BIG DATA II: BIG DATA Y BIG ANALYTICS

MÁSTER EN CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

Analítica de Datos en Big Data

Autora
Lidia Sánchez Mérida

Curso 2021 - 2022

Granada, Junio de 2022

Índice

Descripción del dataset

El conjunto de datos empleado en esta práctica es conocido como ***SUSY Dataset*** y según la descripción disponible en *UCI Machine Learning Repository* [1] representa un problema de clasificación cuyo objetivo consiste en distinguir si un proceso de señales produce partículas supersimétricas. En particular, se han proporcionado dos versiones de este dataset, una reducida con un total de diez mil muestras de entrenamiento y otras diez mil de test para realizar las pruebas pertinentes de la implementación requerida más eficientemente. Mientras que el dataset con el que realmente se pretende obtener los resultados a analizar dispone de **un millón de instancias, dieciocho características y una variable dependiente binaria** para aprender.

Algoritmos de preprocesamiento

Independientemente del volumen de los conjuntos de datos, su principal característica reside en el **desbalanceamiento de clases**, lo que influye negativamente en la construcción de modelos para predecir muestras desconocidas. En la Tabla 1 se encuentra el número de ejemplos contabilizados para cada categoría en los datasets de entrenamiento y validación pertenecientes a sendas versiones. Tal y como se puede observar, el desequilibrio entre clases es sumamente más acusado en los ficheros que contienen la totalidad de información, siendo su diferencia hasta ocho veces mayor en la categoría mayoritaria que se corresponde con la positiva (clase 1).

	Versión reducida	**Versión extendida**
Entrenamiento positivas	4.907	900.000
Entrenamiento negativas	5.093	100.000
Test positivas	3.366	900.000
Test negativas	6.634	100.000

Tabla 1. Número de muestras positivas y negativas de entrenamiento y validación para el dataset reducido y su versión completa.

Con el propósito de paliar los posibles inconvenientes que se producen al modelar conjuntos de datos desbalanceados, a continuación se proponen un conjunto de

técnicas y algoritmos de preprocesamiento con los que equiparar la cantidad de ejemplos correspondientes a cada clase.

ROS

La técnica ***Random Oversampling*** efectúa un muestreo aleatorio de la clase minoritaria, que en este problema se refleja mediante la categoría negativa o cero, con el fin de duplicar las muestras existentes con las que aumentar su representación. Su integración en este proyecto ha sido posible gracias a la aportación del código en la asignatura, en mi caso seleccionando la versión de *DataFrames*. No obstante, se ha añadido una **semilla** para asegurar la reproducibilidad de los resultados.

RUS

El algoritmo ***Random Undersampling*** realiza un procedimiento contrario al explicado anteriormente, que consiste en reducir la cantidad de ejemplos pertenecientes a la clase mayoritaria, que se corresponde con la positiva cuya etiqueta es uno. Al igual que la técnica anterior, el submuestreo que lleva a cabo es completamente aleatorio. La inclusión en esta práctica también ha sido a través del código provisto en la asignatura eligiendo, de nuevo, la versión orientada a *DataFrames*. Como en el caso anterior, también se añade una **semilla** con la que generar resultados reproducibles.

ASMOTE

Existe una variante del algoritmo de balanceado de clases SMOTE denominada ***Approx-SMOTE*** que ha sido especialmente diseñado para ser aplicado de manera paralelizable en problemas de Big Data. Su principal diferencia reside en el uso de una aproximación al método *k-Nearest Neighbor* en lugar del algoritmo original, lo que produce un aumento de su escalabilidad y rendimiento [2].

La implementación de esta técnica se puede encontrar en el repositorio de GitHub titulado *Approx-SMOTE: fast SMOTE for Big Data on Apache Spark* [3] para la misma versión de Scala instalada en la máquina virtual y en el clúster, por lo que para hacer uso de este paquete basta con seguir las instrucciones de compilación proporcionadas por los autores y añadir el fichero JAR resultante a nuestro proyecto. Dispone de una doble implementación que permite su aplicación tanto a estructuras *RDD* como *DataFrames*.

En todos los experimentos su configuración ha sido generalizada al uso de un máximo de **cinco vecinos** para generar muestras sintéticas y a un porcentaje de 800% basado en

el desequilibrio que demuestra tanto el conjunto de entrenamiento como el de test en su versión extendida, cuya **clase mayoritaria supera en ocho veces** el número de muestras a la categoría minoritaria. Adicionalmente se incluye una **semilla** persiguiendo el mismo propósito consistente en la reproducibilidad de los resultados.

HME

Una de las aproximaciones que se diferencia de la tradicional consistente en añadir o eliminar muestras para balancear un dataset consiste en detectar y suprimi**r instancias ruidosas**. A esta categoría pertenece el algoritmo ***Homogeneous Ensemble***, que aplica un filtro capaz de excluir dicho tipo de instancas. Para ello divide el conjunto de datos en un número concreto de particiones de igual tamaño con las que entrenar tantos clasificadores como subconjuntos, empleando la técnica *Random Forest*. Aquellas muestras cuya clasificación haya sido errónea son tachadas de ruidosas y eliminadas del dataset.

Este algoritmo se encuentra integrado en el paquete *NoiseFramework* disponible en *SparkPackages* aunque para la versión de Scala 2.11. Por lo tanto, para poder emplearlo en nuestro proyecto he descargado el código del repositorio correspondiente en GitHub, compilándolo con ***Maven*** aumentando su versión a la 2.12.15 y solucionando los problemas procedentes de la incompatibilidad de versiones de las dependencias recopiladas en el fichero *pom.xml*. Entre ellas se sitúa el algoritmo ***kNN-IS*** cuyo JAR ha sido proporcionado en esta asignatura para la versión adecuada de Scala. A diferencia de las técnicas anteriores, este *framework* únicamente dispone de una implementación orientada al uso de ***RDD***.

En todos los experimentos los valores proporcionados a los argumentos obligatorios han sido los encontrados por defecto en los ejemplos del repositorio, estableciendo un **número máximo de cien árboles, con una profundidad de diez ramas, cuatro particiones y una semilla** para obtener los mismos resultados en distintas ejecuciones.

HTE

Dentro del paquete mencionado en la sección previa también se sitúa una segunda técnica conocida como ***Heterogeneous Ensemble*** diseñada con el mismo propósito que el primer algoritmo. Su principal diferencia reside en la combinación de **tres métodos** distintos con los que generar los clasificadores que identifican las **muestras ruidosas** para suprimirlas del conjunto de datos proporcionado. Por lo tanto, en esta técnica sí ha sido necesario integrar una votación mayoritaria entre los modelos

producidos por *Random Forest*, Regresión Logística y *kNN-IS* con el objetivo de asignar la clase correspondiente a cada instancia analizada [4].

La configuración global para las pruebas efectuadas es similar a la del algoritmo anterior estableciendo un **máximo de cien árboles con profundidad de diez ramas, cuatro particiones, vecindarios de tres muestras, votación mayoritaria y una semilla** para la reproducibilidad de los resultados.

ENN

Finalmente la técnica denominada ***Edited Nearest Neighbor*** se fundamenta en la detección de **muestras ruidosas** gracias al cálculo del nivel de similitud entre sus K vecinos más cercanos. Para asignar la categoría se hace referencia a la métrica de distancia más popularmente utilizada como es la distancia Euclídea, mientras que para llevar a cabo el procedimiento de la generación y comparación de vecindarios se emplea el algoritmo ***kNN-IS*** [4].

La configuración común a las pruebas llevadas a cabo con esta técnica incluye la especificación de un total de **tres vecinos** para la construcción de los vecindarios y una **semilla** con la que asegurar la obtención de los mismos valores en diferentes experimentos.

Algoritmos de clasificación

Una vez han sido introducidos los algoritmos de preprocesamiento con los que se intenta balancear las clases del conjunto de datos de entrenamiento, a continuación se detallan las técnicas supervisadas que resuelven el problema de clasificación. En cada una de ellas se incluye tanto una descripción de su funcionamiento, integración en el proyecto, una **configuración común** que permita realizar comparaciones sobre los rendimientos de los distintos algoritmos de preprocesamiento y el análisis de resultados procedentes de los distintos experimentos llevados a cabo con el dataset extendido. Adicionalmente se establece una **semilla común a todos los algoritmos de preprocesamiento** con el propósito de generar los mismos conjuntos de datos para entrenar los distintos clasificadores mediante diferentes técnicas de aprendizaje, de modo que también se maximice la comparabilidad de sus resultados.

Árbol de Decisión

Se trata de un procedimiento de Aprendizaje Supervisado cuyo propósito reside en construir un esquema en forma de árbol que represente las distintas **ramas** que representan los posibles valores que toman las características del dataset. Los últimos nodos conocidos como **nodos hoja** contienen la etiqueta que se le asigna a cada muestra analizada. De este modo, los ejemplos son clasificados en función de su naturaleza. Este algoritmo es considerablemente utilizado debido a su **elevada interpretabilidad** que permite conocer los motivos de las decisiones que toma, además de proporcionar una visión general del problema en cuestión.

Gracias a las implementaciones disponibles en sendas interfaces *ML,* para *DataFrames* y *MLLib* para estructuras *RDD,* ha sido posible su integración en esta práctica.

A continuación, en la Tabla 2 se muestran los resultados obtenidos de la combinación de todos los algoritmos de preprocesamiento explicados anteriormente y esta técnica particular para su evaluación en entrenamiento y test. La configuración general para los modelos se compone de los valores por defecto que imponen una **profundidad de cinco ramas, el cálculo de la impureza mediante la métrica *gini* y un total de 32 intervalos para la discretización de las características**. En relación a los resultados podemos comprobar cómo los algoritmos orientados al balanceo de clases son sumamente más efectivos que los tres últimos destinados a eliminar instancias ruidosas, por lo que como primera conclusión podemos determinar que un problema de desequilibrio de categorías no puede ser solventado utilizando únicamente este tipo de técnicas. El segundo aspecto más destacable es la perfección conseguida por el algoritmo **ASMOTE** al generar el mismo número de ejemplos positivos que negativos, lo que parece haber permitido que aumente la capacidad de predicción de los modelos predictivos entrenados con sus conjuntos de datos. Esta afirmación se debe a que la métrica de calidad referente *TP x TN*, siendo la tasa de positivos y negativos correctamente identificados, respectivamente, es la **más elevada** utilizando como método de aprendizaje un Árbol de Decisión. Adicionalmente, otro de los motivos explicativos del buen funcionamiento de este algoritmo de procesamiento se fundamenta en su habilidad para **generar muestras sintéticas** disponiendo de las originales como referencia, en lugar de crear ejemplos duplicados de los ya existentes, como son los casos de los algoritmos ROS y RUS. Por lo tanto se incluye una mayor diversidad en el conjunto de entrenamiento que permite construir un clasificador más robusto y preparado para enfrentarse a la identificación de instancias desconocidas. Finalmente es interesante observar cómo las técnicas de eliminación de ruido han **reducido drásticamente el número de muestras negativas** en más de un 90%, lo que

conlleva una pérdida de información y representación de esta clase cuyo impacto ha sido considerablemente negativo en el rendimiento de los clasificadores, obteniendo una tasa de aciertos ínfima. No obstante, en el último caso protagonizado por el método ENN el **decremento de los ejemplos positivos** también ha sido sumamente acusado, lo que ha producido una disminución de su correspondiente tasa de aciertos provocando unos pésimos resultados generales.

Preprocesamiento	Distribución de clases	Evaluación en train	Evaluación en test
ROS	1: 900.000 0: 900.867	TPR = 0,7367 TNR = 0,8032 **TPR x TNR = 0,5917**	TPR = 0,7366 TNR = 0,8013 **TPR x TNR = 0,5902**
RUS	1: 100.336 0: 100.000	TPR = 0,7306 TNR = 0,8081 **TPR x TNR = 0,5903**	TPR = 0,7292 TNR = 0,8044 **TPR x TNR = 0,5865**
ASMOTE	**1: 900.000** **0: 900.000**	**TPR = 0,7371** **TNR = 0,8329** **TPR x TNR = 0,6139**	**TPR = 0,7372** **TNR = 0,8022** **TPR x TNR = 0,5913**
HME	1: 895.243 0: 7.673	TPR = 0,9988 TNR = 0,5836 **TPR x TNR = 0,5828**	TPR = 0,9958 TNR = 0,0529 **TPR x TNR = 0,0526**
HTE	1: 825.319 0: 6.235	TPR = 0,9789 TNR = 0,5725 **TPR x TNR = 0,5604**	TPR = 0,9765 TNR = 0,0423 **TPR x TNR = 0,0413**
ENN	1: 374.526 0: 3.498	TPR = 0,9120 TNR = 0,5632 **TPR x TNR = 0,5136**	TPR = 0,7193 TNR = 0,0351 **TPR x TNR = 0,0252**

Tabla 2. Evaluación en entrenamiento y test de los experimentos realizados con varios algoritmos de preprocesamiento utilizando un Árbol de Decisión.

Gradient-Boosted Trees

Es un algoritmo popularmente utilizado tanto en problemas de clasificación como de regresión cuya principal ventaja reside en la combinación de un conjunto de modelos, generalmente aplicando la técnica de Árboles de Decisión para su construcción. Esta metodología en particular es conocida como ***ensemble*** y se caracteriza por mejorar considerablemente la calidad de los resultados gracias a la conjunción del conocimiento adquirido por cada uno de los clasificadores que participan en la identificación de las muestras proporcionadas.

Para hacer referencia a todos los algoritmos de preprocesamiento comentados anteriormente, se han integrado las dos versiones de la técnica GBT relativas a las estructuras de *DataFrames* y *RDD*.

En la Tabla 3 se encuentran las métricas de calidad calculadas para los experimentos realizados efectuando su evaluación tanto en entrenamiento como en test. Su configuración común consiste en establecer una **profundidad máxima de cinco ramas y un total de 32 intervalos para discretizar las características.** En cuanto a los valores representados podemos observar una tendencia similar al del caso anterior con un único árbol de decisión. No obstante, la principal distinción reside en que en la evaluación sobre **entrenamiento** es la combinación con el algoritmo **ASMOTE** la que consigue una métrica de bondad superior a los restantes métodos de balanceado, mientras que en el dataset de **validación** el mejor resultado lo proporciona la técnica **ROS**, aunque únicamente por cuatro diezmilésimas. Por lo tanto, utilizando GBT las diferencias entre el rendimiento de los modelos entrenados por sendos algoritmos de procesamiento se reducen considerablemente hasta el límite de presentar prácticamente las mismas tasas de aciertos. Una posible explicación de este fenómeno reside en que tanto ROS como ASMOTE llevan a cabo la sintetización de **nuevas muestras** que aumentan la representación de la clase minoritaria, aunque utilizando procedimientos diferentes. Así se potencia la presencia de la categoría negativa mientras que evita la pérdida de información valiosa de la clase positiva.

Preprocesamiento	Distribución de clases	Evaluación en train	Evaluación en test
ROS	1: 900.000 0: 900.867	TPR = 0,7299 TNR = 0,8473 **TPR x TNR = 0,6184**	TPR = 0,7300 TNR = 0,8442 **TPR x TNR = 0,6162**
RUS	1: 100.336 0: 100.000	TPR = 0,7292 TNR = 0,8495 **TPR x TNR = 0,6194**	TPR = 0,7268 TNR = 0,8464 **TPR x TNR = 0,6151**
ASMOTE	1: 900.000 0: 900.000	TPR = 0,7399 TNR = 0,8626 **TPR x TNR = 0,6382**	TPR = 0,7395 TNR = 0,8328 **TPR x TNR = 0,6158**
HME	1: 895.243 0: 7.673	TPR = 0,9991 TNR = 0,5854 **TPR x TNR = 0,5848**	TPR = 0,9962 TNR = 0,0504 **TPR x TNR = 0,0502**
HTE	1: 825.319 0: 6.235	TPR = 0,9834 TNR = 0,5703 **TPR x TNR = 0,5608**	TPR = 0,9798 TNR = 0,0411 **TPR x TNR = 0,0402**
ENN	1: 374.526	TPR = 0,9365	TPR = 0,7225

	0: 3.498	TNR = 0,5739 **TPR x TNR = 0,5374**	TNR = 0,0369 **TPR x TNR = 0,0266**

Tabla 3. Evaluación en entrenamiento y test de los experimentos realizados con varios algoritmos de preprocesamiento utilizando Gradient Boosted Trees.

Random Forest

El algoritmo *Random Forest* es una de las técnicas de Aprendizaje Automático más extendidas debido a su enorme capacidad de adaptación a casi cualquier problema y conjunto de datos, proporcionando resultados de alta calidad. Su mecanismo de funcionamiento reside en el entrenamiento y la combinación de un elevado número de modelos de Árboles de Decisión que utilizan subconjuntos de datos **muestreados aleatoriamente** con el propósito de reducir la probabilidad de sobreaprendizaje. Como en el caso anterior este método también se considera un ***ensemble*** que efectúa un voto mayoritario para asignar la etiqueta con más apoyos.

De nuevo, este algoritmo ha sido incluido en el proyecto mediante la librería *ML* con la que hacer referencia a los algoritmos de preprocesamiento orientados a *DataFrames*, mientras que la interfaz *MLLib* permite el uso de las técnicas únicamente implementadas mediante *RDD*.

En la Tabla 4 se recopilan las medidas de bondad extraídas de las distintas pruebas ejecutadas con *Random Forest* tanto sobre el conjunto de entrenamiento como sobre el conjunto de validación. Como configuración se establece la construcción de un **máximo de veinte árboles de profundidad cinco, usando *gini* como métrica de impureza y 32 intervalos para la discretización** de variables. Tal y como ocurría con el primer algoritmo de aprendizaje, de nuevo ha sido **ASMOTE** con el que los modelos entrenados mediante Random Forest han proporcionado las tasas de aciertos más elevadas de los experimentos realizados, presumiblemente gracias a la **diversidad** que introduce al crear nuevas instancias de la clase minoritaria, en lugar de replicar los ejemplos existentes. No obstante, no existe una gran diferencia con sus homólogas técnicas de balanceo de clases ROS y RUS. Por otro lado, los métodos de eliminación de ruido siguen produciendo una enorme confusión en los clasificadores que son totalmente incapaces de predecir la **clase minoritaria o negativa**, lo que no permite solventar el desequilibrio intrínseco a este problema de clasificación.

Preprocesamiento	Distribución de clases	Evaluación en train	Evaluación en test

ROS	1: 900.000 0: 900.867	TPR = 0,7332 TNR = 0,8159 **TPR x TNR = 0,5982**	TPR = 0,7227 TNR = 0,8132 **TPR x TNR = 0,5876**
RUS	1: 100.336 0: 100.000	TPR = 0,7250 TNR = 0,8193 **TPR x TNR = 0,5939**	TPR = 0,7236 TNR = 0,8167 **TPR x TNR = 0,5909**
ASMOTE	1: 900.000 0: 900.000	TPR = 0,7322 TNR = 0,8336 **TPR x TNR = 0,6103**	TPR = 0,7315 TNR = 0,8032 **TPR x TNR = 0,5875**
HME	1: 895.243 0: 7.673	TPR = 0,9741 TNR = 0,5713 **TPR x TNR = 0,5565**	TPR = 0,9685 TNR = 0,0412 **TPR x TNR = 0,0399**
HTE	1: 825.319 0: 6.235	TPR = 0,9658 TNR = 0,5643 **TPR x TNR = 0,5450**	TPR = 0,9527 TNR = 0,0364 **TPR x TNR = 0,0346**
ENN	1: 374.526 0: 3.498	TPR = 0,9106 TNR = 0,5548 **TPR x TNR = 0,5052**	TPR = 0,7031 TNR = 0,0245 **TPR x TNR = 0,0172**

Tabla 4. Evaluación en entrenamiento y test de los experimentos realizados con varios algoritmos de preprocesamiento utilizando Random Forest.

Regresión Logística

Se trata de un algoritmo estadístico cuyo principal objetivo consiste en aproximar una **función matemática** acotada en el intervalo [0, 1] con la que predecir la probabilidad de pertenencia de una muestra a cada una de las clases disponibles. Esta técnica asume la existencia de una **relación lineal** entre las variables predictoras especificadas en la formulación, si no se concretan por defecto se toman todas las que se encuentren disponibles.

Al igual que el resto de algoritmos de aprendizaje, la Regresión Logística se ha integrado en el proyecto a partir de las interfaces *ML* y *MLLib* para combinar esta técnica con todas las de preprocesamiento explicadas en secciones anteriores.

La tabla 5 representa las métricas de calidad calculadas en los diversos experimentos realizados sobre los conjuntos de entrenamiento y validación. La configuración general consiste en aplicar **regularización L2** con el fin de evitar el sobreajuste de los modelos. Tal y como podemos apreciar se mantiene la misma tónica del comportamiento de los distintos algoritmos de procesamiento en el entrenamiento de

modelos con Regresión Logística Binomial. Las **diferencias** entre los valores proporcionados por las tres primeras técnicas para balancear las clases son **menores** que en los casos previos, a excepción de la evaluación sobre el conjunto de entrenamiento con más de dos puntos generados por **ASMOTE** con respecto a los restantes. Por otra parte, podemos apreciar un ligero incremento en las métricas de bondad relativas al uso exclusivo de métodos de eliminación de instancias ruidosas, aunque apenas alcanza el 10% de los aciertos positivos y negativos. Como conclusión de este fenómenos parece que la Regresión Logística Binomial es levemente **menos sensible a la reducción** tan drástica que realizan estos tres métodos especialmente sobre la clase minoritaria o negativa.

Preprocesamiento	Distribución de clases	Evaluación en train	Evaluación en test
ROS	1: 900.000 0: 900.867	TPR = 0,7079 TNR = 0,8510 **TPR x TNR = 0,6024**	TPR = 0,7084 TNR = 0,8520 **TPR x TNR = 0,6035**
RUS	1: 100.336 0: 100.000	TPR = 0,7101 TNR = 0,8514 **TPR x TNR = 0,6045**	TPR = 0,7084 TNR = 0,8519 **TPR x TNR = 0,6034**
ASMOTE	1: 900.000 0: 900.000	TPR = 0,7212 TNR = 0,8623 **TPR x TNR = 0,6218**	TPR = 0,7218 TNR = 0,8361 **TPR x TNR = 0,6034**
HME	1: 895.243 0: 7.673	TPR = 0,9989 TNR = 0,1085 **TPR x TNR = 0,1083**	TPR = 0,9984 TNR = 0,0186 **TPR x TNR = 0,0185**
HTE	1: 825.319 0: 6.235	TPR = 0,9798 TNR = 0,0963 **TPR x TNR = 0,0943**	TPR = 0,9880 TNR = 0,0173 **TPR x TNR = 0,0170**
ENN	1: 374.526 0: 3.498	TPR = 0,9401 TNR = 0,0863 **TPR x TNR = 0,0811**	TPR = 0,6423 TNR = 0,0146 **TPR x TNR = 0,0093**

Tabla 5. Evaluación en entrenamiento y test de los experimentos realizados con varios algoritmos de preprocesamiento utilizando Regresión Logística.

kNN-IS

El algoritmo ***k-Nearest Neighbors*** es uno de los métodos de clasificación más sencillos de comprender e incluso replicar manualmente. Su procedimiento se fundamenta en disponer de un conjunto de muestras con sus correspondientes etiquetas para posteriormente calcular el grado de similitud entre su totalidad y cada nuevo

ejemplo a identificar. Para ello se emplean diversas métricas de **distancia**, como la distancia Euclídea, que permite componer un **vecindario** de *K* muestras para asignar la categoría.

Con el propósito de combinar esta técnica con los métodos de preprocesamiento se ha incluido tanto su versión con *DataFrames* como con *RDD* tal y como se representa en los ejemplos propuestos situados en el repositorio oficial del algoritmo [7].

En la Tabla 6 se recoge la bondad de los distintos clasificadores construidos a partir de su combinación con los algoritmos de balanceo de clases y reducción de ruido. La configuración común a todos los experimentos establece un **número máximo de tres vecinos, la distancia Euclídea** como métrica de similitud, **cien iteraciones, mil procesos *map* y trescientos *reduce*.** A diferencia de los algoritmos de aprendizaje procedentes de secciones previas, ha sido la técnica de equilibrio de clases **RUS** la que **mejores métricas de calidad** ha proporcionado en combinación con el método kNN-IS. Si bien existen diferencias mínimas con sus homólogos ROS y ASMOTE, presumiblemente la reducción del número de instancias en términos generales ha beneficiado la composición de los vecindarios con los que seleccionar la etiqueta para cada muestra. Adicionalmente, con RUS se confirma que también se produce un **decremento** considerablemente relevante tanto en la inversión de **recursos** computacionales como temporales. No obstante, hasta el momento se trata de los modelos con **peores capacidades predictivas** de los experimentos analizados hasta el momento, y es que tal y como conocemos el modo de funcionamiento del kNN, realmente no es un algoritmo que presente ningún tipo de aprendizaje por lo que sus resultados siempre se encuentran más limitados comparados con los restantes métodos. Este fenómeno produce un tenue decremento de las tasas de aciertos conseguidas por los procedimientos de eliminación de ruido, que utilizados en exclusiva no solventan el problema de desequilibrio de clases que presenta el dataset SUSY.

Preprocesamiento	Distribución de clases	Evaluación en train	Evaluación en test
ROS	1: 900.000 0: 900.867	TPR = 0,7125 TNR = 0,6987 **TPR x TNR = 0,4978**	TPR = 0,7036 TNR = 0,6774 **TPR x TNR = 0,4766**
RUS	1: 100.336 0: 100.000	TPR = 0,7585 TNR = 0,6930 **TPR x TNR = 0,5256**	TPR = 0,7384 TNR = 0,6812 **TPR x TNR = 0,5029**
ASMOTE	1: 900.000 0: 900.000	TPR = 0,7237 TNR = 0,6941 **TPR x TNR = 0,5023**	TPR = 0,7115 TNR = 0,6885 **TPR x TNR = 0,4898**

HME	1: 895.243 0: 7.673	TPR = 0,9760 TNR = 0,0963 **TPR x TNR = 0,0939**	TPR = 0,9658 TNR = 0,0121 **TPR x TNR = 0,0116**
HTE	1: 825.319 0: 6.235	TPR = 0,9629 TNR = 0,0901 **TPR x TNR = 0,0867**	TPR = 0,9622 TNR = 0,0153 **TPR x TNR = 0,0147**
ENN	1: 374.526 0: 3.498	TPR = 0,9137 TNR = 0,0632 **TPR x TNR = 0,0577**	TPR = 0,9041 TNR = 0,0119 **TPR x TNR = 0,0107**

Tabla 6. Evaluación en entrenamiento y test de los experimentos realizados con varios algoritmos de preprocesamiento utilizando kNN-IS.

LightGBM

Light Gradient-Boosted Machines es una variante del algoritmo GBT anteriormente explicado que integra dos principales novedades que le permiten aumentar su eficiencia en tiempo de computación sin decrementar su rendimiento en términos de precisión. Con la técnica ***Gradient-based One-Side Sampling*** (GOSS) es capaz de muestrear el conjunto de datos con el fin de obtener únicamente las muestras más relevantes y representativas para entrenamiento, mientras que las restantes se emplean en la fase de validación. Por otra parte con el procedimiento denominado ***Exclusive Feature Bundling*** (EFB) se realiza una selección de características de modo que se reduce la dimensionalidad del problema [8].

De manera opuesta a los algoritmos previos, **no existe una implementación disponible para RDD** por lo que únicamente se han podido realizar experimentos con aquellos algoritmos de preprocesamiento que permiten trabajar con *DataFrames*.

En la Tabla 7 se reflejan las medidas de calidad obtenidas en las distintas pruebas efectuadas considerando tanto la fase de entrenamiento como la etapa de validación. Como única configuración se ha especificado un número **máximo de cien iteraciones**. Con respecto a los resultados el primer aspecto destacable es la **ínfima calidad** proporcionada por la totalidad de los modelos producidos en los distintos experimentos. En los algoritmos **ROS y ASMOTE**, ambos generadores de nuevas instancias de la clase minoritaria, los clasificadores se encuentran completamente **sesgados por la clase positiva**, la originalmente mayoritaria en los conjuntos de datos SUSY iniciales. Esta indeseada cualidad se traduce directamente en un acierto casi pleno de los ejemplos pertenecientes a esta categoría, mientras que por el contrario

apenas es capaz de identificar un 0,01% de muestras negativas. El caso contrario se refleja en el uso del método **RUS**, que reduce los ejemplos positivos, cuyos clasificadores apuestan totalmente por esta categoría, dejando sin identificar prácticamente a la totalidad de ejemplos negativos. Tras observar los buenos resultados proporcionados por estos tres métodos de balanceo de clases en los algoritmos de aprendizaje previos, la única conclusión plausible que explica este fenómeno es la existencia de algún **tipo de error en la implementación** de esta técnica que no permite la construcción correcta de modelos predictivos.

Preprocesamiento	Distribución de clases	Evaluación en train	Evaluación en test
ROS	1: 900.000 0: 900.867	TPR = 0,9935 TNR = 0,0154 **TPR x TNR = 0,0152**	TPR = 0,9934 TNR = 0,0146 **TPR x TNR = 0,0145**
RUS	1: 100.336 0: 100.000	TPR = 0,0088 TNR = 0,9968 **TPR x TNR = 0,0087**	TPR = 0,0085 TNR = 0,9968 **TPR x TNR = 0,0084**
ASMOTE	1: 900.000 0: 900.000	TPR = 0,9998 TNR = 0,0004 **TPR x TNR = 0,0003**	TPR = 0,9999 TNR = 0,0006 **TPR x TNR = 0,0005**

Tabla 7. Evaluación en entrenamiento y test de los experimentos realizados con varios algoritmos de preprocesamiento utilizando LightGBM.

Conclusiones

Tal y como se ha podido apreciar en los análisis efectuados sobre las distintas combinaciones de algoritmos de preprocesamiento y métodos de aprendizaje, las técnicas orientadas al **balanceado de clases** han sido las únicas capaces de mitigar las nefastas consecuencias que producen los conjuntos desequilibrados en los modelos predictivos. El uso únicamente de algoritmos de **reducción de ruido** no es suficiente para solventar esta problemática, por lo que los datasets resultantes continúan caracterizándose por el mismo desbalance de clases. Adicionalmente, destacamos la **pérdida de información** que se produce al referenciar estos procedimientos que pueden ser extremas, como se ha observado en el caso del algoritmo *ENN*. Como consecuencia directa también se aprecia un impacto negativo en la **capacidad predictiva** de los clasificadores por el decremento producido en la tasa de aciertos de la categoría afectada. Este fenómeno ha sido detectado al emplear el algoritmo ***RUS***, aunque en menor medida, puesto que a excepción del algoritmo *kNN-IS* han sido las

técnicas de generación de **muestras sintéticas** las que más han beneficiado a la construcción de modelos por sus mayores tasas de aciertos en ambas clases.

En la Tabla 8 se recoge un **ranking** de las combinaciones más exitosas para cada algoritmo de aprendizaje tanto para el conjunto de entrenamiento como para el de test. Como se puede apreciar, el algoritmo ***Gradient-Boosted Trees*** junto con el método de balanceado ASMOTE ha sido el que ha proporcionado una mayor tasa de aciertos sobre entrenamiento. Mientras que con el procedimiento ROS ha conseguido el porcentaje de aciertos sobre test más elevado en comparación con todos los participantes. A menos de un punto de diferencia se encuentran los modelos entrenados mediante **Regresión Logística**, en cuyos resultados también se repite la misma conjunción de técnicas de balanceo de clases. De nuevo parece confirmarse la teoría de los beneficios aportados por la generación de muestras sintéticas para resolver el desequilibrio de clases, en lugar de apostar por la reducción de ejemplos procedentes de la categoría mayoritaria. En los siguientes puestos se encuentran los **otros dos algoritmos de árboles** con unos resultados bastante parecidos a los dos primeros aunque el método de balanceo protagonista es únicamente ASMOTE. En penúltimo lugar se sitúa el método *kNN-IS* en consonancia con RUS cuyas tasas de aciertos se encuentran diez puntos por debajo de las comentadas anteriormente, por lo que parece que esta técnica al ser menos sofisticada encuentra más inconvenientes para aprender los patrones de las muestras del dataset SUSY. Finalmente disponemos del algoritmo *LightGBM* con sus pésimos resultados presumiblemente originados por algún tipo de error en la implementación de la biblioteca que se ha integrado en este proyecto.

Evaluación en entrenamiento	Evaluación en test
ASMOTE + GBT = 0,6382	ROS + GBT = 0,6162
ASMOTE + Regresión Logística = 0,6218	ROS + Regresión Logística = 0,6035
ASMOTE + Árbol de Decisión = 0,6139	ASMOTE + Árbol de Decisión = 0,5913
ASMOTE + Random Forest = 0,6103	ASMOTE + Random Forest = 0,5875
RUS + kNN-IS = 0,5256	RUS + kNN-IS = 0,5029
ROS + LightGBM = 0,0152	ROS + LightGBM = 0,0145

Tabla 8. Ranking de las combinaciones entre algoritmos de preprocesamiento y aprendizaje según sus evaluaciones en entrenamiento y test.

Bibliografía

1. UCI Machine Learning Repositor, SUSY Data Set, 2014.
2. Mario Juez-Gil, Álvar Arnaiz-González, Juan J. Rodríguez, Carlos López-Nozal, César García-Osorio, Approx-SMOTE: Fast SMOTE for Big Data on Apache Spark, 2021
3. Mario Juez-Gil, Álvar Arnaiz-González, Juan J. Rodríguez, Carlos López-Nozal, César García-Osorio, Approx-SMOTE: fast SMOTE for Big Data on Apache Spark, GitHub, 2021
4. Diego García, Julián Luengo, Salvador García, Francisco Herrera, Enabling Smart Data: Noise filtering in Big Data classification, 2017.
5. Diego García, Julián Luengo, Salvador García, Francisco Herrera, NoiseFramework, SparkPackages, 2018.
6. Diego García, Julián Luengo, Salvador García, Francisco Herrera, NoiseFramework, GitHub, 2018.
7. J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera. kNN-IS: An Iterative Spark-based design of the k-Nearest Neighbors classifier for big data, 2017.
8. Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu, LightGBM: A Highly Efficient Gradient Boosting Decision Tree, 2017.