 pertenecientes al problema de clasificación de imágenes COREL y uno llamado Thioredoxin relacionado con la predicción de actividad farmacológica. En la tabla 3.1 se muestran algunas de las características más representativas de los conjuntos de datos utilizados. Observe que la taza de desbalance va desde 3,7 hasta 19,0.

Name	attrs	- inst	+ inst	total inst	- bags	+ bags	total bags	IR
Thioredoxin	8	23270	3341	26611	168	25	193	6,7
WIR1	304	2867	556	3423	92	21	113	4,4
WIR2	298	2867	556	3423	92	21	113	4,4
WIR3	303	2867	556	3423	92	21	113	4,4
WIR4	303	757	2666	3423	92	89	113	3,7
WIR5	302	757	2666	3423	92	89	113	3,7
WIR6	304	757	2666	3423	92	89	113	3,7
Corel1	9	7463	484	7947	1900	100	2000	19,0
Corel2	9	7593	354	7947	1900	100	2000	19,0
Corel3	9	7637	310	7947	1900	100	2000	19,0
Corel4	9	7188	759	7947	1900	100	2000	19,0
Corel5	9	7747	200	7947	1900	100	2000	19,0
Corel6	9	7645	302	7947	1900	100	2000	19,0
Corel7	9	7501	446	7947	1900	100	2000	19,0
Corel8	9	7558	389	7947	1900	100	2000	19,0
Corel9	9	7609	338	7947	1900	100	2000	19,0
Corel10	9	7223	724	7947	1900	100	2000	19,0
Corel11	9	7567	380	7947	1900	100	2000	19,0
Corel12	9	7667	280	7947	1900	100	2000	19,0
Corel13	9	7428	519	7947	1900	100	2000	19,0
Corel14	9	7595	352	7947	1900	100	2000	19,0
Corel15	9	7454	493	7947	1900	100	2000	19,0
Corel16	9	7691	256	7947	1900	100	2000	19,0
Corel17	9	7717	230	7947	1900	100	2000	19,0
Corel18	9	7515	432	7947	1900	100	2000	19,0
Corel19	9	7613	334	7947	1900	100	2000	19,0
Corel20	9	7582	365	7947	1900	100	2000	19,0

Cuadro 3.1: Características de los conjuntos de datos usados en los experimentos.

3.1.2. Algoritmos

Para la realización de los experimentos se utilizaron los siguientes algoritmos, todos con número de iteraciones igual a 10 y parámetros como aparecen en la tabla 3.2. Los dos primeros (Bagging y AdaBoost) son los algoritmos originales de referencia, los próximos seis (MIRUBagging, MISOBagging, MISORUBagging, MIRUBoost, MISOBoost y MISORUBoost) son las nuevas propuestas del presente trabajo y los últimos cuatro (Ab1, Ab2, Ab3 y Ab4) son algoritmos sensibles al costo encontrados en el estado del arte.

Clasificador	Parámetros
Bagging	
AdaBoost	
MIRUBagging	$\sigma = 5$, $\gamma = 50$
MISOBagging	$\sigma = 5$, $\gamma = 50$
MISORUBagging	$\sigma = 5$, $\gamma = 50$, $\beta = 100$
MIRUBoost	$\sigma = 5$, $\gamma = 50$
MISOBoost	$\sigma = 5$, $\gamma = 50$
MISORUBoost	$\sigma = 5$, $\gamma = 50$, β=100
Ab1	
Ab2	
Ab3	
Ab4	

Cuadro 3.2: Algoritmos a correr y parámetros.

Como clasificador base se utilizó el MITI (Multi Instance Tree Inducer) [4] para la corrida de todos los algoritmos. MITI es un clasificador multinstancia basado en árbol de toma de decisiones y cómo se va a entrenar en reiteradas ocasiones se tuvo en cuenta que el tiempo que se toma para entrarse es relativamente corto, haciendo posible así múltiples corridas.

3.1.3. Validación de los resultados

Para evaluar los resultados y garantizar que son independientes de la partición entre datos de entrenamiento y prueba se utilizó la técnica de validación cruzada de 5 particiones (5-fold cross-validation). La validación cruzada particiona el conjunto de datos original en conjuntos complementarios, hace al análisis en el conjunto de entrenamiento y lo valida en el conjunto de prueba. En el presente trabajo se tomarán en cuenta para los experimentos las medidas AUC y Gmean, por lo explicado anteriormente en la sección 1.2.2.

Para determinar diferencias significativos entre los algoritmos usados en los experimentos se utilizaron el test de Wilcoxon [51], Friedman [15], Iman-Davenport [22]y el test post-hoc de Holm [20] como pruebas estadísticas. Los test anteriores pertenecen al grupo de pruebas no paramétricas debido a que no se puede afirmar que los datos de los experimentos siguen la distribución normal.

El test de Friedman es el equivalente no paramétrico del test ANOVA. Friedman hace un ranking de los algoritmos separadamente para cada conjunto de datos, el que mejor desempeño tenga ocupa el primer lugar, el segundo mejor el segundo..... En caso de empate los ranking son asignados por promedio.

El test de Wilcoxon se aplica con el mismo objetivo que el de Friedman, pero en caso de que sean 2 muestras relacionadas, mientras que Friedman se puede usar en muestras relacionadas de tamaño N.

El valor p calculado por el test de Wilcoxon y Friedman representa la pro-

babilidad de obtener un resultado al menos tan extremo como el obtenido en el experimento (valor del estadístico calculado), suponiendo que los clasificadores tienen desempeños similares (hipótesis nula). Un valor p muy pequeño (por ejemplo, menor que 0,05) sugiere que la hipótesis de partida es falsa, i.e., realmente existen diferencias significativas entre los métodos comparados.

Iman-Davenport mostró que el coeficiente de Friedman es indeseablemente conservativo y derivó un estadístico mejor.

Una vez que se rechace la hipótesis nula, o sea, se demuestre que hay diferencias significativas entre los algoritmos a comparar, se puede aplicar el test post-hoc de Holm para detectar diferencias significativas par a par entre todos los clasificadores.

3.2. Comparación entre las algoritmos Baggings

El primer experimento consiste en comparar los algoritmos desarrollados a partir de Bagging con el algoritmo original. A continuación se analizarán los valores por algoritmo de clasificación y por conjunto de datos de las medidas AUC y GMean.

En la tabla 3.3 se muestran los valores del AUC resultantes de la corrida de los algoritmos de la familia Bagging, así como los promedios generales por cada clasificador. En la primera columna se encuentran los conjuntos de datos utilizados, en la segunda el algoritmo de referencia Bagging y a continuación las tres variantes desarrolladas en esta tesis: MIRUBagging, MISOBagging y MISORUBagging. Los valores en negritas muestran donde los nuevos algoritmos presentan desventajas en cuanto al algoritmo original y la flecha $\uparrow$ significa mejora y la flecha $\downarrow$ significa desventaja.

Conjunto de datos	Bagging	MIRUBagging	MISOBagging	MISORUBagging
Thioredoxin	0.7701	**0.5536**↓	**0.5119**↓	**0.5119**↓
WIR1	0.8639	**0.8049**↓	**0.8385**↓	0.8929↑
WIR2	0.8028	0.8041↑	0.8799↑	0.8942↑
WIR3	0.8678	**0.8605**↓	**0.8499**↓	0.8789↑
WIR4	0.7364	0.8003↑	0.7584↑	0.8240↑
WIR5	0.7022	0.7446↑	0.8502↑	0.8713↑
WIR6	0.6791	0.7025↑	0.8455↑	0.8425↑
Corel20-1	0.7709	0.9052↑	0.9187↑	0.9275↑
Corel20-2	0.6825	0.8558↑	0.9077↑	0.9340↑
Corel20-3	0.7720	0.8926↑	0.9333↑	0.9362↑
Corel20-4	0.9326	0.9718↑	0.9681↑	0.9763↑
Corel20-5	0.9886	0.9979↑	0.9920↑	0.9938↑
Corel20-6	0.7813	0.8967↑	0.9287↑	0.9388↑
Corel20-7	0.9410	0.9783↑	0.9750↑	0.9806↑
Corel20-8	0.9750	0.9889↑	0.9726↑	0.9750↑
Corel20-9	0.7091	0.8647↑	0.9003↑	0.9053↑
Corel20-10	0.9044	0.9452↑	0.9494↑	0.9649↑
Corel20-11	0.7773	0.8694↑	0.8987↑	0.9046↑
Corel20-12	0.6941	0.8261↑	0.9002↑	0.9171↑
Corel20-13	0.8378	0.9308↑	0.9413↑	0.9464↑
Corel20-14	0.8643	0.9437↑	0.9623↑	0.9626↑
Corel20-15	0.8241	0.9031↑	0.9298↑	0.9403↑
Corel20-16	0.7412	0.8968↑	0.9365↑	0.9503↑
Corel20-17	0.8811	0.9509↑	0.9404↑	0.9560↑
Corel20-18	0.8074	0.9130↑	0.9459↑	0.9493↑
Corel20-19	0.8003	0.9067↑	0.9255↑	0.9326↑
Corel20-20	0.6937	0.6975↑	0.8110↑	0.8485↑
Promedios Generales	0.8074	0.8669	0.8953	0.9095

Cuadro 3.3: Tabla de los valores del AUC para los algoritmos de la familia Bagging.

Para los algoritmos MIRUBagging, MISOBagging y MISORUBagging se obtienen mejores medidas en todos los conjuntos de datos, excepto en los casos del WIR1 y WIR3 para MIRUBagging y MISOBagging, así como el caso del Thioredoxin para los 3. Cuando se analizan los promedios generales se puede observar ventajas, ordenándolos en el siguiente orden de forma ascendente: Bagging, MIRUBagging, MISOBagging y MISORUBagging.

A continuación en la tabla 3.4 se presentan los resultados de aplicar el test de Friedman para verificar que existen diferencias en al menos dos de los clasificadores de la familia Bagging con respecto a la medida AUC.

Como se puede observar en la tabla 3.4 el test de Friedman que el mejor resultado es para MISORUBagging mientras que el peor resultado lo obtiene Bagging. El coeficiente obtenido por Iman-Davenport tiene un valor p de signi-

Algorithm	Ranking
Bagging	3.6852
MIRUBagging	2.7037
MISOBagging	2.3889
MISORUBagging	1.2222

Cuadro 3.4: Rankings de Friedman de los algoritmos basados en Bagging para la medida AUC

ficación igual a 5.551115123125783E-16, el cual es menor que 0.05 por lo cual se rechaza la hipótesis nula y se afirma que existen diferencias significativas entre al menos 2 de los clasificadores.

Para descubrir entre cuales de ellos existe diferencia significativa se aplicó el test de Holm, cuyos resultados se muestran en la tabla 3.5.

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
6	**Bagging vs. MISORUBagging**	**7.009715**	**0**	**0.008333**
5	**MIRUBagging vs. MISORUBagging**	**4.21637**	**0.000025**	**0.01**
4	**Bagging vs. MISOBagging**	**3.689324**	**0.000225**	**0.0125**
3	**MISOBagging vs. MISORUBagging**	**3.320392**	**0.000899**	**0.016667**
2	**Bagging vs. MIRUBagging**	**2.793345**	**0.005217**	**0.025**
1	MIRUBagging vs. MISOBagging	0.895979	0.370264	0.05

Cuadro 3.5: P-values Table for $\alpha = 0.05$

Los resultados señalados en negritas muestran la existencia de diferencias significativas en las filas con i=2,3,...,6. Como se puede observar, todos los algoritmos nuevos desarrollados presentan diferencias en cuanto al Bagging original. También se puede concluir que MISORUBagging es significativamente mejor que las otras dos propuestas.

A modo de resumen, los clasificadores ensamblados MIRUBagging, MISOBagging y MISORUBagging presentan mejoras significativas con respecto al algoritmo Bagging original tomando en cuenta la medida AUC, siendo el MISORUBagging el de mejores resultados.

En la tabla 3.6 se muestran los valores del GMean resultantes de la corrida de los algoritmos de la familia Bagging, así como los promedios generales por cada clasificador. En la primera columna se encuentran los conjuntos de datos utilizados, en la segunda el algoritmo de referencia Bagging y a continuación las tres variantes desarrolladas en esta tesis: MIRUBagging, MISOBagging y MISORUBagging.

Conjunto de datos	Bagging	MIRUBagging	MISOBagging	MISORUBagging
Thioredoxin	0.5398	**0.1091**↓	**0.0000**↓	**0.0000**↓
WIR1	0.7960	**0.6342**↓	**0.7782**↓	0.8056↑
WIR2	0.7445	**0.6594**↓	0.8117↑	0.8452↑
WIR3	0.7809	**0.6994**↓	0.8140↑	0.8470↑
WIR4	0.5708	0.5842↑	0.6787↑	0.6616↑
WIR5	0.5741	0.6234↑	0.7331↑	0.7331↑
WIR6	0.4539	0.6577↑	0.7837↑	0.6910↑
Corel20-1	0.2826	0.8020↑	0.8405↑	0.8575↑
Corel20-2	0.3291	0.7697↑	0.8091↑	0.8314↑
Corel20-3	0.4235	0.8136↑	0.8573↑	0.8772↑
Corel20-4	0.7902	0.8908↑	0.9133↑	0.9211↑
Corel20-5	0.9517	0.9887↑	0.9657↑	0.9620↑
Corel20-6	0.3993	0.8300↑	0.8404↑	0.8481↑
Corel20-7	0.7171	0.9219↑	0.9398↑	0.9392↑
Corel20-8	0.8037	0.9554↑	0.9240↑	0.9156↑
Corel20-9	0.1411	0.8032↑	0.8236↑	0.8207↑
Corel20-10	0.6532	0.8472↑	0.8929↑	0.8966↑
Corel20-11	0.4234	0.8054↑	0.8254↑	0.8253↑
Corel20-12	0.2821	0.7584↑	0.8062↑	0.8158↑
Corel20-13	0.5814	0.8405↑	0.8645↑	0.8829↑
Corel20-14	0.6441	0.8821↑	0.9100↑	0.9087↑
Corel20-15	0.5174	0.8210↑	0.8566↑	0.8717↑
Corel20-16	0.2446	0.7984↑	0.8628↑	0.8769↑
Corel20-17	0.5549	0.8962↑	0.8600↑	0.8791↑
Corel20-18	0.5269	0.8305↑	0.8787↑	0.8772↑
Corel20-19	0.4663	0.8412↑	0.8441↑	0.8413↑
Corel20-20	0.2814	0.6521↑	0.7075↑	0.7513↑
Promedios Generales	0.5361	0.7672↑	0.8082↑	0.8142↑

Cuadro 3.6: Tabla de los valores del GMean para los algoritmos de la familia Bagging.

Para los algoritmos MIRUBagging, MISOBagging y MISORUBagging se obtienen mejores medidas en todos los conjuntos de datos, excepto en los casos del WIR1 para MIRUBagging y MISOBagging, WIR2 y WIR3 para el MIRUBagging y Thioredoxin para los 3. Cuando se analizan los promedios generales se puede observar que el de mejor resultado es el algoritmo MISORUBagging y el de peores resultados es el Bagging.

En la tabla 3.7 se presentan los resultados de aplicar el test de Friedman para verificar que existen diferencias en al menos dos de los clasificadores de la familia Bagging con respecto a la medida GMean.

Como se puede observar en la tabla 3.7 el test de Friedman muestra que el mejor resultado es para MISORUBagging seguido de cerca por MISOBagging mientras que el peor resultado lo obtiene Bagging. Este resultado es concor-

Algorithm	Ranking
Bagging	3.7407
MIRUBagging	2.8519
MISOBagging	1.8148
MISORUBagging	1.5926

Cuadro 3.7: Rankings de Friedman de los algoritmos basados en Bagging para la medida GMean

dante con lo observado para la medida AUC. El coeficiente obtenido por Iman-Davenport tiene un valor p de significación igual a 3.885780586188048E-15, el cual es menor que 0.05 por lo cual se puede rechazar la hipótesis nula y afirmar que existen diferencias significativas entre al menos 2 de los clasificadores.

Para descubrir entre cuales de ellos existe diferencia significativa se aplicó el test de Holm, cuyos resultados se muestran en la tabla 3.8.

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
6	**Bagging vs. MISORUBagging**	**6.113737**	**0**	**0.008333**
5	**Bagging vs. MISOBagging**	**5.481281**	**0**	**0.01**
4	**MIRUBagging vs. MISORUBagging**	**3.583915**	**0.000338**	**0.0125**
3	**MIRUBagging vs. MISOBagging**	**2.951459**	**0.003163**	**0.016667**
2	**Bagging vs. MIRUBagging**	**2.529822**	**0.011412**	**0.025**
1	MISOBagging vs. MISORUBagging	0.632456	0.527089	0.05

Cuadro 3.8: P-values Table for $\alpha = 0.05$

Los resultados señalados en negritas muestran la existencia de diferencias significativas en las filas con i=2,3,...,6. Como se puede observar el resultado obtenido es similar al obtenido cuando se analizó la medida AUC. Los clasificadores desarrollados (MIRUBagging, MISOBagging y MISORUBagging) son significativamente mejores que el Bagging original y al mismo tiempo el MISORUBagging es mejor significativamente que las otras propuestas.

3.3. Comparación entre las algoritmos Boostings

El segundo experimento consiste en comparar los algoritmos desarrollados a partir de Boosting con el algoritmo AdaBoost original. A continuación se analizarán los valores por algoritmo de clasificación y por conjunto de datos de las medidas AUC y GMean.

En la tabla 3.9 se muestran los valores del AUC resultantes de la corrida de los algoritmos de la familia Boosting, así como los promedios generales por cada clasificador. En la primera columna se encuentran los conjuntos de datos utilizados, en la segunda el algoritmo de referencia AdaBoost y a continuación las tres variantes desarrolladas en esta tesis: MIRUBoost, MISOBoost y MISORUBoost.

Conjunto de datos	AdaBoost	MIRUBoost	MISOBoost	MISORUBoost
Thioredoxin	0.7463	**0.4821**↓	**0.5024**↓	**0.5286**↓
WIR1	0.8261	**0.7446**↓	**0.7782**↓	**0.8007**↓
WIR2	0.8175	**0.6643**↓	0.8227↑	**0.8183**↓
WIR3	0.8447	**0.7306**↓	**0.8142**↓	0.8678↑
WIR4	0.8125	**0.7877**↓	**0.7657**↓	**0.8015**↓
WIR5	0.7905	0.8336↑	**0.7823**↓	**0.7860**↓
WIR6	0.8069	**0.7292**↓	**0.7460**↓	**0.7374**↓
Corel20-1	0.8346	0.8918↑	0.8864↑	0.8978↑
Corel20-2	0.7590	0.7766↑	0.8457↑	0.8550↑
Corel20-3	0.7968	0.8362↑	0.8765↑	0.8935↑
Corel20-4	0.9635	0.9645↑	0.9652↑	0.9787↑
Corel20-5	0.9962	0.9903↑	0.9922↑	0.9948↑
Corel20-6	0.8043	0.8533↑	0.8455↑	0.8752↑
Corel20-7	0.9742	0.9638↑	0.9567↑	0.9774↑
Corel20-8	0.9680	0.9799↑	0.9665↑	0.9801↑
Corel20-9	0.7109	0.7656↑	0.8160↑	0.8576↑
Corel20-10	0.9409	0.9147↑	0.9439↑	0.9517↑
Corel20-11	0.7795	0.8365↑	0.8507↑	0.8358↑
Corel20-12	0.7404	0.7861↑	0.8047↑	0.8310↑
Corel20-13	0.8889	0.8491↑	0.9247↑	0.9369↑
Corel20-14	0.8968	0.9336↑	0.9319↑	0.9180↑
Corel20-15	0.8623	0.8402↑	0.8772↑	0.9247↑
Corel20-16	0.7631	0.8720↑	0.8936↑	0.8858↑
Corel20-17	0.9023	0.9213↑	0.9152↑	0.9332↑
Corel20-18	0.8656	0.8554↑	0.9109↑	0.9052↑
Corel20-19	0.8850	0.8628↑	0.8919↑	0.8993↑
Corel20-20	0.7007	0.6445↑	0.7138↑	0.7411↑
Promedios Generales	0.8399	0.8263↓	0.8526↓	0.8671↑

Cuadro 3.9: Tabla de los valores del AUC para los algoritmos de la familia Boosting.

Como se observar en la tabla para los conjuntos Thioredoxin y los WIR el algoritmo AdaBoost tradicional obtiene mejores resultados en la mayoría de los casos que las nuevas propuestas de algoritmos basadas en Boost. Sin embargo, en todos los casos de los Corel los nuevos clasificadores mejoran el algoritmo AdaBoost original. Cuando se analizan los promedios generales, solo el clasificador MISORUBoost presenta mejores valores que el AdaBoost clásico.

A continuación en la tabla 3.10 se presentan los resultados de aplicar el test de Friedman para verificar que existen diferencias en al menos dos de los clasificadores de la familia Boosting con respecto a la medida AUC.

El ranking que elabora Friedman sitúa al algoritmo MISORUBoost como el mejor, obteniendo el peor resultado el algoritmo MIRUBoost. El coeficiente obtenido por Iman-Davenport tiene un valor p de significación igual a 5.888611070758998E-

Algorithm	Ranking
AdaBoost	2.9259
MIRUBoost	3.1111
MISOBoost	2.4074
MISORUBoost	1.5556

Cuadro 3.10: Rankings de Friedman de los algoritmos basados en Boosting para la medida AUC

6, el cual es menor que 0.05 por lo cual se puede rechazar la hipótesis nula y afirmar que existen diferencias significativas entre al menos 2 de los clasificadores.

Para descubrir entre cuales de ellos existe diferencia significativa se aplicó el test de Holm, cuyos resultados se muestran en la tabla 3.11.

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
6	**MIRUBoost vs. MISORUBoost**	**4.427189**	**0.00001**	**0.008333**
5	**AdaBoost vs. MISORUBoost**	**3.900142**	**0.000096**	**0.01**
4	MISOBoost vs. MISORUBoost	2.424413	0.015333	0.0125
3	MIRUBoost vs. MISOBoost	2.002776	0.045201	0.016667
2	AdaBoost vs. MISOBoost	1.47573	0.140017	0.025
1	AdaBoost vs. MIRUBoost	0.527046	0.598161	0.05

Cuadro 3.11: P-values Table for $\alpha = 0.05$

Los resultados señalados en negritas muestran la existencia de diferencias significativas en las filas con i=5,6. Como se puede observar, solo el algoritmo MISORUBoost presenta diferencias significativas en cuanto al AdaBoost original. También existen diferencias significativas entre el MISORUBoost y MIRUBoost.

A modo de resumen, solo el clasificador MISORUBoost presenta mejoras significativas con respecto al algoritmo AdaBoost original tomando en cuenta la medida AUC. Para los problemas Thioredoxin y WIR las nuevas propuestas no mejoran los resultados del AdaBoost original.

En la tabla 3.12 se muestran los valores del GMean resultantes de la corrida de los algoritmos de la familia Boosting, así como los promedios generales por cada clasificador. En la primera columna se encuentran los conjuntos de datos utilizados, en la segunda el algoritmo de referencia AdaBoost y a continuación las tres variantes desarrolladas en esta tesis: MIRUBoost, MISOBoost y MISORUBoost.

Conjunto de datos	AdaBoost	MIRUBoost	MISOBoost	MISORUBoost
Thioredoxin	0.6701	**0.0000**↓	**0.1336**↓	**0.0772**↓
WIR1	0.7337	**0.6483**↓	**0.7144**↓	**0.6916**↓
WIR2	0.7792	**0.6633**↓	**0.7477**↓	**0.7678**↓
WIR3	0.7539	**0.7082**↓	0.7682↑	0.8203↑
WIR4	0.5708	0.6620↑	0.6704↑	0.6746↑
WIR5	0.5949	0.6704↑	0.6978↑	0.6787↑
WIR6	0.5708	0.6538↑	0.5914↑	0.6498↑
Corel20-1	0.4446	0.8100↑	0.8010↑	0.8070↑
Corel20-2	0.3973	0.7101↑	0.7481↑	0.7773↑
Corel20-3	0.4968	0.7730↑	0.7850↑	0.8063↑
Corel20-4	0.8098	0.8989↑	0.8943↑	0.9239↑
Corel20-5	0.9721	0.9860↑	0.9403↑	0.9644↑
Corel20-6	0.3726	0.7543↑	0.7670↑	0.7824↑
Corel20-7	0.7584	0.9231↑	0.9073↑	0.9206↑
Corel20-8	0.8020	0.9357↑	0.8926↑	0.9300↑
Corel20-9	0.2976	0.6886↑	0.7347↑	0.7746↑
Corel20-10	0.6890	0.8476↑	0.8726↑	0.8824↑
Corel20-11	0.4456	0.7602↑	0.7901↑	0.7474↑
Corel20-12	0.3576	0.7310↑	0.7301↑	0.7421↑
Corel20-13	0.6052	0.7700↑	0.8480↑	0.8535↑
Corel20-14	0.6810	0.8638↑	0.8735↑	0.8531↑
Corel20-15	0.5063	0.7882↑	0.7994↑	0.8240↑
Corel20-16	0.3455	0.7973↑	0.8218↑	0.8073↑
Corel20-17	0.6381	0.8407↑	0.8352↑	0.8457↑
Corel20-18	0.4659	0.7724↑	0.8407↑	0.8175↑
Corel20-19	0.5364	0.8039↑	0.8031↑	0.8089↑
Corel20-20	0.3714	0.6236↑	0.6453↑	0.6991↑
Promedios Generales	0.5802	0.7439↑	0.7649↑	0.7751↑

Cuadro 3.12: Tabla de los valores del GMean para los algoritmos de la familia Boosting.

Como se puede observar para el conjunto Thioredoxin ninguno de los nuevos algoritmos logra mejorar el desempeño de la clasificación, para los conjuntos de datos WIR, en los casos del WIR1 y el WIR2 los nuevos clasificadores disminuyen la eficacia del proceso de clasificación y en el caso del WIR3, solo el MIRUBoosting empeora los resultados.En todos los casos de los Corel los nuevos clasificadores mejoran el algoritmo AdaBoost original. Cuando se analizan los promedios generales, todos los nuevos algoritmos presentan mejores valores que el AdaBoost.

A continuación en la tabla 3.13 se presentan los resultados de aplicar el test de Friedman para verificar que existen diferencias en al menos dos de los clasificadores de la familia Boosting con respecto a la medida GMean.

El ranking que elabora Friedman sitúa al algoritmo MISORUBoost como

Algorithm	Ranking
AdaBoost	3.5556
MIRUBoost	2.5556
MISOBoost	2.2222
MISORUBoost	1.6667

Cuadro 3.13: Rankings de Friedman de los algoritmos basados en Boosting para la medida GMean

el mejor. Luego le siguen por su orden el algoritmo MISOBoost, MIRUBoost y AdaBoost. El coeficiente obtenido por Iman-Davenport tiene un valor p de significación igual a 4.104926043524415E-8 el cual es menor que 0.05 por lo cual se puede rechazar la hipótesis nula y afirmar que existen diferencias significativas entre al menos 2 de los clasificadores.

Para descubrir entre cuales de ellos existe diferencia significativa se aplicó el test de Holm, cuyos resultados se muestran en la tabla 3.14.

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
6	**AdaBoost vs. MISORUBoost**	**5.375872**	**0**	**0.008333**
5	**AdaBoost vs. MISOBoost**	**3.794733**	**0.000148**	**0.01**
4	**AdaBoost vs. MIRUBoost**	**2.84605**	**0.004427**	**0.0125**
3	**MIRUBoost vs. MISORUBoost**	**2.529822**	**0.011412**	**0.016667**
2	MISOBoost vs. MISORUBoost	1.581139	0.113846	0.025
1	MIRUBoost vs. MISOBoost	0.948683	0.342782	0.05

Cuadro 3.14: P-values Table for $\alpha = 0.05$

Los resultados señalados en negritas muestran la existencia de diferencias significativas en las filas con i=3,...,6. Como se puede observar, todos los algoritmos desarrollados en el presente trabajo presentan diferencias significativas en cuanto al AdaBoost original. También existen diferencias significativas entre el MISORUBoost y MIRUBoost.

A modo de resumen, los clasificadores MIRUBoost, MISOBoost y MISORUBoost presentan mejoras significativas con respecto al algoritmo AdaBoost original tomando en cuenta la medida GMean, siendo el MISORUBoost el de mejores resultados.

3.4. Mejor propuesta global

El tercer experimento consiste en comparar los algoritmos MISORUBagging y MISORUBoost debido a ser los clasificadores con mejores resultados obtenidos en los experimentos anteriores. A continuación se analizarán los valores por algoritmo de clasificación y por conjunto de datos de las medidas AUC y GMean.

Los valores del AUC resultantes de la corrida de los algoritmos MISORUBagging y MISORUBoost, pueden encontrarse en las tablas 3.3 y 3.9 respectivamente. MISORUBagging obtiene mejores resultados que MISORUBoost en

todos los conjuntos de datos excepto cuatro, alcanzando también mejor promedio general.

Los valores del GMean resultantes de la corrida de los algoritmos MISORUBagging y MISORUBoost, pueden encontrarse en las tablas 3.6 y 3.12 respectivamente. MISORUBagging obtiene mejores resultados que MISORUBoost en todos los conjuntos de datos excepto cinco, alcanzando también mejor promedio general.

A continuación en la tabla 3.15 se presentan los resultados de aplicar el test de Wilcoxon (1x1) para verificar que existen diferencias significativas entre los clasificadores MISORUBagging y MISORUBoost con respecto a las medidas AUC y GMean.

Medida	R+	R-	p
AUC	16	362	0.000
GMean	35	343	0.000

Cuadro 3.15: Prueba de Wilcoxon para determinar diferencias significativas entre MISORUBoost vs MISORUBagging para AUC y GMean

Como se puede apreciar en la tabla 3.15 la suma de los rangos favorece al algoritmo MISORUBagging en ambas medidas de desempeño y el valor de p implica que es significativamente mejor en los dos casos.

3.5. Comparación con algoritmos del estado del arte

El cuarto y último experimento consiste en comparar los algoritmos desarrollados MISORUBagging y MISORUBoosting con los algoritmos del estado del arte sensibles al costo: Ab1, Ab2, Ab3 y Ab4. A continuación se analizarán los valores por algoritmo de clasificación y por conjunto de datos de las medidas AUC y GMean.

En la tabla 3.16 se muestran los valores del AUC resultantes de la corrida de los algoritmos MISORUBagging, MISORUBoosting, Ab1, Ab2, Ab3 y Ab4, así como los promedios generales por cada clasificador. En la primera columna se encuentran los conjuntos de datos utilizados, en las 4 columnas que le siguen se encuentran las variantes de los algoritmos sensibles al costo Ab1, Ab2, Ab3 y Ab4 y a continuación los dos mejores algoritmos desarrollados en el presente trabajo: MISORUBagging y MISORUBoost.

Conjunto de datos	Ab1	Ab2	Ab3	Ab4	MISORUBagging	MISORUBoosting
Thioredoxin	0.6571	0.5000	0.6437	0.6304	**0.5119**↓	**0.5286**↓
WIR1	0.7857	0.5054	0.7635	0.7459	0.8929↑	0.8007↑
WIR2	0.8196	0.6674	0.8043	0.7384	0.8942↑	**0.8183**↓
WIR3	0.8709	0.7495	0.7979	0.7562	0.8789↑	**0.8678**↓
WIR4	0.6585	0.5625	0.7743	0.7252	0.8240↑	0.8015↑
WIR5	0.6695	0.5185	0.7278	0.7125	0.8713↑	0.7860↑
WIR6	0.8277	0.7800	0.7144	0.6788	0.8425↑	**0.7374**↓
Corel20-1	0.7512	0.6931	0.8746	0.7960	0.9275↑	0.8978↑
Corel20-2	0.7703	0.5197	0.7490	0.7131	0.9340↑	0.8550↑
Corel20-3	0.7131	0.7897	0.8507	0.7621	0.9362↑	0.8935↑
Corel20-4	0.9100	0.9283	0.9744	0.9203	0.9763↑	0.9787↑
Corel20-5	0.9910	0.9282	0.9953	0.9940	**0.9938**↓	0.9948↑
Corel20-6	0.7236	0.7688	0.8614	0.7881	0.9388↑	0.8752↑
Corel20-7	0.9436	0.9542	0.9723	0.9224	0.9806↑	0.9774↑
Corel20-8	0.9591	0.9612	0.9888	0.9638	**0.9750**↓	**0.9801**↓
Corel20-9	0.7059	0.7156	0.8266	0.7653	0.9053↑	0.8576↑
Corel20-10	0.8961	0.8603	0.9405	0.8857	0.9649↑	0.9517↑
Corel20-11	0.7124	0.7514	0.8486	0.7639	0.9046↑	**0.8358**↓
Corel20-12	0.6740	0.7065	0.7864	0.7248	0.9171↑	0.8310↑
Corel20-13	0.8113	0.8403	0.9048	0.8311	0.9464↑	0.9369↑
Corel20-14	0.8517	0.8428	0.9308	0.8560	0.9626↑	**0.9180**↓
Corel20-15	0.8132	0.8673	0.8885	0.8104	0.9403↑	0.9247↑
Corel20-16	0.7188	0.6974	0.8504	0.7443	0.9503↑	0.8858↑
Corel20-17	0.8596	0.8699	0.9235	0.8281	0.9560↑	0.9332↑
Corel20-18	0.8022	0.8033	0.9000	0.8316	0.9493↑	0.9052↑
Corel20-19	0.7844	0.7992	0.8835	0.8070	0.9326↑	0.8993↑
Corel20-20	0.6872	0.7014	0.7209	0.6335	0.8485↑	0.7411↑
Promedios Generales	0.7914	0.7512	0.8480	0.7900	0.9095↑	0.8671↑

Cuadro 3.16: Tabla de los valores del AUC para los algoritmos de la familia Ab1, Ab2, Ab3, Ab4, MISORUBagging y MISORUBoost.

Como se puede observar los algoritmos MISORUBagging y MISORUBoost presentan mejores resultados que los algoritmos sensibles al costo para la mayoría de los conjuntos de datos. Particularmente en el caso del Thioredoxin, ninguno de los dos mejora a los algoritmos del estado del estado del arte. Cuando se analizan los promedios generales, los dos algoritmos desarrollados (MISORUBagging y MISORUBoost) presentan mejores valores que los algoritmos sensibles al costo.

A continuación en la tabla 3.17 se presentan los resultados de aplicar el test de Friedman para verificar que existen diferencias en al menos dos de los clasificadores de los sensibles al costo y los MISORUBagging y MISORUBoost con respecto a la medida AUC.

El ranking que elabora Friedman sitúa al algoritmo MISORUBagging como

Algorithm	Ranking
Ab1	4.7037
Ab2	5.1111
Ab3	2.963
Ab4	4.5926
MISORUBagging	1.3704
MISORUBoost	2.2593

Cuadro 3.17: Rankings de Friedman de los algoritmos sensibles al costo, MISORUBagging y MISORUBoost y para la medida AUC

el mejor. Luego le siguen por su orden el algoritmo MISORUBoost, Ab3, Ab4, Ab1 y Ab2. El coeficiente obtenido por Iman-Davenport tiene un valor p de significación igual a -2.220446049250313E-16, el cual es menor que 0.05 por lo cual se puede rechazar la hipótesis nula y afirmar que existen diferencias significativas entre al menos 2 de los clasificadores.

Para descubrir entre cuales de ellos existe diferencia significativa se aplicó el test de Holm, cuyos resultados se muestran en la tabla 3.18.

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
15	**Ab2 vs. MISORUBagging**	**7.346669**	**0**	**0.003333**
14	**Ab1 vs. MISORUBagging**	**6.546537**	**0**	**0.003571**
13	**Ab4 vs. MISORUBagging**	**6.328319**	**0**	**0.003846**
12	**Ab2 vs. MISORUBoost**	**5.600926**	**0**	**0.004167**
11	**Ab1 vs. MISORUBoost**	**4.800794**	**0.000002**	**0.004545**
10	**Ab4 vs. MISORUBoost**	**4.582576**	**0.000005**	**0.005**
9	**Ab2 vs. Ab3**	**4.218879**	**0.000025**	**0.005556**
8	**Ab1 vs. Ab3**	**3.418747**	**0.000629**	**0.00625**
7	**Ab3 vs. Ab4**	**3.200529**	**0.001372**	**0.007143**
6	**Ab3 vs. MISORUBagging**	**3.12779**	**0.001761**	**0.008333**
5	MISORUBagging vs. MISORUBoost	1.745743	0.080856	0.01
4	Ab3 vs. MISORUBoost	1.382047	0.166957	0.0125
3	Ab2 vs. Ab4	1.01835	0.308512	0.016667
2	Ab1 vs. Ab2	0.800132	0.423634	0.025
1	Ab1 vs. Ab4	0.218218	0.827259	0.05

Cuadro 3.18: P-values Table for $\alpha = 0.05$

Los resultados señalados en negritas muestran la existencia de diferencias significativas en las filas con i=6,...,15. Como se puede observar, el algoritmo MISORUBagging presenta diferencias con todos los algoritmos del estado del arte y el algoritmo MISORUBoost presenta diferencias significativas con todos excepto el Ab3.

A modo de resumen, el clasificador MISORUBagging presenta mejoras significativas con respecto a todos los algoritmos sensibles al costo del estado del arte, tomando en cuenta la medida AUC. Por otro lado el algoritmo MISORU-

Boost presenta mejoras significativas con respecto a los algoritmos Ab1, Ab2 y Ab4 tomando en cuenta la medida AUC.

En la tabla 3.19 se muestran los valores del AUC resultantes de la corrida de los algoritmos MISORUBagging, MISORUBoost, Ab1, Ab2, Ab3 y Ab4, así como los promedios generales por cada clasificador.

Conjunto de datos/Clasificador	Ab1	Ab2	Ab3	Ab4	MISORUBagging	MISORUBoosting
Thioredoxin	0.2673	0.0000	0.0000	0.0000	**0.0000**↓	0.0772↑
WIR1	0.6594	0.0000	0.1043	0.1043	0.8056↑	0.6916↑
WIR2	0.7323	0.3018	0.5898	0.0000	0.8452↑	0.7678↑
WIR3	0.8015	0.6711	0.5517	0.0000	0.8470↑	0.8203↑
WIR4	0.3536	0.2041	0.7595	0.0000	**0.6616**↓	**0.6746**↓
WIR5	0.4082	0.2041	0.6665	0.0000	0.7331↑	0.6787↑
WIR6	0.6256	0.0000	0.2998	0.2998	0.6910↑	0.6498↑
Corel20-1	0.1995	0.2819	0.5358	0.0000	0.8575↑	0.8070↑
Corel20-2	0.4527	0.0000	0.2115	0.0000	0.8314↑	0.7773↑
Corel20-3	0.4351	0.4450	0.5315	0.0000	0.8772↑	0.8063↑
Corel20-4	0.7715	0.8066	0.6951	0.0000	0.9211↑	0.9239↑
Corel20-5	0.9729	0.2645	0.9781	0.0000	0.9620↑	0.9644↑
Corel20-6	0.3156	0.4441	0.5026	0.0000	0.8481↑	0.7824↑
Corel20-7	0.7242	0.7365	0.7222	0.0000	0.9392↑	0.9206↑
Corel20-8	0.7830	0.7893	0.7887	0.0000	0.9156↑	0.9300↑
Corel20-9	0.2443	0.2427	0.5237	0.0000	0.8207↑	0.7746↑
Corel20-10	0.6445	0.6648	0.5620	0.0000	0.8966↑	0.8824↑
Corel20-11	0.2822	0.3981	0.4329	0.0000	0.8253↑	0.7474↑
Corel20-12	0.2826	0.3842	0.4542	0.0000	0.8158↑	0.7421↑
Corel20-13	0.5806	0.5888	0.5174	0.0000	0.8829↑	0.8535↑
Corel20-14	0.6049	0.6352	0.6525	0.0000	0.9087↑	0.8531↑
Corel20-15	0.4872	0.5262	0.5144	0.0000	0.8717↑	0.8240↑
Corel20-16	0.1729	0.2444	0.5084	0.0000	0.8769↑	0.8073↑
Corel20-17	0.5275	0.6227	0.5891	0.0000	0.8791↑	0.8457↑
Corel20-18	0.4768	0.5145	0.5722	0.0000	0.8772↑	0.8175↑
Corel20-19	0.4449	0.4765	0.5448	0.0000	0.8413↑	0.8089↑
Corel20-20	0.3587	0.3716	0.2542	0.0000	0.7513↑	0.6991↑
Promedios Generales	0.5040	0.4007	0.5208	0.0150	0.8142↑	0.7751↑

Cuadro 3.19: Tabla de los valores del GMean para los algoritmos de la familia Ab1, Ab2, Ab3, Ab4, MISORUBagging y MISORUBoosting.

Como se puede observar los algoritmos MISORUBagging y MISORUBoost presentan mejores resultados que los algoritmos sensibles al costo para la mayoría de los conjuntos de datos. Particularmente en el caso del WIR4, ninguno de los dos mejora a los algoritmos del estado del estado del arte. Cuando se analizan los promedios generales, los dos algoritmos desarrollados (MISORUBagging y MISORUBoost) presentan mejores valores que los algoritmos sensibles al costo.

A continuación se presentan los resultados de aplicar el test de Friedman para verificar que existen diferencias en al menos dos de los clasificadores de los sensibles al costo y los MISORUBagging y MISORUBoost con respecto a la medida GMean (tabla 3.20).

Algorithm	Ranking
Ab1	4.0741
Ab2	4.1111
Ab3	3.6481
Ab4	5.8148
MISORUBagging	1.3889
MISORUBoost	1.963

Table 3.20: Rankings de Friedman de los algoritmos sensibles al costo, MISORUBagging y MISORUBoost y para la medida GMean

El ranking que elabora Friedman sitúa al algoritmo MISORUBagging como el mejor. Luego le siguen por su orden el algoritmo MISORUBoost, Ab3, Ab4, Ab1 y Ab2. El coeficiente obtenido por Iman-Davenport tiene un valor p de significación igual a 0.0, el cual es menor que 0.05 por lo cual se puede rechazar la hipótesis nula y afirmar que existen diferencias significativas entre al menos 2 de los clasificadores.

Para descubrir entre cuales de ellos existe diferencia significativa se aplicó el test de Holm, cuyos resultados se muestran en la tabla 3.21.

i	algorithms	$z = (R_0 - R_i)/SE$	p	Holm
15	**Ab4 vs. MISORUBagging**	**8.692346**	**0**	**0.003333**
14	**Ab4 vs. MISORUBoosting**	**7.564887**	**0**	**0.003571**
13	**Ab2 vs. MISORUBagging**	**5.346338**	**0**	**0.003846**
12	**Ab1 vs. MISORUBagging**	**5.273599**	**0**	**0.004167**
11	**Ab3 vs. MISORUBagging**	**4.437097**	**0.000009**	**0.004545**
10	**Ab3 vs. Ab4**	**4.255249**	**0.000021**	**0.005**
9	**Ab2 vs. MISORUBoost**	**4.218879**	**0.000025**	**0.005556**
8	**Ab1 vs. MISORUBoost**	**4.14614**	**0.000034**	**0.00625**
7	**Ab1 vs. Ab4**	**3.418747**	**0.000629**	**0.007143**
6	**Ab2 vs. Ab4**	**3.346008**	**0.00082**	**0.008333**
5	**Ab3 vs. MISORUBoost**	**3.309638**	**0.000934**	**0.01**
4	MISORUBagging vs. MISORUBoost	1.127459	0.259548	0.0125
3	Ab2 vs. Ab3	0.909241	0.363223	0.016667
2	Ab1 vs. Ab3	0.836502	0.402873	0.025
1	Ab1 vs. Ab2	0.072739	0.942014	0.05

Cuadro 3.21: P-values Table for $\alpha = 0.05$

Los resultados señalados en negritas muestran la existencia de diferencias significativas en las filas con i=5,...,15. Como se puede observar, los dos algoritmos desarrollados (MISORUBagging y MISORUBoost) presentan diferencias

con todos los algoritmos del estado del arte.

A modo de resumen, los clasificadores MISORUBagging y MISORUBoost presentan mejoras significativas con respecto a los algoritmos sensibles al costo del estado del arte, tomando en cuenta la medida GMean.

Conclusiones

La pregunta de investigación planteada al inicio de este estudio tiene una respuesta afirmativa:

Sí es posible desarrollar algoritmos basados en clasificadores ensamblados bagging y boosting, que incorporando soluciones al problema de clases desbalanceadas basadas en muestreo, mejoren significativamente la calidad de la clasificación multinstancia.

Particularmente a partir de este estudio se puede afirmar lo siguiente:

- Los ensambles propuestos basados en bagging son significativamente mejores que el bagging original.
- El algoritmo MISORUBagging es el mejor de las tres propuestas basadas en bagging.
- Tomando en cuenta GMean, los ensambles propuestos basados en Adaboost son significativamente mejores que el Adaboost original, sin embargo según AUC solo MISORUBoost es significativamente mejor que el Adaboost original. En cualquier caso el MISORUBoost es el mejor de las tres propuestas basadas en boosting y también es significativamente mejor que el Adaboost original.
- El algoritmo MISORUBagging es significativamente mejor el MISORUBoost.
- Los algoritmos propuestos MISORUBagging y MISORUBoost mejoran significativamente el desempeño de los algoritmos ensambles basados en costo del estado del arte.

Bibliografía

[1] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support Vector Machines for Multiple-Instance Learning. In *Advances in Neural Information Processing Systems 15 (NIPS)*, pages 561–568, 2002.

[2] M. Basu and T.K. Ho, editors. *Data Complexity in Pattern Recognition*. Advanced Information and Knowledge Processing. Springer, 2006.

[3] P. Bermejo, J. A. Gámez, and J. M. Puerta. Improving the performance of naive bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. 38:2072– 2080, 2011.

[4] H. Blockeel, D. Page, and A. Srinivasan. Multi-instance tree learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 57–64, 2005.

[5] Leo Breiman. Bagging Predictors. 1996.

[6] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16(1):321–357, 2002.

[7] Y. Chen, J. Bi, and J.Z. Wang. MILES: Multiple-instance learning via embedded instance selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1931–1947, 2006.

[8] T. G. Dietterich, R. H. Lathrop, and T. LozanoPerez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[9] A. Estabrooks and N Japkowicz. A mixture of experts framework concept-learning from imbalanced data sets. 2001.

[10] A. Estabrooks, T. Jo, and N. Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational Intelligence*, 20(1):18–36, 2004.

[11] T. Fawcett. ROC graphs with instance-varying costs. *Pattern Recognition Letters*, 27(8):882–891, 2006.

[12] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[13] Yoav Freund and Robert E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. 1997.

[14] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Dept. of Statistics, Stanford University, 1998.

[15] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. 1937.

[16] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(4):463–484, 2012.

[17] V. García, R. A. Mollineda, and J. S. Sánchez. On the k-NN performance in a challenging scenario of imbalance and overlapping. *Pattern Analysis and Applications*, 11(3-4):269–280, 2008.

[18] V. García, J. S. Sánchez, and R. A. Mollineda. On the effectiveness of preprocessing methods when dealing with different levels of class imbalance. *Knowledge-Based Systems*, 25(1):13–21, 2012.

[19] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[20] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[21] S Hu, Y Liang, L Ma, and Y He. MSMOTE: Improving classica- tion performance when training data is imbalanced. 2009.

[22] R. Iman and J. Davenport. Approximations of the critical region of the Friedman statistic. *Commun Stat Part A Theory Methods*, 9:571–595, 1980.

[23] N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–449, 2002.

[24] W. Khreich, E. Granger, A. Miri, and R. Sabourin. Iterative boolean combination of classiers in the roc space: An application to anomaly detection with hmms. 43:2732–2752, 2010.

[25] J. Kittler, M. Hatef, R Duin, and J. Matas. On combining classiers. 20, 1998.

[26] M. Kubat, R. C. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. 30:195–215, 1998.

[27] L.I. Kuncheva. Diversity in multiple classier systems. 6, 2005.

[28] V. López, A. Fernández, S. García, V. Palade, and F. Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, 2013.

[29] Y. H Lui and Y.-T Chen. Total margin-based adaptive fuzzy support vector machines for multiview face recognition. 2:1704–1711, 2005.

[30] O. Maron. *Learning from Ambiguity.* PhD Thesis, Massachusetts Institute of Technology, United States, 1998.

[31] O. Maron and T. Lozano-Pérez. A Framework for Multiple-Instance Learning. In *Advances in Neural Information Processing Systems (NIPS Conference)*, pages 570–576. MIT Press, 1998.

[32] Oded Maron and Aparna Lakshmi Ratan. Multiple-Instance Learning for Natural Scene Classification. In *Proceedings of the 15th International Conference on Machine Learning*, pages 341–349, 1998.

[33] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi. Training neural network classiers for medical decision making: The effects of imbalanced datasets on classication performance. 21:427–436, 2008.

[34] T. M. Mitchell. *Machine learning.* McGraw Hill series in computer science. McGraw-Hill, New York, NY, USA., 1997.

[35] J.F. Murray, G.F. Hughes, and K. Kreutz. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research*, 6:783–816, 2005.

[36] A. Orriols-Puig and E. Bernadó-Mansilla. Evolutionary rule-based systems for imbalanced data sets. *Soft Computing*, 13(3):213–225, 2009.

[37] R. C. Prati, G. E. Batista, and M. C. Monard. Class imbalances versus class overlapping: an analysis of a learning system behavior. In *MICAI 2004: Advances in Artificial Intelligence*, pages 312–321. Springer, 2004.

[38] F. Provost and T. Fawcett. Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distribution. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48, 1997.

[39] S. Ray and M. Craven. Supervised versus multiple instance learning: An empirical comparison. In *Proceedings of 22nd International Conference on Machine Learning (ICML-2005)*, pages 697–704. ACM Press, 2005.

[40] Soumya Ray, Stephen Scott, and Hendrik Blockeel. Multi-Instance Learning. March 2007.

[41] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

[42] Robert E. Schapire. The Strength of Weak Learnability. 1990.

[43] Stephen Scott, Jun Zhang, and Joshua Brown. On Generalized Multiple-Instance Learning. Tech Rept UNL-CSE-2003-5, University of Nebraska, Dept. of Comp. Sci., 2003.

[44] Dánel Sánchez Tarragó. *Algoritmos para la Clasificación Multinstancia*. Doctoral Thesis, Universidad de Granada, Granada, España, 2014.

[45] Y. Sun, A. K. C. Wong, and M. S. Kamel. Classification of imbalanced data: A review. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(04):687–719, 2009.

[46] Yanmin Sun, Mohamed S. Kamel, and Yang Wang. Boosting for Learning Multiple Classes with Imbalanced Class Distribution. 2006.

[47] M. Tavallaee, N. Stakhanova, and A. Ghorbani. Toward credible eval- uation of anomaly-based intrusion-detection methods. 40:516–524, September 2010.

[48] K Tumer and J. Ghosh. Error correlation and error reduction in ensemble classiers. 8, 1996.

[49] Paul Viola, John Platt, and Cha Zhang. Multiple Instance Boosting for Object Detection. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18 (NIPS Conference)*, pages 1417–1424, Cambridge, MA, 2006. MIT Press.

[50] X. Wang, S. Matwin, N. Japkowicz, and X. Liu. Cost-Sensitive Boosting Algorithms for Imbalanced Multi-instance Datasets. In *Advances in Artificial Intelligence*, pages 174–186. Springer, 2013.

[51] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, 1945.

[52] X. Xu and E. Frank. Logistic Regression and Boosting for Labeled Bags of Instances. In *Proc. of the PacificAsia Conf. on Knowledge Discovery and Data Mining*, pages 272–281. Springer-Verlag, 2004.

[53] Q. Yang and X. Wu. 10 challenging problems in data mining research. 5:597 – 604, 2006.

[54] Z. Yang, W. Tang, A. Shintemirov, and Q. Wu. Association rule mining- based dissolved gas analysis for fault diagnosis of power transformers. 39:597 – 610, 2009.

[55] Z.-H. Zhou and X.-Y. Liu. On multi-class cost-sensitive learning. *Computational Intelligence*, 26(3):232–257, 2010.

[56] Zhi-Hua Zhou, Min-Ling Zhang, Sheng-Jun Huang, and Yu-Feng Li. Multi-instance multi-label learning. *Artificial Intelligence*, 176(1):2291–2320, 2012. 00044.

[57] Z.-B Zhu and Z.-H. Song. Fault diagnosis based on imbalance modied kernel sher discriminant analysis. 88:936–951, 2010.