UNIVERSIDAD DE GRANADA

MÁSTER DE CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

CURSO ACADÉMICO 2019-2020

BIG DATA II

Analítica de datos con Big Data.

Análisis, preprocesamiento y construción de modelos de clasificación sobre un conjunto masivo de datos con Spark.

Nicolás Cubero

29 de Mayo de 2020

Índice

1. Introducción

El término **Big Data** caracteriza un nuevo tipo de datos que se caracterizan, principalmente, por su gran volumen, diversidad y velocidad en la que son generados y que imposibiltan su tratamiento por métodos y arquitecturas tradicionales.

La aparición del **Big Data** trae consigo el desarrollo de nuevas arquitecturas, algoritmos y métodos de procesamiento y análisis con la finalidad de permitir el tratamiento de este tipo de datos y, en concreto, en el ámbito de la Minería de Datos, para permitir la explotación y el descubrimiento de información no implícita y de interés a partir de este conjunto voluminoso de datos.

En este proyecto se persigue la aplicación de diversos procesos de Minería de Datos en el contexto de **Big Data** para el estudio, preprocesamiento y desarrollo de modelos de clasificación óptimos sobre el conjunto de datos masivo ***Higgs*** [1]

El *dataset* ***Higgs*** contiene 11.000.000 instancias de las cuales se reservan 1.002.434 para el conjunto de *train* y 999.404 para el conjunto de *test*, referidas a simulaciones de Monte Carlo sobre colisiones entre partículas de *Bosón de Higgs*, midiendo para las mismas 28 características numéricas reales referentes a sus propiedades kinemáticas así, como a otras propiedades físicas de más alto nivel para tratar de determinar si cada colisión se corresponde a una señal de energía detectable o de fondo.

El *dataset* ***Higgs*** es un dataset masivo y desbalanceado con una proporción del 90 % de las instancias pertenecientes a la clase mayoritaria y el 10 % restante a la clase minoritaria, por lo que se hace necesario hacer uso de diveros métodos de preprocesamiento para el balanceamiento de las clases además de otras técnicas de preprocesamiento oportunas para garantizar la limpieza de los datos. Más concretamente, se probarán a usar como métodos de balanceamiento de clases, *Random Oversamplig* (ROS), *Random Undersamplig* (RUS) y el *ensembler* homogéneo *HME BD* para el filtrado de ruido, analizando y comparando la mejora proporcinada por los mismos a los diferentes metodos de clasificación aplicados.

[1] Enlace al repositorio de la UCI con el dataset Higgs: https://archive.ics.uci.edu/ml/datasets/HIGGS

Respecto a los modelos predictivos de clasificación, se hará uso de modelos basados en árboles de decisión (*Decission Tree*), ***Random Forest*** y ***K-Nearest-Neighbourgh*** (kNN), acompañados con los métodos de preprocesamiento citados y analizando, optimizando y comparando sus rendimientos sobre el *dataset*.

Todas estas ejecuciones se desarrollarán en una arquitectura distribuida haciendo uso de **Spark** y utilizando las implementaciones de estos métodos proporcionadas en *Spark Mllib* y *Spark package*.

2. Construción de modelos *Decission Tree*

Primeramente, se trabaja con modelos de clasificación basados en *Decission Tree* junto a las diferentes técnicas de preprocesamiento y limpieza de datos mencionadas en la Introducción.

2.1. Sobremuestreo de la clase minoritaria con *ROS*

En primer lugar, se trata el entrenamiento de modelos de *Decission Tree* con datos de entrenamiento para los cuales se ha hecho uso de *Random Oversampling* (ROS) para incrementar el número de patrones de la clase minoritaria:

El flujo de ejecución de cada experimento es el siguiente: Tras cargar los datos, a los datos de entrenamiento, le son aplicados el método *ROS* para incrementar el número de patrones pertenecientes a la clase minoritaria, considerando diferentes ratios de balanceamiento.

Una vez llevado a cabo este sobremuestreo que lleva a la introducción de nuevos patrones en el conjunto de entrenamiento, se entrena un modelo de *Decision Tree* variando sus parámetros de profundidad máxima (`maxDepth`), el número máximo de divisiones (`maxBins`). En todos los experimentos se ha hecho uso del índice *gini* como medida de impuridad de los nodos.

De forma más concreta, se probarán los siguientes valores:

- $maxDepth \in \{5, 10, 17, 25\}$
- $maxBins \in \{10, 32, 48, 64, 100, 256, 500, 1000\}$
- *ratio sobremuestreo* $:\in \{1,\ 0{,}7,\ 0{,}3\}$

Finalmente, se evaluá el modelo desarrollado sobre el conjunto de *train* y sobre el conjunto de *test* y se miden las métricas de rendimiento: *accuracy* y el producto del ratio de aciertos positivos y el ratio de aciertos negativos *TPR x TNR*:

- **Considerando un ratio de balanceamiento de 1.0**: Se representa en la siguiente tabla los valores de las métricas de rendimiento obtenidos en cada experimentación con *Decission Tree* considerando diferentes valores de `maxDepth` y `maxBins`:

		Training		Test	
Max Depth	Max Bins	TPR x TNR	accuracy	TPR x TNR	accuracy
5	10	0.03989	0.71990	0.10982	0.66326
	32	0.03935	0.72239	0.10843	0.65944
	48	0.039102	0.72751	0.10786	0.65816
	64	0.03938	0.71984	0.10856	0.65972
	100	0.039399	0.71914	0.10844	0.65930
	256	0.03927	0.72709	0.10807	0.65867
	500	0.03922	0.73065	0.10789	0.65841
	1000	0.03925	0.72988	0.10797	0.65861
10	10	0.04529	0.70818	0.12090	0.69617
	32	0.04563	0.71341	0.12118	0.69675
	48	0.04573	0.71212	0.12117	0.69682
	64	0.04573	0.71282	0.12105	0.69640
	100	0.04588	0.70790	0.12129	**0.69746**
	256	0.04582	0.70955	0.12101	0.69648
	500	0.04596	0.71073	0.12123	0.69708
	1000	0.04608	0.71399	**0.12140**	0.69742
17	10	0.06432	0.80512	0.10835	0.65998
	32	0.06324	0.79501	0.11085	0.66677
	48	0.06325	0.80355	0.11054	0.66646
	64	0.06362	0.80312	0.11152	0.66911
	100	0.06318	0.79572	0.11228	0.67096
	256	0.06315	0.79619	0.11253	0.67167
	500	0.06328	0.80228	0.11216	0.67096
	1000	0.06286	0.79934	0.11280	0.67267
25	10	**0.08660**	**0.96884**	0.06981	0.58391
	32	0.08485	0.95190	0.08071	0.60612
	48	0.08396	0.94477	0.08435	0.61382
	64	0.08405	0.94512	0.08475	0.61482

100	0.08366	0.94062	0.08732	0.62009
256	0.08297	0.93490	0.09067	0.62770
500	0.08264	0.93417	0.09091	0.62811
1000	0.08249	0.93236	0.09084	0.62770

Tabla 1: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Decission Tree considerando un ratio de sobrebalanceamiento del 100 %. Se ha remarcado en negrita los valores más altos de cada métrica

Se observa que en el anterior conjunto de experimentos, los parámetros que permiten el desarrollo del modelo óptimo que maximiza el valor de la métrica $TPRxTNR$ sobre el conjunto de test son $MaxDepth = 10$ y $MaxBins = 1000$, que proporciona un valor de $TPRxTNR = 0{,}121402019347122$, mientras que para el conjunto de *train*, el modelo que provee mejores resultados resulta $MaxDepth = 25$ y $MaxBins = 10$.

Probamos a realizar otras experimentaciones considerando valores menores para el ratio de sobremuestreo.

- **Considerando un ratio de balanceamiento de 0.7**: Los valores de las métricas considerando las mismas configuraciones de los parámetros `maxDepth` y `maxBins` resultan los siguientes:

		Training		Test	
Max Depth	Max Bins	TPR x TNR	Accuracy	TPR x TNR	Accuracy
5	10	0.03643	0.77432	0.10041	0.64383
	32	0.03662	0.78358	0.10099	0.64686
	48	0.03458	0.80545	0.09527	0.63718
	64	0.03614	0.79367	0.09954	0.64504
	100	0.03674	0.78226	0.10123	0.64733
	256	0.03672	0.78730	0.10102	0.64748
	500	0.03642	0.78931	0.10016	0.64575
	1000	0.03414	0.81337	0.09400	0.63590
10	10	0.04234	0.79596	0.11283	0.67814
	32	0.04377	0.79021	0.11574	0.68471
	48	0.04349	0.79443	0.11458	0.68227
	64	0.04376	0.78910	0.11547	0.68394
	100	0.04444	0.78436	**0.11715**	**0.68762**
	256	0.04378	0.79472	0.11460	0.68220

	500	0.04424	0.78771	0.11645	0.68623
	1000	0.04344	0.79807	0.11348	0.67976
17	10	0.06331	0.84180	0.10516	0.65589
	32	0.06338	0.85190	0.10469	0.65640
	48	0.06316	0.85261	0.10503	0.65742
	64	0.06416	0.84835	0.10614	0.65939
	100	0.06397	0.85388	0.10588	0.65959
	256	0.06327	0.85090	0.10595	0.65964
	500	0.06345	0.84988	0.10648	0.66079
	1000	0.06361	0.85148	0.10680	0.66187
25	10	**0.08707**	**0.97722**	0.06903	0.58340
	32	0.08615	0.96888	0.07569	0.59763
	48	0.08562	0.96533	0.07799	0.60301
	64	0.08571	0.96594	0.07859	0.60414
	100	0.08511	0.96240	0.08040	0.60818
	256	0.08443	0.95733	0.08324	0.61400
	500	0.08452	0.95919	0.08218	0.61176
	1000	0.08407	0.95626	0.08403	0.61601

Tabla 2: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Decission Tree considerando un ratio de sobrebalanceamiento del 70 %. Se ha remarcado en negrita los valores más altos de cada métrica.

Considerando un ratio de sobrebalanceamiento de 70 %, el modelo óptimo que ofrece mejore resultados sobre el conjunto de *test* se obtiene considerando valores de $MaxDepth = 10$ y $MaxBins = 100$. Sobre el conjunto de *train*, el modelo que ofrece mejores resultados se obtiene nuevamente con $MaxDepth = 25$ y $MaxBins = 10$.

- **Considerando un ratio de balanceamiento de 0.3**:

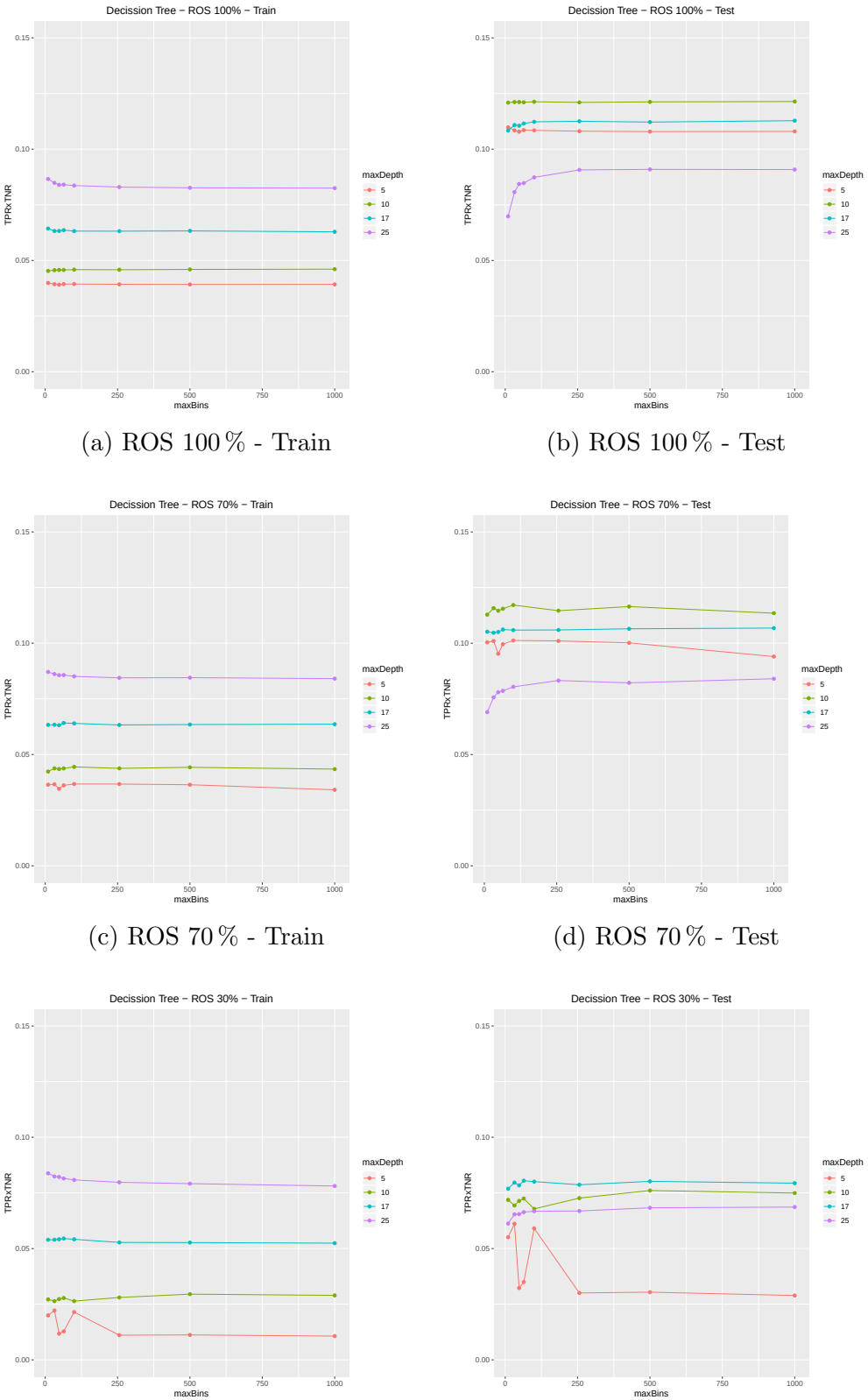
		Training		Test	
Max Depth	Max Bins	TPR x TNR	Accuracy	TPR x TNR	Accuracy
5	10	0.01996	0.88243	0.05509	0.57217
	32	0.02218	0.87672	0.06115	0.58265
	48	0.01175	0.89504	0.03226	0.53111
	64	0.01279	0.89270	0.03497	0.53538
	100	0.02146	0.87964	0.05908	0.57934
	256	0.01108	0.89567	0.03005	0.52696
	500	0.01120	0.89551	0.03041	0.52762
	1000	0.01067	0.89657	0.02891	0.52510

10	10	0.02716	0.88559	0.07186	0.60717
	32	0.02637	0.88801	0.06934	0.60257
	48	0.02731	0.88651	0.07141	0.60627
	64	0.02777	0.88646	0.07247	0.60836
	100	0.02638	0.88898	0.06783	0.59932
	256	0.02802	0.88599	0.07267	0.60849
	500	0.02952	0.88241	0.07608	0.61432
	1000	0.02896	0.88441	0.07494	0.61279
17	10	0.05396	0.91525	0.07687	0.60615
	32	0.05397	0.91604	0.07962	0.61370
	48	0.05417	0.91919	0.07840	0.61168
	64	0.05452	0.91752	**0.08048**	0.61630
	100	0.05416	0.91768	0.08006	0.61572
	256	0.05275	0.91861	0.07866	0.61352
	500	0.05269	0.91636	0.08019	**0.61691**
	1000	0.05242	0.91768	0.07938	0.61559
25	10	**0.08382**	**0.98829**	0.06124	0.56764
	32	0.08239	0.98404	0.06538	0.57864
	48	0.08217	0.98392	0.06546	0.57901
	64	0.08149	0.98266	0.06636	0.58144
	100	0.08084	0.98125	0.06679	0.58304
	256	0.07977	0.98010	0.06687	0.58384
	500	0.07916	0.97829	0.06831	0.58735
	1000	0.07810	0.97613	0.06863	0.58834

Tabla 3: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Decission Tree considerando un ratio de sobrebalanceamiento del 30 %. Se ha remarcado en negrita los valores más altos de cada métrica.

Con este ratio de sobrebalanceamiento, el modelo que obtiene mejor medida de $TPRxTNR$ en test, es aquel con $maxDepth = 17$ y $maxBins = 64$, mientras que el modelo que ofrece mejores rendimientos sobre el conjunto de *train* es también $maxDepth = 25$ y $maxBins = 10$.

Analizamos estos resultados de forma resumida por medio de las siguientes gráficas en la que se representa los valores de $TPRxTNR$ obtenidos para cada valor de `maxBins` y `maxDepth`:



(a) ROS 100 % - Train

(b) ROS 100 % - Test

(c) ROS 70 % - Train

(d) ROS 70 % - Test

(e) ROS 30 % - Train

(f) ROS 30 % - Test

Figura 1: Valores de TPRxTNR obtenidos por los modelos de Decission Tree ante diferentes valores de los parámetros maxBins, maxDepth y variando el ratio de sobremuestreo evaluados sobre el conjunto de train (a la izquierda) y sobre el conjunto de test (a la derecha)

De los anteriores experimentos, podemos realizar las siguientes observaciones:

- Al **disminuir el ratio de sobremuestreo, los rendimientos de los modelos disminuyen de forma proporcional**. Se puede comprobar que el desbalanceamiento del *dataset* supone un importante obstáculo para la obtención de resultados óptimos, de modo que los mejores resultados se obtienen con un ratio de sobrebalanceo del 100 %.

- Los modelos con rendimientos óptimos y que generalizan mejor, se obtiene con una profundidad máxima de árbol (`maxDepth`) entorno a 10 si se considera un sobrebalanceamiento total.

  Para profundidades mayores, en los rendimientos medidos sobre el conjunto de *train*, se aprecia un incremento en este rendimiento sobre este conjunto, al tiempo que el rendimiento sobre el conjunto de *test* disminuye progresivamente, lo cual refleja un **sobreentrenamiento del modelo al incrementar la profundidad máxima por encima de este valor**.

- **El modelo óptimo** con mayor capacidad de generalización (modelo con las métricas de rendimiento más altas sobre el conjunto de test), **se obtiene con un ratio de sobremuestreo del 100 % con un número máximo de divisiones (`maxBins`) de 1000 y una profundidad máxima de árbol de 10**.

2.2. Submuestreo de la clase mayoritaria con *RUS*

Repetimos las anteriores eperimentaciones para el entrenamiento de modelos basados en *Decission Tree*, pero esta vez se hace de *Random Undersampling* (*RUS*) para submuestrear la clase mayoritaria hasta igualar su proporción a la de la clase minoritaria. Este método comprende por tanto, una reducción en el número de instancias usadas en el entrenamiento.

Al igual que se realizó en las anteriores ejecuciones, tras cargar los datos, se aplica el submuestreo de la clase mayoritaria con (*RUS*) sobre los datos de *train*. En esta ocasión, se realiza un balanceo total entre las clases.

Una vez llevado a cabo este submuestreo, se entrenan modelos de *Decision Tree* variando sus parámetros de profundidad máxima (`maxDepth`) y el número máximo de divisiones (`maxBins`), manteniendo el índice *gini* como medida de impuridad. En esta experimentación se repetirán los mismos conjuntos de

valores de parámetros del modelo barridos en la experimentación anterior.

Para acabar, se evaluá el modelo desarrollado sobre el conjunto de *test* con las mismas métricas consideradas en los anteriores experimentos.

Los resultados obtenidos en este conjunto de experimentos se muestran en la siguiente tabla:

		Training		Test	
Max Depth	Max Bins	TPR x TNR	accuracy	TPR x TNR	accuracy
5	10	0.03988	0.72371	0.11009	0.66426
	32	0.03966	0.65787	0.10941	0.66310
	48	0.03960	0.70647	0.10923	0.66115
	64	0.03960	0.65765	0.10926	0.66260
	100	0.03964	0.70991	0.10934	0.66157
	256	0.03972	0.67108	0.10958	0.66270
	500	0.03933	0.70960	0.10832	0.65857
	1000	0.03957	0.71738	0.10896	0.66069
10	10	0.04482	0.71294	0.11980	0.69259
	32	0.04513	0.69431	0.11996	0.69421
	48	0.04539	0.71466	**0.12028**	0.69404
	64	0.04509	0.71037	0.12001	0.69334
	100	0.04530	0.71935	0.12019	0.69359
	256	0.04523	0.71747	0.11950	0.69158
	500	0.04533	0.70059	0.12014	**0.69435**
	1000	0.04538	0.71746	0.11974	0.69233
17	10	0.05462	0.70136	0.10565	0.65131
	32	0.05445	0.70041	0.10906	0.66242
	48	0.05414	0.71102	0.10962	0.66314
	64	0.05460	0.70544	0.10911	0.66203
	100	0.05464	0.70940	0.10940	0.66264
	256	0.05447	0.70854	0.11089	0.66737
	500	0.05467	0.71157	0.10978	0.66378
	1000	0.05471	0.71428	0.11044	0.66564
25	10	0.06035	0.70525	0.09706	0.62421
	32	0.06062	0.70878	0.09938	0.63176
	48	0.06084	0.71243	0.10030	0.63450
	64	**0.06101**	0.71393	0.10053	0.63522
	100	0.06073	0.71116	0.10049	0.63524
	256	0.06093	0.71513	0.10185	0.63941
	500	0.06065	0.71363	0.10200	0.64003

1000	0.06072	**0.71530**	0.10259	0.64182

Tabla 4: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Decission Tree tras aplicar RUS sobre el conjunto de entrenamiento para el balanceo de clases. Se ha remarcado en negrita los valores más altos de cada métrica logrados en este conjunto de experimentos.

Se resume la información del rendimiento $TPRxTNR$ en la siguiente gráfica:

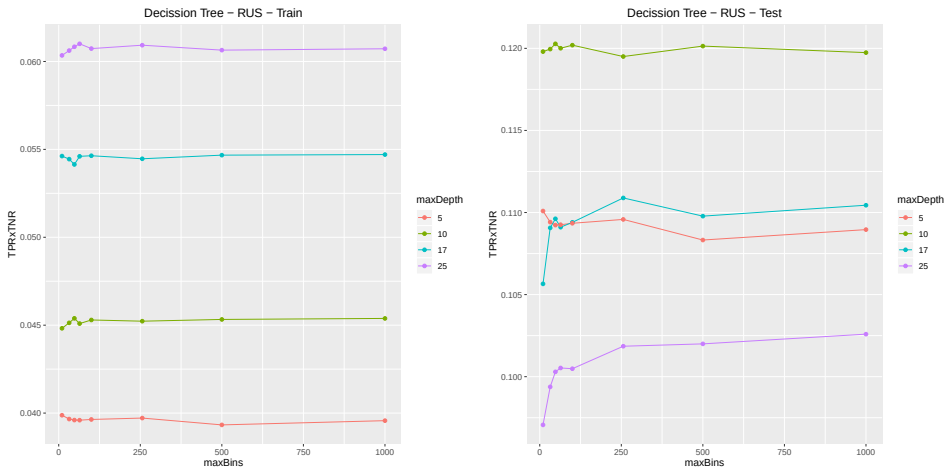


Figura 2: Valores de TPRxTNR obtenidos por los modelos de Decission Tree ante diferentes valores de los parámetros maxBins, maxDepth utilizando RUS para balancear la proporción entre clases. A la izquierda se reflejan los valores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

El modelo óptimo se obtiene considerando, al igual que los anteriores experimentos usando *ROS*, con una profundidad máxima (`maxDepth`) de 10, pero a diferencia del anterior, el rendimiento óptimo se obtiene con un número máximo de divisiones (`maxBins`) de 48. Se comprueba también en las anteriores gráficas que **al incrementar la profundidad del árbol, los modelos comienzan a sobreentrenar**.

Comparamos el mejor modelo obtenido usando (*RUS*) con el mejor modelo obtenido con *ROS*:

			Training		Test	
Método	maxDepth	maxBins	TPR x TNR	Accuracy	TPR x TNR	Accuracy
ROS 100 %	10	1000	0.046085	0.71399	0.12140	0.69742
RUS	10	48	0.04539	0.71466	0.12028	0.69404

Tabla 5: Rendimientos TPRxTNR de los mejores modelos Decission Tree obtenidos con ROS y RUS

Si bien el rendimiento obtenido usando *ROS* como método de balanceamiento de clases es ligeramente superior al obtenido con *RUS*, **los rendimientos de los modelos con ambos métodos de balanceo resultan muy similares**, lo que lleva a plantear, dado que con *RUS* se ha aplicado un fuerte submuestreo de la clase mayoritaria, dando lugar a un subconjunto de las instancias de entrenamiento muy reducido en comparación con el conjunto original, la existencia de una gran redundancia en el conjunto de datos.

2.3. Eliminación de ruido con *ensembler HME BD*

Como última experimentación con *Decission Tree*, se propone probar otro método de preprocesamiento diferente a los métodos de balanceo de clases y se opta por llevar a cabo un filtrado de ruido del *dataset*, de modo que se mejore la calidad de los datos usados durante el entrenamiento y se consiga con ello, aumentar el rendimiento de los clasificadores.

En estas experimentaciones, se hará uso de un filtro de ruido consistente en un *ensembler* homogéneo (*HME BD*) [2].

Los experimentos se ejecutan del siguiente modo: Tras cargar los datos de *train* y *test*, a los datos de *train* le son aplicados este filtro *HME BD*, cuyo *ensembler* se configura con un número determinado de árboles, una profundidad máxima de los árboles y un número de particiones máximas. A continuación, los datos resultantes del preprocesamiento son usados para entrenar el modelo de *Decission Tree*.

Finalmente, al igual que en las dos anteriores experimentaciones, el rendimiento del modelo entrenado es evaluado con el conjunto de *test*.

[2]Enlace al repositorio GitHub con el código y documentación de esta herramienta: https://github.com/djgarcia/NoiseFramework

Debido al alto tiempo que tomaron los experimentos, se realizaron únicamente una serie de experimentos con una configuración única para este filtro de ruido y se realizaron únicamente algunos experimentos con diferentes configuraciones para el clasificador *Decission Tree*. Los resultados de estos experimentos se exponen en la siguiente tabla:

HME BD: Num Trees: 25, Max Bins: 32 y Max Depth: 10

		Training		Test	
maxDepth	maxBins	TPR x TNR	Accuracy	TPR x TNR	Accuracy
10	48	0.00728	0.90446	0.01892	0.50898
10	64	0.00704	0.90424	0.01843	0.50800
17	48	0.00815	**0.90631**	0.01927	0.50967
17	64	**0.00821**	0.90623	**0.01961**	**0.51043**

Tabla 6: Valores de las métricas de rendimiento TPRxTNR y Accuracy medidos sobre el conjunto de train y el de test ante diferentes configuraciones de modelos de Decission Tree y configurando el ensembler homogéneno HME BD con con número de árboles de 25, un número máximo de particiones de 32 y una profundidad máxima de 10

El mejor modelo de *Decission Tree* probado se obtiene considerando una profundidad máxima (`maxDepth`) de 17 y un número máximo de particiones (`maxBins`) de 64.

Comparamos este rendimiento con los mejores modelos obtenidos haciendo uso de los métodos de preprocesamiento de balanceo de clases:

			Training		Test	
Método	maxDepth	maxBins	TPR x TNR	Accuracy	TPR x TNR	Accuracy
ROS 100 %	10	1000	0.04608	0.71399	0.12140	0.69742
RUS	10	48	0.04539	0.71466	0.12028	0.69404
HME BD	17	64	0.00821	0.90623	0.01961	0.51043

Tabla 7: Rendimientos TPRxTNR de los mejores modelos Decission Tree obtenidos con ROS y RUS

El mejor modelo obtenido con *Decission Tree* se ha logrado aplicando balanceo de clases con *ROS*, mientras que el modelo con peores resultados es el que se ha obtenido aplicando como filtro de ruido, el filtro *HME BD*.

Si bien no ha sido posible obtener una cantidad suficiente de experimentos con la aplicación del filtro *HME BD* que permita la extracción de un

mayor número de conclusiones, los resultados nos muestran claramente que **el desbalanceamiento de clases** (presente en esta última experimentación al no haber aplicado ningún preprocesamiento para el balanceo de clases) **supone un inconveniente mayor al aprendizaje de los modelos que la presencia de ruido en el conjunto de datos**.

3. Construción de modelos *Random Forest*

A continuación, se tratarán de ejecutar otro conjunto de experimentos haciendo uso de clasificadores de ***Random Forest*** y repitiendo el mismo conjunto de métodos de preprocesamiento utilizado en la anterior experimentación.

A lo largo de todos los experimentos se han probado diferentes valores para los parámetros de profundidad máxima de los árboles (`maxDepth`), número máximo de divisiones (`maxBins`) y número total de árboles aleatorios generados (`numTrees`).

Debido al elevado tiempo que tomaron los experimentos, el conjunto de valores probados para este conjunto de parámetros se redujo respecto de *Decission Tree* y en algunos experimentos sólo fue posible realizar algunas experimentaciones.

3.1. Sobremuestreo de la clase minoritaria con *ROS*

Los primeros experimentos con modelos basados en *Random Forest*, se prueba a realizar un sobremuestreo simple de la clase minoritaria con *ROS*.

La secuencia completa de ejecución de los experimentos es la que sigue: Tras cargar los datos, se aplica el método *ROS* sobre los datos de entrenamiento para incrementar el número de patrones pertenecientes a la clase minoritaria.

Tras el sobremuestreo, se entrena un modelo de *Random Forest* con diferentes parámetros de `maxDepth`, `maxBins` y `numTrees`.

Debido al enorme coste computacional y, tomando como precedente los resultados obtenidos con los experimentos con *Decission Tree* y *ROS*, se prueba únicamente un sobrebalanceamiento total, es decir, se hace uso de *ROS* con un ratio de sobremuestreo de 1.

El conjunto de parámetros del clasificador *Random Forest* que se probaron fueron los siguientes:

- $maxDepth \in \{5, 10, 17\}$
- $maxBins \in \{32, 64, 100, 256, 500\}$
- $numTrees \in \{10, 25, 50, 100, 500, 1000\}$

A continuación, se exponen los rendimientos obtenidos tanto sobre el conjunto de *train* como sobre el conjunto de *test* para los diferentes valores de `maxDepth` considerados:

- **Considerando un nivel de produndidad máximo de 5**:

			Training		Test
Max Bins	Num Trees	TPR x TNR	Accuracy	TPR x TNR	Accuracy
32	10	0.03959	0.64095	0.10918	0.66404
	25	0.04071	0.63111	0.11219	0.67623
	50	0.04163	0.66126	0.11490	0.68087
	100	0.04198	0.68877	0.11571	0.68106
	500	0.04205	0.68743	0.11597	0.68192
64	10	0.03857	0.58565	0.10656	0.66607
	25	0.04038	0.64762	0.11136	0.67073
	50	0.04181	0.67402	0.11510	0.68028
	100	0.04175	0.68392	0.11507	0.67930
	500	0.04212	0.68055	0.11616	0.68301
100	10	0.04019	0.61497	0.11076	0.67465
	25	0.04100	0.66077	0.11289	0.67432
	50	0.04154	0.66153	0.11426	0.67877
	100	0.04221	0.69070	0.11622	0.68253
	500	0.04206	0.68769	0.11582	0.68148
256	10	0.03822	0.57352	0.10503	0.66441
	25	0.04174	0.69698	0.11475	0.67774
	50	0.04144	0.67741	0.11424	0.67715
	100	0.04182	0.67335	0.11505	0.68016
	500	0.04204	0.67545	0.11576	0.68219
500	10	0.04086	0.65207	0.11219	0.67309
	25	0.04149	0.67488	0.11400	0.67660
	50	**0.04245**	**0.70102**	**0.11679**	**0.68381**
	100	0.04213	0.68434	0.11594	0.68204
	500	0.04203	0.68317	0.11571	0.68142

	10	0.04071	0.64748	0.11202	0.67299
	25	0.04196	0.67826	0.11530	0.68054
1000	50	0.04192	0.69203	0.11526	0.67952
	100	0.04193	0.67665	0.11538	0.68090
	500	0.04217	0.68886	0.11600	0.68196

Tabla 8: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Random Forest tras aplicar ROS sobre el conjunto de entrenamiento para el balanceo de clases y considerando una profundidad máxima de árbol de 5. Se ha remarcado en negrita los valores más altos de cada métrica.

El modelo que ofrece los mejores resultados en todas las métricas tanto en *train* como en *test* es el modelo que considera un número máximo de divisiones de 500 y 50 árboles aleatorios.

En la siguiente gráfica se muestra la evolución de la métrica $TPRxTNR$ frente al número de árboles considerados, reflejándose una evolución para cada valor máximo de divisiones considerados:

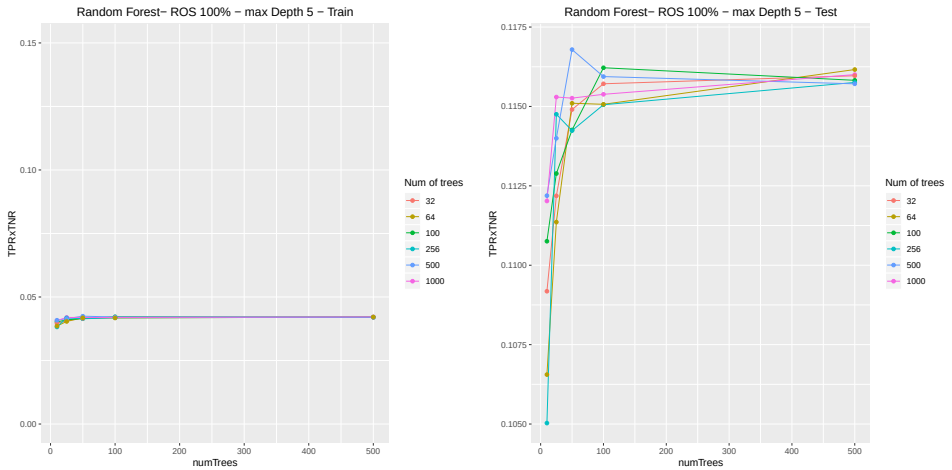


Figura 3: Valores de TPRxTNR obtenidos por los modelos de Random Forest ante diferentes valores de los parámetros maxBins y numTrees, considerando una profundidad máxima de 5 y utilizando ROS para balancear la proporción entre clases con un ratio del 100 %. A la izquierda se reflejan los calores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

La gráfica nos muestra un crecimiento que se asemeja a una función logarítmico del rendimiento del modelo conforme se incrementa el número de árboles, lo cual nos permite comprobar que al incrementar el número de árboles por encima de cierto valor, no se producen ganacias significativas en el rendimiento del modelo y por lo tanto, no interesa incrementar el número de árboles aleatorios.

Por otra parte, también se aprecia para este conjunto de experimentos, que los rendimientos obtenidos sobre el conjunto de *test* es superior al obtenido sobre el conjunto de *train*.

- **Considerando un nivel de produndidad máximo de 10**:

			Training		Test
Max Bins	Num Trees	TPR x TNR	Accuracy	TPR x TNR	Accuracy
32	10	0.04563	0.68547	0.12229	0.70241
	25	0.04668	0.72926	0.12474	0.70665
	50	0.04638	0.72303	0.12412	0.70507
	100	0.04686	0.72133	0.12519	0.70835
	500	0.04694	0.72923	0.12554	0.70899
64	10	0.04610	0.69533	0.12321	0.70444
	25	0.04680	0.72151	0.12485	0.70733
	50	0.04693	0.72650	0.12512	0.70792
	100	0.04704	0.72989	0.12553	0.70895
	500	0.04703	0.72786	0.12547	0.70887
100	10	0.04639	0.70288	0.12331	0.70405
	25	0.04694	0.71525	0.12508	0.70842
	50	0.04702	0.72212	0.12519	0.70835
	100	0.04723	0.72845	0.12569	0.70950
	500	0.04712	0.72943	0.12544	0.70873
256	10	0.04592	0.69934	0.12238	0.70140
	25	0.04701	**0.73414**	0.12469	0.70640
	50	0.04720	0.72060	0.12549	0.70931
	100	0.04727	0.72868	0.12580	0.70981
	500	0.04733	0.72776	**0.12582**	**0.70994**
500	10	0.04659	0.70757	0.12363	0.70460
	25	0.04675	0.72019	0.12441	0.70611
	50	0.04711	0.72393	0.12510	0.70799
	100	0.04722	0.72582	0.12548	0.70902
	500	0.04739	0.73077	0.12580	0.70975
1000	10	0.04599	0.70967	0.12255	0.70112

25	0.04685	0.71265	0.12438	0.70651
50	0.04724	0.72470	0.12545	0.70900
100	0.04736	0.72771	0.12559	0.70929
500	**0.04740**	0.73154	0.12568	0.70939

Tabla 9: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Random Forest tras aplicar ROS sobre el conjunto de entrenamiento para el balanceo de clases y considerando una profundidad máxima de árbol de 10. Se ha remarcado en negrita los valores más altos de cada métrica.

El modelo más óptimo según la métrica $TPRxTNR$ en test, se obtiene con un número máximo de divisiones de 256 y 500 árboles aleatorios.

Se aprecia que, al considerar una profundidad mayor, el modelo presenta una mayor facilidad para sobreentrenar, de esta manera, el modelo que ofrece la mejor métrica $TPRxTNR$ sobre el conjunto de *train*, se obtiene con los valores más altos probados de profundidad máxima y número de árboles.

Al igual que con la anterior experimentación, se representa la evolución de las métricas $TPRxTNR$ medidas sobre el conjunto de *train* y sobre el conjunto de *test* en la siguiente gráfica:

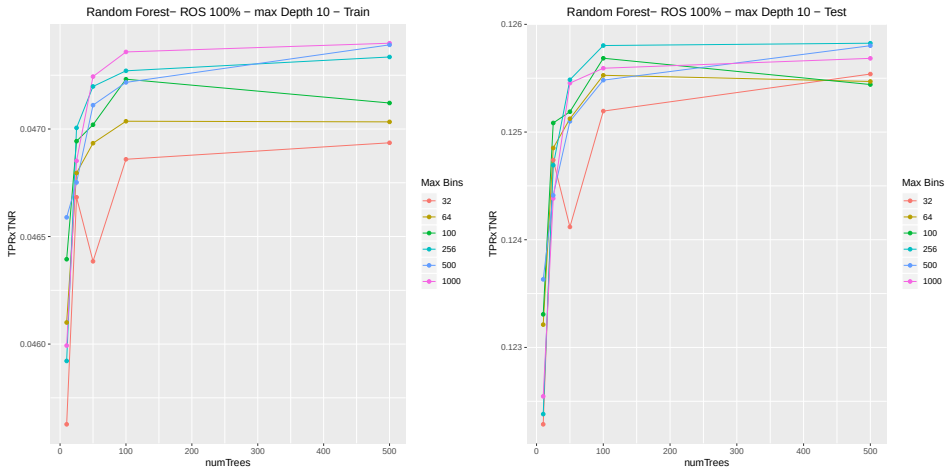


Figura 4: Valores de TPRxTNR obtenidos por los modelos de Random Forest ante diferentes valores de los parámetros maxBins y numTrees, considerando una profundidad máxima de 10 y utilizando ROS para balancear la proporción entre clases con un ratio del 100 %. A la izquierda se reflejan los calores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

En esta gráfica ya queda reflejado de forma clara la tendencia del modelo a sobreajustar a los datos de *train* al incrementar el número de divisiones, tal y como se aprecia en la gráfica de la evolución de $TPRxTNR$ medido sobre el conjunto de *train*, en la que al aumentar este parámetro, se obtienen rendimientos superiores sobre el propio modelo de *train*. De este modo, la gráfica refleja fielmente que, al considerar valores de número de divisiones superiores a 256, el modelo comienza a sobreentrenar.

- **Considerando un nivel de produndidad máximo de 17**:

			Training		Test
Max Bins	Num Trees	TPR x TNR	Accuracy	TPR x TNR	Accuracy
32	25	0.06872	0.84544	0.12382	0.70603
	50	**0.06963**	0.84748	0.12518	0.70952
	100	0.06922	**0.84865**	0.12533	0.71008
64	10	0.06522	0.80666	0.12538	0.70825
	25	0.06860	0.84390	0.12463	0.70793
	50	0.06893	0.84178	0.12630	0.71214
	100	0.06944	0.84775	0.12614	0.71212

100	10	0.06591	0.80874	0.12575	0.70929
	25	0.06828	0.84171	0.12510	0.70900
	50	0.06935	0.84485	0.12602	0.71154
	100	0.06947	0.84787	0.12631	0.71254
256	10	0.06551	0.80544	0.12563	0.70892
	25	0.06903	0.84555	0.12490	0.70876
	50	0.06869	0.84147	**0.12673**	**0.71331**
	100	0.06921	0.84643	0.12653	0.71307
500	10	0.06578	0.80897	0.12621	0.71058
	25	0.06820	0.84121	0.12539	0.70976
	50	0.06874	0.84140	0.12668	0.71315
	100	0.06882	0.84415	0.12653	0.71293
1000	10	0.06591	0.81041	0.12552	0.70867
	25	0.06800	0.84319	0.12481	0.70850
	50	0.06824	0.83952	0.12661	0.71292
	100	0.06884	0.84466	0.12662	0.71321

Tabla 10: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Random Forest tras aplicar ROS sobre el conjunto de entrenamiento para el balanceo de clases y considerando una profundidad máxima de árbol de 17. Se ha remarcado en negrita los valores más altos de cada métrica.

El modelo más óptimo según la métrica $TPRxTNR$ medida sobre test, se obtiene con un número máximo de divisiones de 256 y un número de árboles de 50.

Se representa también la evolución de las métricas de $TPRxTNR$ sobre los conjuntos de *train* y de *test*:

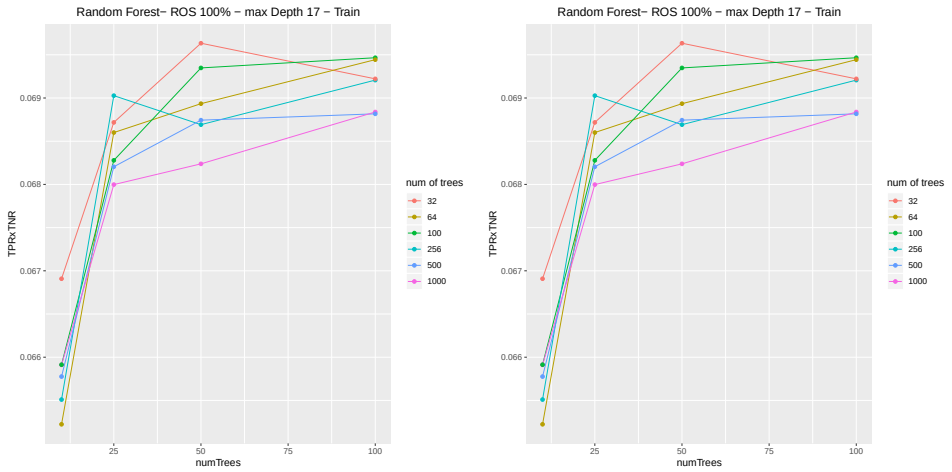


Figura 5: Valores de TPRxTNR obtenidos por los modelos de Random Forest ante diferentes valores de los parámetros maxBins y numTrees, considerando una profundidad máxima de 17 y utilizando ROS para balancear la proporción entre clases con un ratio del 100 %. A la izquierda se reflejan los calores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

> En estas experimentaciones se aprecia una pérdida de rendimiento considerable para número de árboles cercanos a 25 sobre el conjunto de *test*

Por último, comparamos los rendimientos obtenidos para los diferentes parámetros de `maxDepth` y `maxBins` mediante las siguientes gráficas donde se representan los valores medios de $TPRxTNR$ sobre todos los valores de números de árboles para el conjunto de *train* y de *test*:

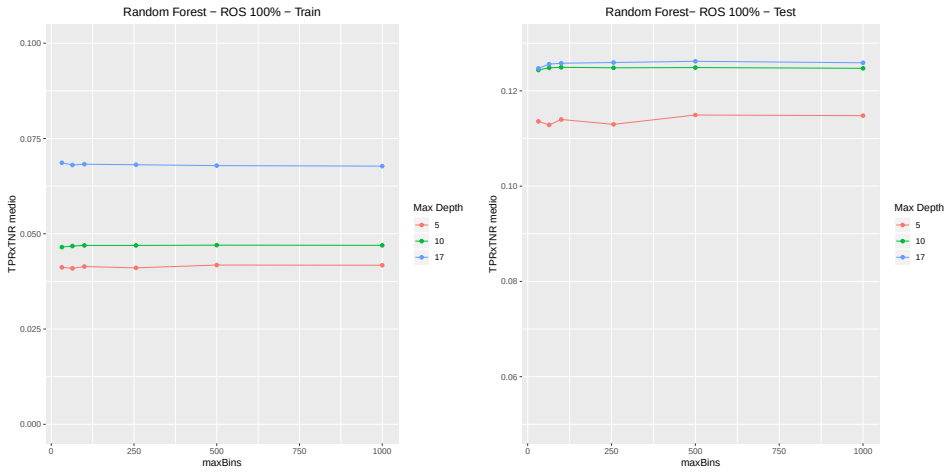


Figura 6: Valores de TPRxTNR medios obtenidos por los modelos de Random Forest ante diferentes valores de los parámetros maxDepth y maxBins de todos los valores de número de árboles considerados en los experiemntos y utilizando ROS para balancear la proporción entre clases con un ratio del 100 %. A la izquierda se reflejan los valores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

Comparamos estos resultados con los obtenidos con *Decission Tree* usando también *ROS* con un sobremuestreo del 100 % sobre el conjunto de *test*:

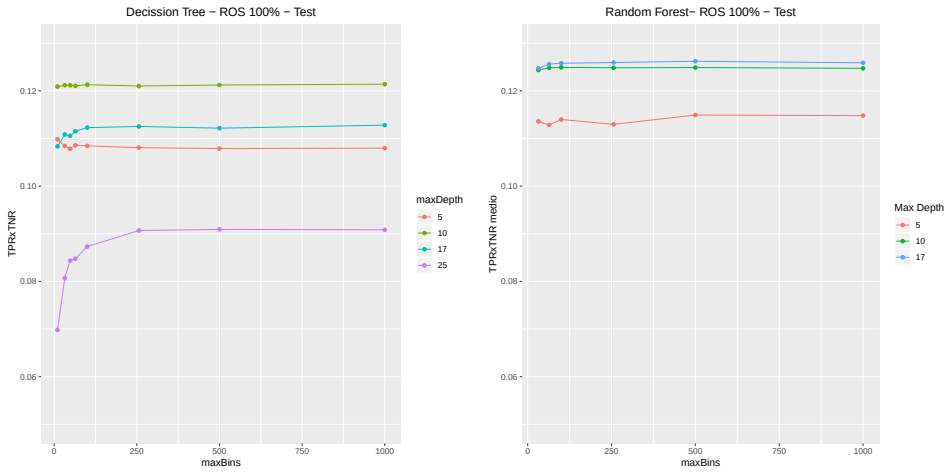


Figura 7: Valores de TPRxTNR obtenidos por los modelos de Decission Tree con un ratio de sobremuestreo igual a 1 (a la izquierda) en comparación con los valores de TPRxTNR medios obtenidos por los modelos de Random Forest usando también ROS con un ratio de 1 y aplicando este cálculo medio sobre los rendimientos obtenidos con diferentes número de árboles (a la derecha).

En la anterior figura se aprecia claramente que, el **rendimiento general obtenido por *Random Forest* es superior al de los modelos *Decission Tree***.

De este modo, el concepto de *bagging* de árboles de decisión explotado por *Random Forest* muestra mejores resultados que los obtenidos por un único modelo de árbol de decisión.

Del conjunto de experimentaciones podemos concluir lo siguiente:

- **El mejor modelo se ha obtenido considerando una produndidad máxima de árbol (`maxDepth`) de 17, un número máximo de divisiones de 256 y un número de árboles de 50**.
- En media y considerando valores altos de `maxDepth`, **los rendimientos muestran sobreentrenamiento al considerar números máximo de divisiones superiores a 256**.
- Los rendimientos de los modelos según la métrica $TPRxTNR$ sobre el conjunto de *test* son superiores a los obtenidos en el conjunto de *train*. No sucede lo mismo para la métrica *accuracy* en la que se muestra el efecto contrario. Teniendo en cuenta que el *accuracy* otorga menos

peso a los aciertos sobre la clase minoritaria mientras que $TPRxTNR$ trata de otorgar un peso similar a ambos, se puede comprobar que estos modelos de *bagging* junto con esta técnica de balanceo permiten obtener mejores resultados en la predición de patrones de la clase minoritaria no usados durante el entrenamiento.

3.2. Submuestreo de la clase mayoritaria con *RUS*

Repetimos las anteriores experimentaciones usando *RUS* como método de balanceo en lugar de *ROS*, con el que se efectúa el submuestreo de la clase mayoritaria y se compararán los rendimientos obtenidos con los obtenidos usando *ROS* como método de balanceo de clases.

Al igual que se realizó en las anteriores ejecuciones, tras cargar los datos, se aplica el submuestreo de la clase mayoritaria con (*RUS*) sobre el conjunto de *train*.

Una vez balanceado el *dataset*, se construyen los diferentes modelos de *Random Forest* utilizando la misma batería de parámetros que se estudió en los experimentos anteriores realizados con *ROS*.

De igual modo que en las anteriores experimentaciones, se muestran los rendimientos obtenidos para los diferentes modelos en las siguientes tablas y gráficos:

Se debe de indicar que, debido al enorme tiempo que tomaron los experimentos, no ha sido posible barrer todas las configuraciones de parámetros, existiendo una enorme carencia de experimentos para una profundidad máxima de 17.

- **Considerando una profundidad máxima de 5**:

		Training		Test	
Max Bins	Num Trees	TPR x TNR	Accuracy	TPR x TNR	Accuracy
32	10	0.03908	0.61231	0.10815	0.66520
	25	0.04209	0.69348	0.11613	0.68214
	50	0.04153	0.66873	0.11456	0.67893
	100	0.04197	0.67860	0.11582	0.68209
	500	0.04225	0.68745	0.11651	0.68364
	1000	0.04202	0.67886	0.11604	0.68277
64	10	0.03985	0.62205	0.10990	0.66988

	25	0.04073	0.69142	0.11229	0.67037
	50	0.04202	0.68304	0.11586	0.68188
	100	0.04162	0.67631	0.11487	0.67924
	500	0.04190	0.68555	0.11566	0.68103
	1000	0.04210	0.68399	0.11606	0.68242
100	10	0.03875	0.60016	0.10697	0.66383
	25	0.04171	0.65242	0.11474	0.68164
	50	0.04185	0.67525	0.11523	0.68056
	100	0.04180	0.66890	0.11523	0.68105
	500	0.04231	0.68983	0.11655	0.68361
	1000	0.04206	0.68603	0.11593	0.68195
256	10	0.04033	0.64744	0.11113	0.66998
	25	0.04189	0.68642	0.11541	0.68027
	50	0.04175	0.65428	0.11521	0.68279
	100	0.04211	0.67597	0.11604	0.68306
	500	0.04233	0.68321	0.11653	0.68399
	1000	0.04228	0.68903	0.11653	0.68358
500	10	0.04132	0.67380	0.11388	0.67627
	25	0.04133	0.69666	0.11389	0.67512
	50	0.04196	0.68827	0.11545	0.68029
	100	**0.04256**	**0.70722**	**0.11707**	**0.68447**
	500	0.04217	0.68539	0.11610	0.68251
	1000	0.04228	0.68790	0.11653	0.68367
1000	10	0.04136	0.66406	0.11388	0.67724
	25	0.04127	0.67152	0.11347	0.67521
	50	0.04187	0.67491	0.11538	0.68104
	100	0.04202	0.67315	0.11569	0.68221
	500	0.04234	0.68881	0.11677	0.68435

Tabla 11: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Random Forest tras aplicar RUS sobre el conjunto de entrenamiento para el balanceo de clases y considerando una profundidad máxima de árbol de 5. Se ha remarcado en negrita los valores más altos de cada métrica.

El mejor modelo obtenido considerando una profundidad máxima de árboles de 5, se obtiene con un número máximo de divisiones (`maxBins`) de 500 y un número de árboles (`numTrees`) de 100.

En la siguiente gráfica se reflejan los resultados expuestos en la tabla 11:

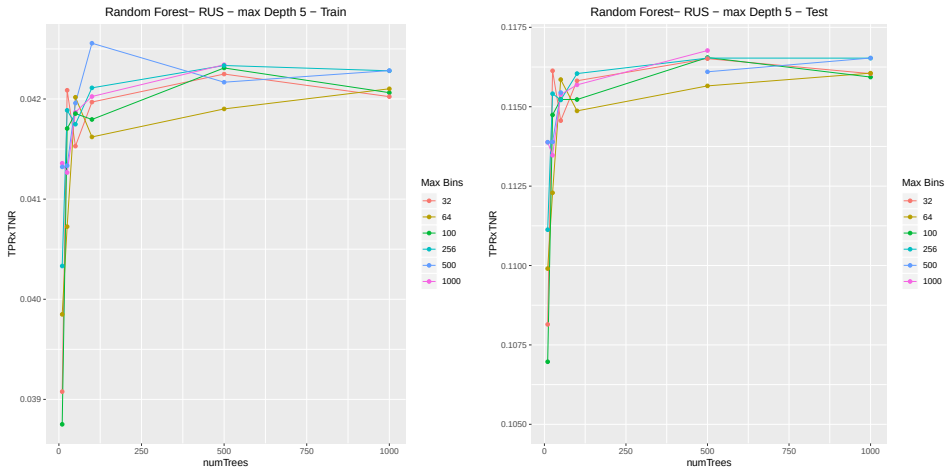


Figura 8: Valores de TPRxTNR obtenidos por los modelos de Random Forest ante diferentes valores de los parámetros maxBins y numTrees, considerando una profundidad máxima de 5 y utilizando RUS para balancear la proporción entre clases. A la izquierda se reflejan los valores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

- **Considerando una profundidad máxima de 10**:

		Training		Test	
Max Bins	Num Trees	TPR x TNR	Accuracy	TPR x TNR	Accuracy
32	10	0.04486	0.67389	0.12085	0.69922
	25	0.04645	0.72531	0.12476	0.70683
	50	0.04653	0.71445	0.12496	0.70801
	100	0.04656	0.71891	0.12513	0.70826
	500	0.04663	0.72738	0.12527	0.70825
	1000	0.04653	0.71932	0.12526	0.70859
64	10	0.04518	0.68160	0.12123	0.69958
	25	0.04642	0.71589	0.12466	0.70705
	50	0.04646	0.71561	0.12478	0.70742
	100	0.04666	0.72084	0.12546	0.70913
	500	0.04676	0.72358	0.12546	0.70900
	1000	0.04677	**0.72921**	0.12550	0.70886
100	10	0.04552	0.69276	0.12202	0.70079
	25	0.04622	0.72438	0.12431	0.70556
	50	0.04646	0.70671	0.12492	0.70855
	100	0.04672	0.72247	0.12523	0.70838
	500	0.04684	0.72289	**0.12561**	**0.70951**

256	10	0.04534	0.68623	0.12169	0.70055
	25	0.04630	0.71545	0.12382	0.70457
	50	0.04675	0.72251	0.12523	0.70840
	100	0.04690	0.72061	0.12554	0.70944
	500	0.04682	0.72433	0.12546	0.70901
500	10	0.04566	0.69985	0.12210	0.70040
	25	0.04628	0.71157	0.12408	0.70558
	50	0.04673	0.72417	0.12523	0.70829
	100	0.04681	0.72157	0.12536	0.70881
	500	**0.04690**	0.72492	0.12551	0.70910
1000	10	0.04511	0.68279	0.12138	0.69980
	25	0.04651	0.72145	0.12453	0.70637
	50	0.04665	0.71493	0.12486	0.70777

Tabla 12: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Random Forest tras aplicar RUS sobre el conjunto de entrenamiento para el balanceo de clases y considerando una profundidad máxima de árbol de 10. Se ha remarcado en negrita los valores más altos de cada métrica.

El modelo que ofrece mejor rendimiento sobre el conjunto de *test* se obtiene con un número máximo de divisiones (`maxBins`) de 100 y un número de árboles de 500 (`numTrees`).

En el siguiente gráfico, se muestra un resumen de los rendimientos de los modelos, medidos según el producto $TPRxTNR$:

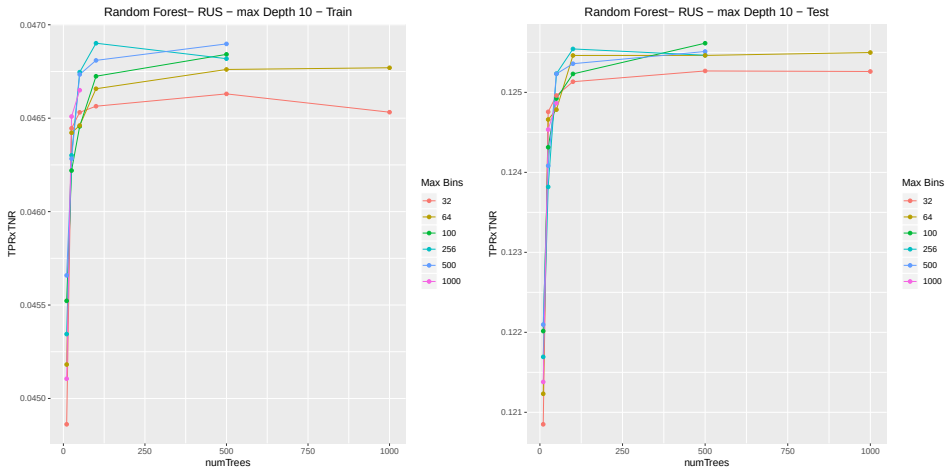


Figura 9: Valores de TPRxTNR obtenidos por los modelos de Random Forest ante diferentes valores de los parámetros maxBins y numTrees, considerando una profundidad máxima de 10 y utilizando RUS para balancear la proporción entre clases. A la izquierda se reflejan los valores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

- **Considerando una profundidad máxima de 17**: Debido al tiempo que tomaron los experimentos, sólo fue posible realizar algunos experimentos:

		Training		Test	
Max Bins	Num Trees	TPR x TNR	Accuracy	TPR x TNR	Accuracy
32	10	0.05562	0.70587	0.12226	0.70408
	25	0.06073	0.77089	0.12760	0.71473
	50	0.06089	0.76131	0.12892	0.71911
	100	**0.06211**	**0.77454**	**0.13031**	**0.72251**

Tabla 13: Valores medidos del producto TPR y TNR y accuracy medidos sobre el conjunto de train y sobre el conjunto de test con Random Forest tras aplicar RUS sobre el conjunto de entrenamiento para el balanceo de clases y considerando una profundidad máxima de árbol de 17. Se ha remarcado en negrita los valores más altos de cada métrica.

El modelo que provee los mejores resultados es aquel obtenido considerando un número máximo de divisiones (`maxBins`) de 32 y un número de árboles (`numTrees`) de 100. No obstante, no se cuenta con un número suficiente de documentos para extraer conclusiones de interés sobre el

rendimiento del modelo ante esta configuración de parámetros, del mismo modo, en la siguiente gráfica se resume el rendimiento $TPRxTNR$ a lo largo de los experimentos:

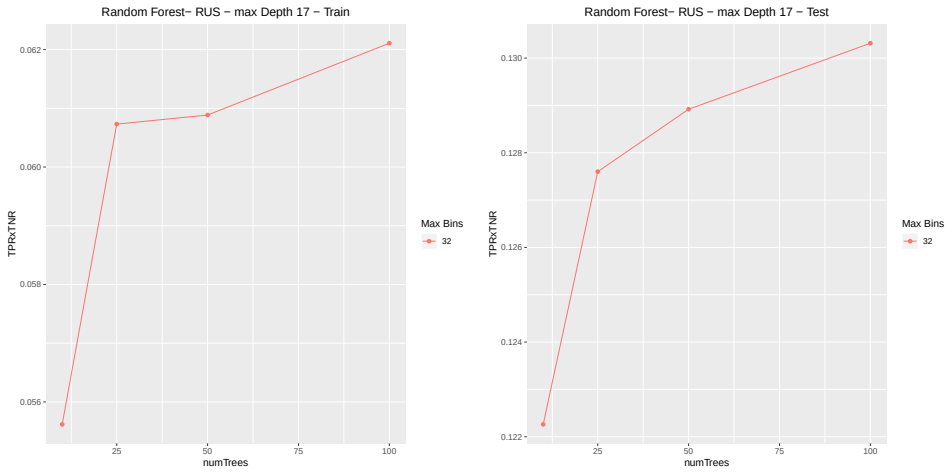
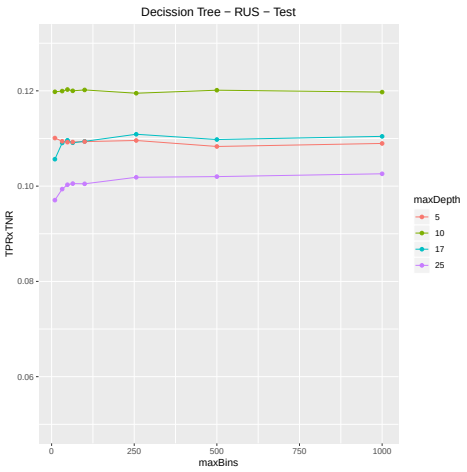


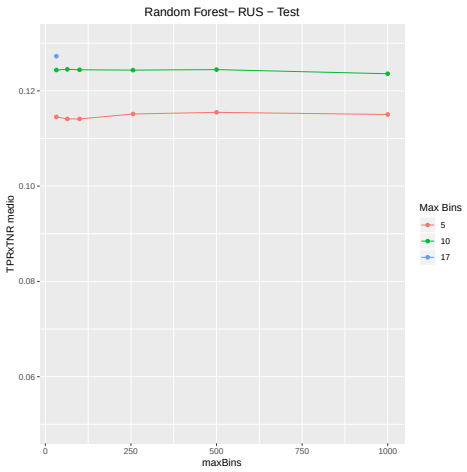
Figura 10: Valores de TPRxTNR obtenidos por los modelos de Random Forest ante diferentes valores de los parámetros maxBins y numTrees, considerando una profundidad máxima de 17 y utilizando RUS para balancear la proporción entre clases. A la izquierda se reflejan los valores de estas métricas sobre el conjunto de train, mientras que a la derecha los valores medidos para el conjunto de test

El modelo que ofrece mejores resultados sobre el conjunto de *test* haciendo uso de *RUS* para el balanceamiento de clases es el último experimento realizado con una profundidad máxima (`maxDepth`) de 17, para el cual se considera un número máximo de divisiones (`maxBins`) de 32 y un número de árboles aleatorios (`numTrees`) de 100.

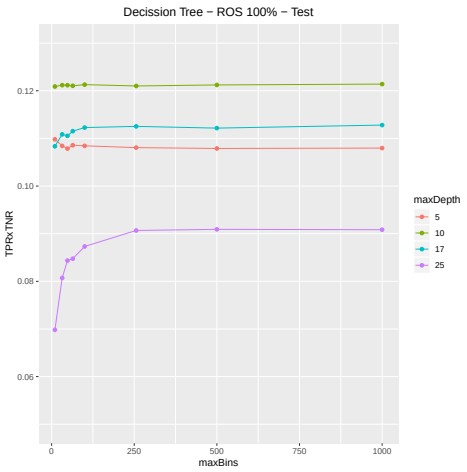
En la siguiente gráfica, se refleja para cada par de valores de `maxDepth` y `maxBins` el rendimiento medio según la métrica $TPRxTNR$ de todas las configuraciones de número de árboles realizadas, sobre los conjuntos de *train* y *test*. A su vez, se comparan con los rendimientos medios obtenidos usando *ROS* como método de balanceo de clases:



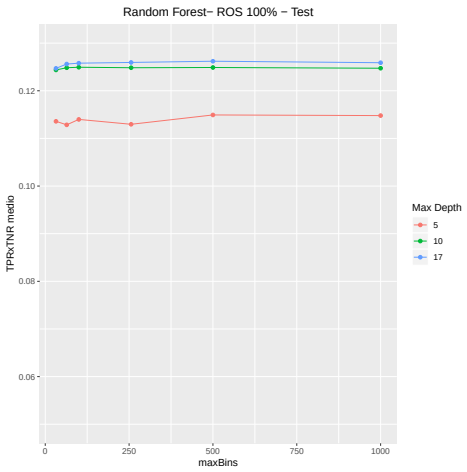
(a) Decission Tree - RUS



(b) Random Forest - RUS



(c) Decission Tree - ROS 100 %



(d) Random Forest - ROS 100 %

Figura 11: Valores de TPRxTNR obtenidos por los modelos de Decission Tree con un ratio de sobremuestreo igual a 1 (a la izquierda) en comparación con los valores de TPRxTNR medios obtenidos por los modelos de Random Forest ante usando también ROS con un ratio de 1 y aplicando este cálculo medio sobre los rendimientos obtenidos con diferentes número de árboles (a la derecha).

Comparamos también los rendimientos de los mejores modelos obtenidos con ambos métodos de preprocesamiento en la siguiente tabla:

Método	maxDepth	maxBins	num Trees	Test TPR x TNR	Test Accuracy
ROS 100 %	17	256	50	0.12673	0.71331
RUS	17	32	100	0.13031	0.72251

Tabla 14: Rendimientos TPRxTNR de los mejores modelos Decission Tree obtenidos con ROS y RUS

La anterior deja entrever, pese al sesgo introducido por la carencia de resultados en esta última experimentación, un claro mejor rendimiento de *Random Forest* con balanceo haciendo uso de *RUS* para una profundidad máxima de 17 frente al rendimiento del *Random Forest* con *ROS*, lo cual lleva a plantear nuevamente la **existencia de redundancia en el *dataset*** y que, en este caso, **llevaría a un peor rendimiento al incrementar esta rendundacia de forma implícita con el sobremuestreo efectuado mediante *ROS***.

3.3. Eliminación de ruido con *ensembler HME BD*

Para acabar, se prueba otra experimentación en la que se emplea nuevamente el filtro *HME BD* para tratar de mejorar el rendimiento de los clasificadores de *Random Tree* con la eliminación de ruido, en lugar de la aplicación de los métodos de balanceamiento de clases *ROS* o *RUS*.

Los experimentos siguiendo el siguiente flujo de operaciones: Tras cargar los datos de *train* y *test*, a los datos de *train* le son aplicados este filtro *HME BD*, cuyo *ensembler* se configura del mismo modo que con los experimentos con *Decission Tree* efectuados con este filtro. A continuación, los datos resultantes del preprocesamiento son usados para entrenar el modelo de *Random Forest*, cuyo rendimiento es evaluado con el conjunto de *test*.

Al igual que sucedió en la experimentación con *Decission Tree* en la que se aplicó este filtro de ruido, sólo ha sido posible efectuar algunos experiementos debido al elevado tiempo tomado por los mismos.

Los resultados de estos experimentos se recogen en la siguiente tabla:

HME BD: Num Trees: 25, Max Bins: 32 y Max Depth: 10

			Train		Test	
maxDepth	maxBins	Num Trees	TPR x TNR	Accuracy	TPR x TNR	Accuracy
5	32	100	$4.39918*10^{-4}$	0.90003	$1.14490*10^{-4}$	0.47111
10	32	50	**0.00271**	**0.90242**	0.00623	0.48374
10	32	100	0.00201	0.90186	0.00450	0.48017
10	32	500	0.00213	0.90195	**0.00475**	**0.48069**

Tabla 15: Valores de las métricas de rendimiento TPRxTNR y Accuracy medidos sobre el conjunto de train y el de test ante diferentes configuraciones de modelos de Random Forest y configurando el ensembler homogéneno HME BD con con número de árboles de 25, un número máximo de particiones de 32 y una profundidad máxima de 10

El mejor modelo *Random Forest* en esta experimentación, se obtiene con una profundidad máxima (`maxDepth`) de 10, un número máximo de divisiones (`maxBins`) de 32 y un número de árboles aleatorios (`numTrees`) de 500.

Comparamos el rendimiento de este modelo con los rendimientos obtenidos sobre *Random Forest* usando las técnicas de balanceo de clases *ROS* y *RUS* y con el mejor modelo de *Decission Tree* que se obtuvo:

					Test	
Clasificador	Método	maxDepth	maxBins	Num Trees	TPR x TNR	Accuracy
Decission Tree	ROS 100 %	10	1000	-	0.12140	0.69742
Decission Tree	HME BD	17	64	-	0.01961	0.51043
Random Forest	ROS 100 %	17	256	50	0.12673	0.71331
Random Forest	RUS	17	32	100	0.13031	0.72251
Random Forest	HME BD	10	32	500	0.00475	0.48069

Tabla 16: Rendimientos TPRxTNR de los mejores modelos Decission Tree obtenidos con ROS y RUS

El mejor modelo obtenido con *Random Forest* se ha obtenido usando *RUS* como método de balanceamiento de clases, de forma que, este modelo supera también el rendimiento del mejor modelo obtenido con *Decission Tree*.

Por su parte, el modelo con el peor rendimiento es el que se ha obtenido en esta última experimentación usando *HME BD* como filtro de ruido, cuyo rendimiento es incluso inferior que el del modelo basado en *Decission Tree* con esta técnica de preprocesamiento.

4. Construcción de modelos *K Nearest Neighbourgh*

Como último modelo, se tratarán modelos basados en el clasificador *lazy K Nearest Neighbourgh* (*kNN*), basado en la implementación efectuada en *KNN IS* [3].

Se ha escogio este tercer clasificadot debido al interés en probar clasificadores con una metodología diferente a la seguida por *Decission Tree* y *Random Forest*, ambos basados en la construción de árboles de decisión cuyos nodos son establecidos minimizando una función de impureza (en este caso el índice *gini*)
Debido al tiempo tomado en la ejecución de los experimentos, se ha barrido únicamente algunos valores del parámetro número de vecinos más cercanos (k), y tampoco se ha evaluado el rendimiento del modelo sobre el propio conjunto de *train* debido al gran tiempo que consumiría realizar dicha evaluación.

4.1. Sobremuestreo de clase minoritaria con *ROS*

Como primer conjunto de experimentaciones con este clasficador, se prueba a realizar un sobremuestreo simple de la clase minoritaria con el método *ROS*.

El flujo de ejecución de los experimentos pasa por una primera fase de **normalización** de los atributos de los *datasets* de *train* y *test* para evitar la predominancia de atributos definidos en un dominio más amplio en el cálculo de las distancias seguido, acto seguido, se aplica *ROS* para balancear las clases y se generaría y evaluaría el modelo *kNN*.

De este modo, tras la carga de los datos de *train* y *test*, se ajusta el estandarizador a los datos de *train* y se normalizan los atributos de los conjuntos de *train* y *test* según las medias y desviaciones típicas aprendidas por el estandarizador. A continuación, a los datos estandarizados del conjunto de *train*, se les aplica el sobremuestreo simple de la clase minoritaria con *ROS* y se construye el clasificador *kNN* con este conjunto de datos sobremuestreado considerando diferentes configuraciones del número de vecinos más cercanos (k).

[3]Enlace al repositorio GitHub con el código fuente y la documentación del clasificador KNN IS: https://github.com/JMailloH/kNN_IS

Finalmente, el modelo desarrollado se evalúa con el conjunto de *test* y los resultados de esta evaluación son almacenados.

A continuación, se exponen los resultados de los experimentos realizados considerando diferentes diferentes valores de k. Para todos los experimentos se ha considerado un ratio de sobremuestreo de 1:

k	TPR x TNR Test	Accuracy Test
1	0.04631	0.53572
3	**0.06807**	**0.56679**

Tabla 17: Rendimientos TPRxTNR del modelo obtenido con KNN tras aplicar estandarización sobre el conjunto de datos y ROS con un ratio de sobremuestreo del 100 % sobre el conjunto de train

El mejor modelo se ha obtenido considerando 3 vecinos más cercanos en la clasificación de los patrones.

Comparamos este rendimiento con los obtenidos en las anteriores experimentaciones con modelos basados en *Decission Tree* y *Random Forest* usando también *ROS* como técnica de balanceo de clases:

					Test	
Clasificador	Método	maxDepth	maxBins	Num Trees	TPR x TNR	Accuracy
Decission Tree	ROS 100 %	10	1000	-	0.12140	0.69742
Random Forest	ROS 100 %	17	256	50	0.12673	0.71331
3-NN	ROS 100 %	-	-	-	0.06807	0.56679

Tabla 18: Rendimientos TPRxTNR de los mejores modelos Decission Tree, Random Forest y kNN obtenidos con ROS

De esta comparación se aprecia que *Random Forest* provee los mejores resultados sobre el conjunto de *test*, mientras que *kNN* es el modelo que ofrece los peores resultados.

4.2. Submuestreo de clase mayoritaria con *RUS*

Se repite la anterior experimentación aplicando nuevamente *RUS* para el balanceo de clases en lugar de *ROS*, de modo que el flujo de ejecución queda de la siguiente forma:

Tras la carga de los datos de *train* y *test*, se ajusta el estandarizador a los datos de *train* y se normalizan los atributos de los conjuntos de *train* y *test* según las medias y desviaciones típicas aprendidas por el estandarizador. A continuación, a los datos estandarizados del conjunto de *train*, se les aplica el submuestreo simple de la clase mayoritaria con *RUS* y se construye el clasificador *kNN* con este conjunto de datos submuestreado considerando diferentes configuraciones del número de vecinos más cercanos (k).

Finalmente, el modelo desarrollado se evalúa con el conjunto de *test* y los resultados de esta evaluación son almacenados.

Los resultados de los experimentos se muestran en la siguiente tabla:

k	TPR x TNR Test	Accuracy Test
1	0.08919	0.59917
3	0.09536	0.62033
5	**0.09854**	**0.63085**

Tabla 19: Rendimientos TPRxTNR del modelo obtenido con KNN tras aplicar estandarización sobre el conjunto de datos y RUS sobre el conjunto de train

El mejor modelo se obtiene considerando $k = 5$.

Comparando estos rendimientos obtenidos por los modelos *kNN* usando esta técnica de balanceo de clases con los obtenidos usando *ROS* para el balanceo de clases, se observa que el rendimiento obtenido con *RUS* supera al rendimiento obtenido con *RUS*.

De este modo se comprueba que, primeramente **no es necesario emplear un número tan elevado de instancias para obtener rendimientos similares** y, en segundo lugar, que **la introdución de instancias efectuada por *ROS* introduciría más ruido en el conjunto de datos que llevaría a un empeoramiento en los rendimientos de los modelos**.

Por último, se compara también el rendimiento obtenido por los modelos *kNN* usando *RUS* con los rendimientos obtenidos por *Decission Tree* y *Random Forest* usando esta técnica de preprocesamiento:

	Train		Test	
Clasificador	TPR x TNR	Accuracy	TPR x TNR	Accuracy
Decission Tree	0.04539	0.71466	0.12028	0.69404
Random Forest	0.06211	0.77454	0.13031	0.72251
1-NN	-	-	0.08919	0.59917

Tabla 20: Rendimientos TPRxTNR de los mejores modelos Decission Tree, Random Forest y kNN obtenidos con ROS

Del mismo modo que en la anterior experimentación con *ROS*, *Random Forest* provee mejores resultados, mientras que *kNN* ofrece los peores resultados.

4.3. Eliminación de ruido con *ensembler HME BD*

Por último, se realiza una última experimentación en la que se aplica el *ensembler* homogéneo *HME BD* para el filtrado de ruido en lugar de tratar un balanceo de las clases.

El flujo de ejecución de los experimentos resultaría como sigue: Tras la carga de los datos de *train* y *test*, se ajusta el estandarizador a los datos de *train* y se normalizan los atributos de los conjuntos de *train* y *test* según las medias y desviaciones típicas aprendidas por el estandarizador. A continuación, a los datos estandarizados del conjunto de *train*, se les aplica el filtrado de ruido con el *ensembler* homogéneo, el cual ha sido configurado con la misma configuración fija que se utilizó en las experimentaciones con los anteriores modelos de *Decission Tree* y *Random Forest*.

De este modo, se construye el clasificador *kNN* con este conjunto de datos preprocesado y se procedería a evaluar el clasificador el conjunto de *test* almacenando las métricas asociadas a esta evaluación.

Los resultados de esta experimentación, se exponen en la siguiente tabla:

k	TPR x TNR Test	Accuracy Test
1	0.01026	0.48900
3	0.00418	0.47909

Tabla 21: Rendimientos TPRxTNR del modelo obtenido con KNN tras aplicar estandarización sobre el conjunto de datos y filtrado de ruido con HME BD sobre el conjunto de train

Con la aplicación del filtro de ruido *HME BD*, el rendimiento de los modelos *kNN* empeora al incrementar el número de vecinos más cercanos para la clasificación de nuevos patrones de 1 a 3, el mejor modelo en esta experimentación, se obtiene por tanto, para $k = 1$.

Comparamos estos resultados con los obtenidos haciendo uso de las técnicas de balanceo de clases *ROS* y *RUS*:

Preprocesamiento	k	TPR x TNR Test	Accuracy Test
ROS 100 %	1	0.04631	0.53572
ROS 100 %	3	0.06807	0.56679
RUS	1	0.08919	0.59917
RUS	3	0.09536	0.62033
RUS	5	**0.09854**	**0.63085**
HME BD	1	0.01026	0.48899
HME BD	3	0.00418	0.47909

Tabla 22: Rendimientos TPRxTNR de los modelos óptimos obtenidos con KNN tras aplicar diferentes técnicas de preprocesamiento

En la anterior comparación, se aprecia que *RUS* permite obtener los mejores rendimientos, mientras que el filtrado con *HME BD* provee los peores rendimientos para los clasificadores *kNN*.

El hecho de que el filtrado de ruido con *HME BD* provea peores resultados que el balanceo de clases con *ROS*, lleva a reafirmar que el desbalanceamiento de clases presente en el *dataset* constituye el elemento que más sesgo introduce a los clasificadores.

Para terminar, se comparan los rendimientos de los mejores modelos obtenidos haciendo uso de cada uno de los métodos de preprocesamiento evaluados en este proyecto:

Clasificador	Preprocesamiento	Parámetros	TPR x TNR Test	Accuracy Test
Decission Tree	ROS 100 %	maxDepth:10 maxBins:1000	0.12140	0.69742
Decission Tree	RUS	maxDepth:10, maxBins: 48	0.12028	0.69404
Decission Tree	HME BD	maxDepth:17 maxBins:64	0.01961	0.51043
Random Forest	ROS 100 %	maxDepth: 17, maxBins: 256, numTrees:50	0.12673	0.71331
Random Forest	RUS	maxDepth:17, maxBins: 32, numTrees:100	0.13031	0.72251
Random Forest	HME BD	maxDepth:10, maxBins: 32, numTrees:500	0.00475	0.48069
kNN	ROS 100 %	k=3	0.06807	0.56679
kNN	RUS	k=5	0.09854	0.63085
kNN	HME BD	k=1	0.01026	0.48899

Tabla 23: Rendimientos TPRxTNR de los mejores modelos Decission Tree obtenidos con ROS y RUS

El mejor modelo elaborado en todo el conjunto de experimentaciones realizados en este proyecto, se obtiene con *Random Forest* haciendo uso de submuestreo aleatorio simple (*RUS*) de la clase mayoritaria para el balanceo de clases, configurado con una profundidad máxima de 17, un número máximo de divisiones de 32 y un número de árboles de 100.

Los peores rendimientos se han obtenido usando el filtro de ruido de *HME BD* en lugar de métodos de balanceo de clase, lo que lleva a afirmar nuevamente que este desbalanceamiento de clases afecta en mayor medida al rendimiento de los modelos. Por su parte, el clasificador mejor se ha beneficiado del filtrado con *HME BD* es *Decission Tree*