




Memoria competición en Kaggle

Harvard-Cete

Jaume Cloquell Capo
José A. Fernández Sánchez
Antonio R. Moya Martín-Castaño
Nuria Rodríguez Barroso



1. Introducción

En este primer capítulo de la memoria veremos en detalle la introducción al problema que se pedía resolver como parte de la evaluación práctica de la asignatura Minería de Datos: Pre-procesamiento y clasificación, enmarcado dentro del Máster en Ciencia de Datos de la Universidad de Granada.

1.1. Reparto de tareas

Para empezar, resumimos en Tabla 1 cómo hemos repartido las tareas entre los miembros del equipo, que básicamente se ha reducido a dividir los 4 algoritmos entre los mismos. Además de esto se creó un repositorio privado en Github para trabajar de forma colaborativa.

| Miembro | Algoritmo |
|------------------------|---------------------|
| Jaume Cloquell | Árboles de decisión |
| José Alberto Fernández | KNN |
| Antonio R. Moya | RIPPER |
| Nuria Rodríguez | SVM |

Tabla 1: Reparto de algoritmos

2. EDA

El problema propuesto para su resolución es un problema de clasificación que debería abordarse usando técnicas de pre-procesado y cuatro algoritmos de clasificación de diversas vertientes (Árboles de clasificación, SVM, RIPPER y KNN). El grueso del problema reside en el proceso de pre-procesado de datos ya que los algoritmos que veremos no admiten muchas modificaciones en sus parámetros.

En este capítulo veremos el proceso seguido para afrontar y resolver el problema definido en puntos anteriores. El capítulo comienza detallando el proceso exploratorio inicial para continuar con el grueso de la memoria, la especificación de los procesos de pre-procesado llevados a cabo y la solución que mejor se adapta en todos los algoritmos.

2.1. Tipos de datos y dimensiones

El conjunto de datos de entrenamiento está formado por 9144 observaciones con 50 variables además de la clase catalogada como C y que puede tomar dos valores distintos (0, 1). Las variables de entrenamiento, tienen todos valores numéricos continuos.

2.2. Distribuciones de las Variables

Inicialmente sin realizar una imputación a los datos, ni filtrar el ruido, se puede observar que todas las variables tienen un número bastante elevado de outliers y valores muy extremos tanto en el extremo inferior como superior de los datos.

Sin embargo, tras haber realizado una imputación de missing values, filtrado el ruido y habiendo tratado los outliers, obtenemos distribuciones en las variables como las siguientes.

Si miramos el boxplot conjunto de todas las variables normalizadas encontramos lo siguiente

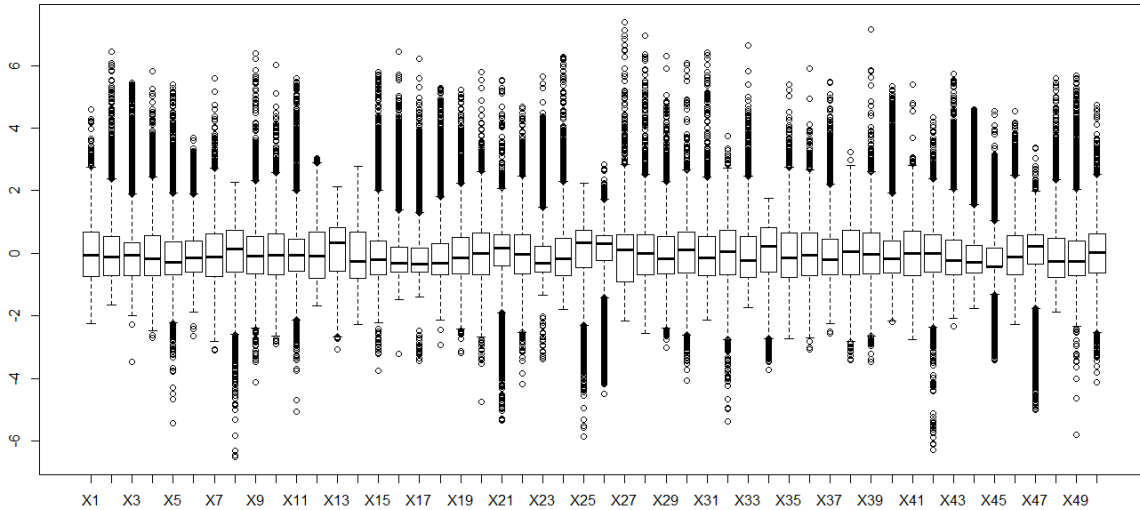


Figura 1: Distribución de algunas variables

Donde observamos que una vez tratado el ruido y los outliers no encontramos demasiada discrepancia entre cómo se distribuyen los outliers en las variables.

Si vamos a un análisis uno a uno de las distribuciones de las variables podemos encontrar distribuciones como las siguientes.

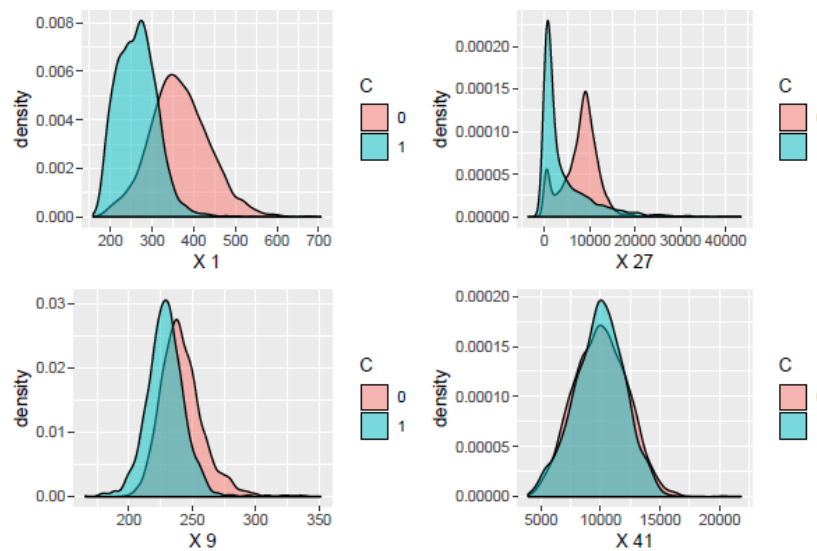


Figura 2: Distribución de algunas variables

Con este procedimiento pudimos ver como algunas variables parecen separar bien las dos clases, en este ejemplo $X1$ y $X27$, mientras que otras parecen ofrecer muy poca información en primera instancia. Además las distribuciones sirvieron para comprobar el grado de simetría y de curtosis de las variables (de manera visual). Información que puede ser útil a la hora de plantearse hacer algunas transformaciones sobre las variables.

2.3. Missing Values

Al estudiar las distribuciones de los datos en el punto anterior descubrimos la existencia de valores perdidos en todas las variables. Para ver si este problema era muy acentuado se creó una función que nos ofrece el número de valores perdidos de un dataset por variables con diversos estadísticos. Tras obtener estos valores se representaron gráficamente para ver cuantos eran estos valores perdidos en función de la variable y el conjunto de train (figura 3.1).

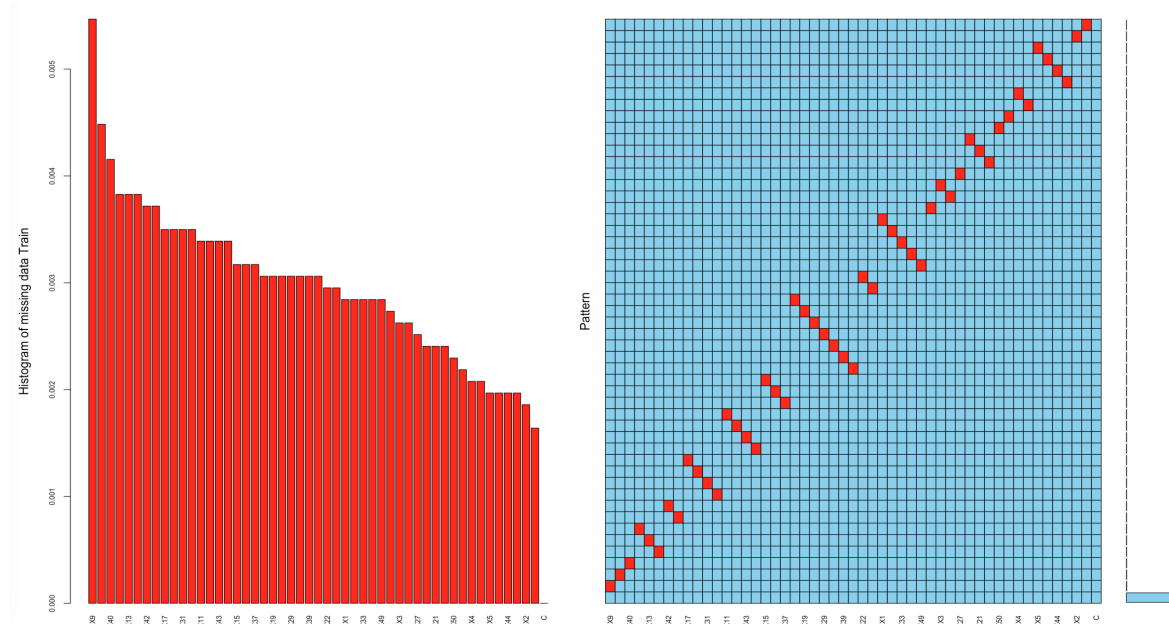


Figura 3: Distribución de los Missing Values

Este gráfico nos llevo a comprobar que los valores perdidos no siguen patrones sino que son valores perdidos que parecen haber sido añadidos aleatoriamente o pertenecer a fallos en la toma de datos. La variable $X9$, es la que tiene un mayor número de missing values, seguidos por $X40$ y $X13$. A pesar de todo, no existe una diferencia significativa entre ellos.

2.4. Outliers

Echando un vistazo inicial al dataset realizando una imputación de los valores perdidos y normalizando encontramos lo siguiente.

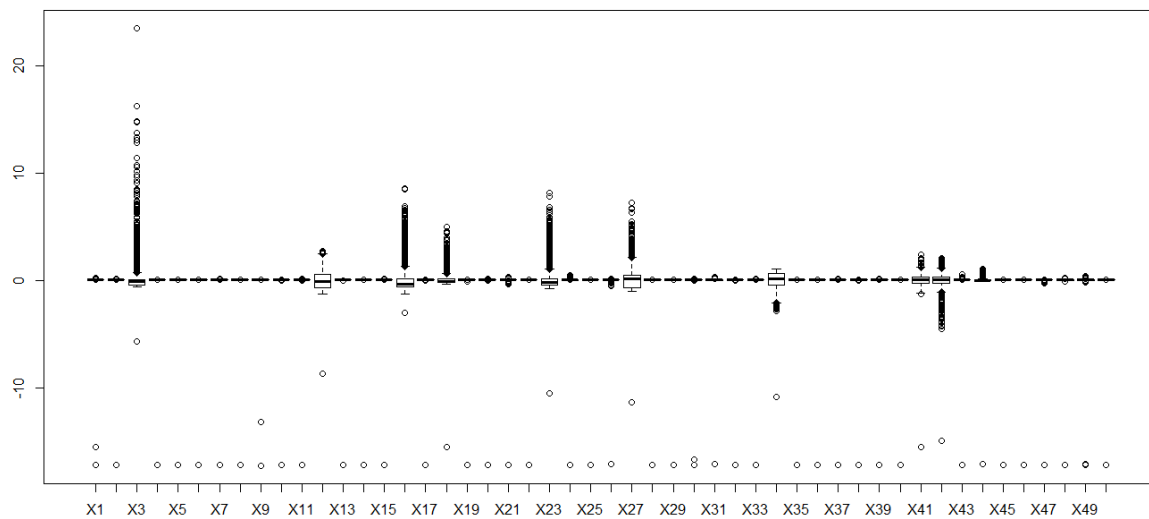


Figura 4: Boxplot Inicial

Lo que se observa es que hay valores que resultan ser tan extremos, que no nos permiten ver cómo se distribuyen realmente las variables. Además desvían en gran medida la media y la varianza, lo cual es también un problema.

En definitiva parece evidente la presencia de outliers en el dataset, los cuales podemos distinguir en: outliers univariantes y multivariantes. Las siguientes gráficas muestran un biplot utilizando como coordenadas las dos componentes principales de los datos, y marcando los valores anómalos en cada caso.

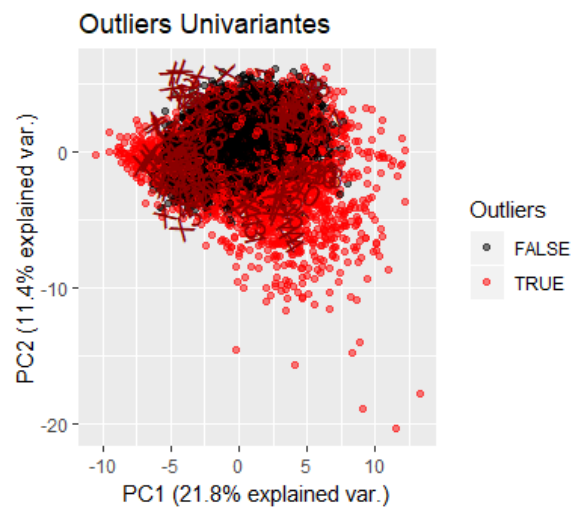


Figura 5: Outliers Univariantes

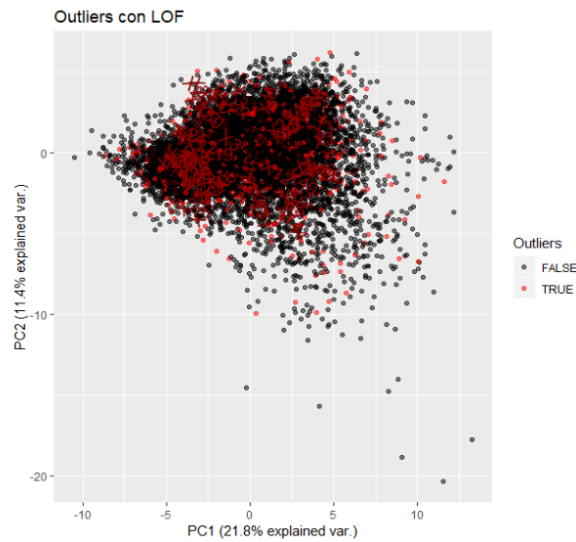


Figura 6: Outliers Multivariantes con LoF score

En el caso de outliers univariantes se detectan todos aquellos que se salgan del intervalo intercuartílico más 3 veces el rango intercuartílico. En el caso de los outliers multivariantes mostramos los 1000 con peor lof score. Éste método determina como outliers aquellos puntos que tienen una densidad muy baja con respecto a la densidad que tienen sus vecinos. Como podemos comprobar los criterios son distintos y los resultados también. Se escogen 1000 outliers estimando que pueda existir esa cantidad de outlier multivariantes, pero realmente es una cantidad arbitraria, con lo cual no llega a ser del todo fiable que existan tantos o tan pocos outliers multivariantes.

2.5. Correlaciones

Otro aspecto a observar es la correlación entre variables. De este modo, podemos sacar información útil con vistas a combinaciones de variables, eliminar algunas de ellas (muy correlacionadas) o encontrar variables muy correlacionadas con la clase. Tengamos en cuenta que para observar esta correlación, inicialmente se deben eliminar aquellos datos que contengan valores perdidos en algunas de sus variables.

Observemos el resultado de realizar un **corrplot** tras eliminar los datos con valores perdidos:

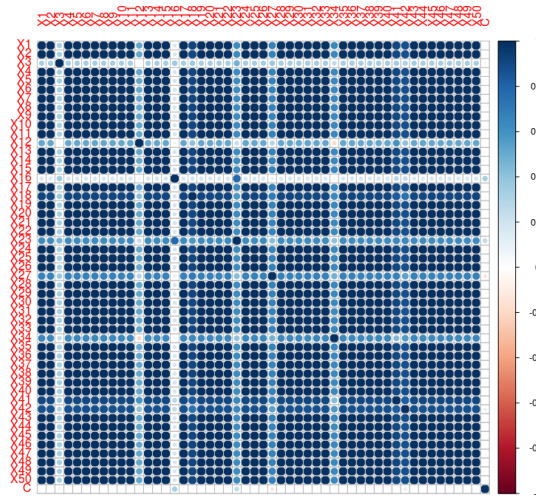


Figura 7: Corrplot sobre conjunto sin datos con valores perdidos

Atendiendo a este corrplot, parece que hay muchas variables muy correlacionadas entre ellas, a la par que pocas muy relacionadas con la clase. No parece que se saque información muy útil de este corrplot.

Sería recomendable obtener el corrplot de nuestro conjunto de datos una vez ya se hayan imputado los valores perdidos de manera adecuada (esta imputación se explica en un apartado posterior) y se haya eliminado ruido. Así pues, tras imputar los datos y eliminar ruido, el corrplot obtenido es el siguiente:

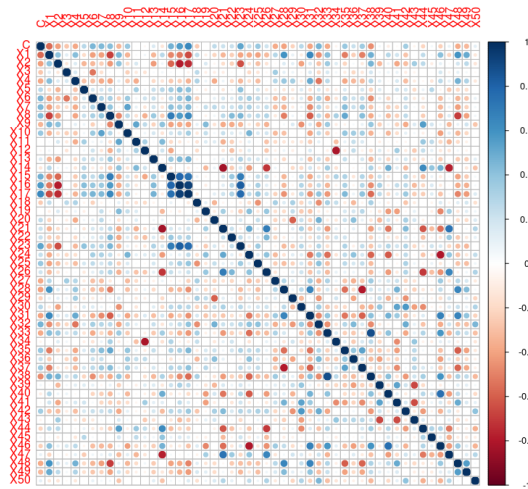


Figura 8: Corrplot sobre conjunto con valores perdidos imputados y ruido eliminado

Podemos observar como de este corrplot si se puede sacar información algo más útil. Hay variables que parecen muy correlacionadas, como podrían ser X2 y X16; X2 y X17; X21 y X14, etc. Dentro de estas correlaciones, cabe destacar X15, X16 y X17 están muy correlacionadas.

También podemos observar como algunas variables están correlacionadas con la clase, como podrían ser X_1 , X_{17} o X_{23} .

2.6. Distribución clases

Por último, en nuestro proceso de análisis exploratorio, se realizó un gráfico de distribución de variables para comprobar si estamos ante un problema de clases balanceadas o en su defecto no balanceadas. El resultado puede verse en la figura 2, donde queda constatado que estamos ante un problema donde la clase 0 y la 1 están en clara desventaja por lo que podría ser interesante usar técnicas de oversampling o undersampling.

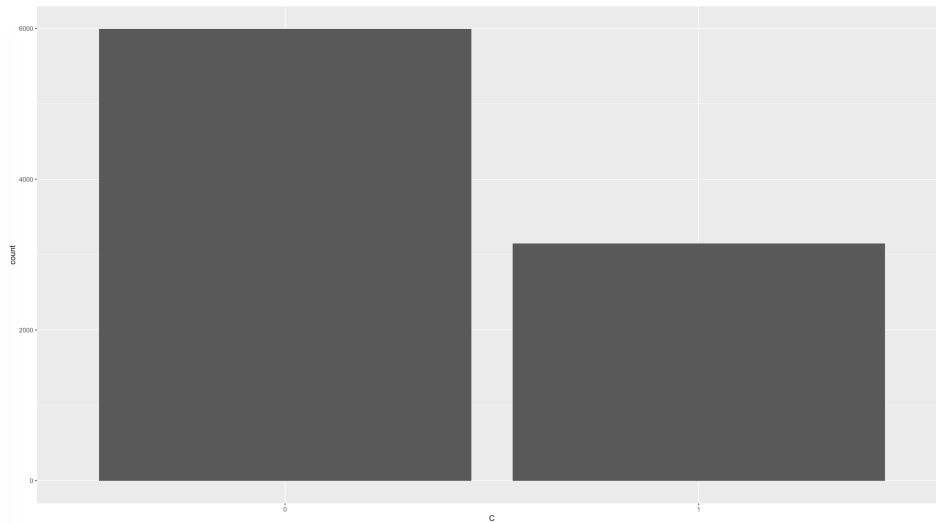


Figura 9: Distribución de clases

3. Preprocesamiento

3.1. Selección de características

Como puede darse el caso que alguna o muchas variables usadas en el modelo no estén asociadas con la variable respuesta, incluir variables irrelevantes añade complejidad innecesaria al modelo. Por lo que eliminar dichas variables puede mejorar la interpretabilidad de los resultados, minimizar el sobre-aprendizaje, reducir la dimensionalidad de los datos y el tiempo de aprendizaje de los modelos construidos.

Para la selección de caraterística se han desarrollado metodologías de los tres enfoques propuestos en la asignatura:

3.1.1. Filter feature selectors

Para la implementación de estas funciones de selección de caraterísticas se ha utilizado el paquete *FSelector*. De entre todas las que incluye, las utilizadas han sido:

- **Chi Cuadrado:** determina la importancia de los atributos realizando un test de independencia Chi Cuadrado entre las variables y la etiqueta.
- **Basados en medidas de correlación:** estos métodos miden la correlación entre las variables y la variable objetivo haciendo un ranking de las X variables más correladas con la etiqueta. Se han utilizado tanto la correlación lineal de Pearson como la de Spearman.
- **Basados en entropía:** mide la ganancia/pérdida de información al seleccionar variables. Se implementaron *information.gain*, *gain.ration* y *symmetrical.uncertainly*.
- **Relief:** este método se basa en medir la distancia a los vecinos k vecinos más cercanos. Si los vecinos son de su misma clase, la importancia decrece (no aporta mucha información extra a la otra variable) y viceversa.

En general, los algoritmos de aproximación o filtros descritos se basan en establecer un ranking entre las variables en función de su correlación con la variable objetivo, información aportada, etcétera. Así, determinando el número de variables a las que queremos reducir los datos, basta con reducir a las X primeras.

3.1.2. Wrappers

Este enfoque consiste en ir seleccionando subconjuntos de características evaluando la calidad del subconjunto entrenando un modelo. De esta forma, se seleccionará el subconjunto que mejor resultado haya proporcionado.

Para evaluar los subconjuntos se utiliza **rpart**, construyendo árboles de clasificación que midan la calidad del subconjunto seleccionado. Para ello, se define una función *evaluator* de la misma forma que se explicó en la parte de Preprocesamiento de la asignatura, la cual será un parámetro para la función del paquete FSelector de R.

Se han implementado las siguientes modalidades:

- `best.first.search`
- `exhaustive.search`
- `forward.search`

3.1.3. Embedded

Este enfoque trata de aprovechar las ventajas de los dos enfoques anteriores. Calcula la importancia o peso de las características evaluando sobre un modelo. En este caso el modelo se trata de un **randomForest**. Los nodos del modelo se analizan (como en enfoque 1) para medir la información de importancia de cada variable.

3.2. Imputación

El subconjunto de train del dataset con el que trabajamos presenta muchos missing values, como ya hemos observado en la parte de EDA. Para solventar estos problemas estudiamos los siguientes métodos de imputación.

- Media, mediana, moda: este método consiste en imputar los valores perdidos utilizando los valores de la media, mediana o moda de esa variable. En train podemos incluso distinguir entre clases para obtener una imputación algo mejor.
- Paquete 'Amelia': este paquete ofrece la función `amelia` que realiza un proceso iterativo basado en bootstrap con el que reemplaza los missing values con valores extraídos de las distribuciones a posteriori de las variables. Dicha función admite que se le introduzca un parámetro para especificar conocimientos a priori de las variables.
- Paquete 'randomForest': este paquete ofrece la función `rflmpute` que imputa los valores perdidos con una media ponderada del resto de observaciones. Los pesos de esa media ponderada vienen dados por la proximidad determinada por el Random Forest.
- Paquete 'mice': este paquete ofrece la función `mice` que imputa cada columna utilizando el resto de columnas como regresoras y como método para hacer regresión pmm, regresión logística o polinomial, entre otras.
- Paquete 'robCompositions': el paquete ofrece la función `impKNNa` que implementa distintos métodos de imputación basados todos en knn.

3.3. Filtrado de Ruido y Tratamiento de Outliers

Como hemos observado en el apartado EDA, hemos obtenido evidencias de la existencia de outliers en el dataset de train, que pueden estar relacionados con errores en los instrumentos de medición, en las entradas de datos, etc ... Como medida para solucionar el problema de la presencia de ruido en los datos hemos recurrido a la función IPF (filtro de particionamiento iterativo).

El IPF propone una técnica elimina los datos ruidosos de forma iterativa utilizando varios clasificadores construidos con el mismo algoritmo de aprendizaje. IPF elimina los ejemplos ruidosos en múltiples iteraciones hasta que el porcentaje de las instancias consideradas como outliers está por debajo de un umbral.

Para el tratamiento de los outliers lo que funcionó bien fue realizar un proceso iterativo en el que se reemplazaban valores que constitúan outliers univariantes en cada columna por NA's. Posteriormente se volvía a realizar una imputación sobre todo el dataset. Este proceso pareció dar buenos resultados cuando se iteraba de 2 a 3 veces.

El criterio para detectar outliers univariantes fue tomar como referencia el intervalo entre los cuartiles primero y tercero más un coeficiente por el rango intercuartílico. Este coeficiente varió entre 1,5, 3 y 5.

Este proceso se aplicó a test, pero con ciertas modificaciones, ya que para imputar los valores perdidos solo se podía hacer uso del conjunto de train y esto añadía cierta dificultad a la hora de implementar el método.

3.4. Transformaciones

Dentro del preprocesamiento puede interesar llevar a cabo algunas transformaciones de las variables. La idea sería que todas las variables siguieran una distribución normal (o al menos, lo más cercana a una distribución normal posible). A continuación se indican algunas variables que han sido transformadas,

mostrando la operación realizada sobre las mismas y su función de densidad antes y después de la transformación (el uso o no de estas transformaciones dependerá de los resultados obtenidos con las mismas, ya que el uso de las mismas no garantiza mejores resultados):

- $X1$: Se aplica logaritmo:

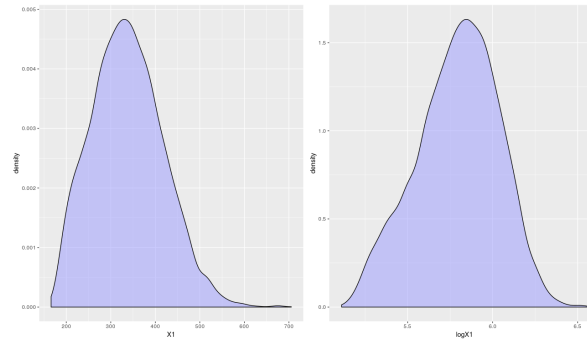


Figura 10: Funciones de densidad variable 1

- $X2$: Se aplica la raíz cuadrada:

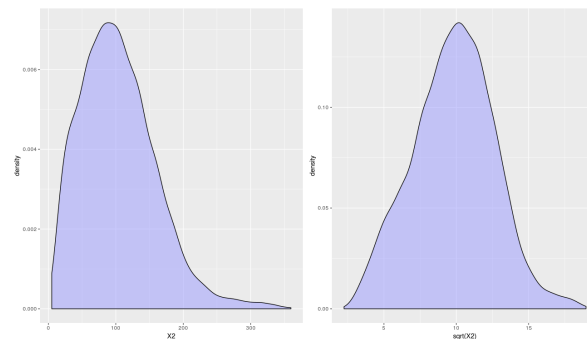


Figura 11: Funciones de densidad variable 2

- $X3$: Se aplica el logaritmo. En esta variable era complicado encontrar una buena transformación, con lo que vemos que no se obtienen grandes resultados:

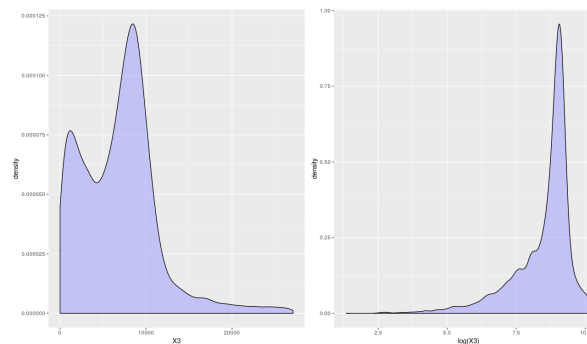


Figura 12: Funciones de densidad variable 3

- X_4 : Se aplica el logaritmo:

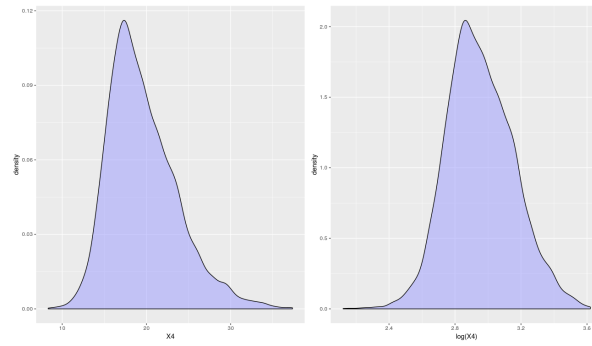


Figura 13: Funciones de densidad variable 4

- X_8 : Se elevan los datos a 3.5:

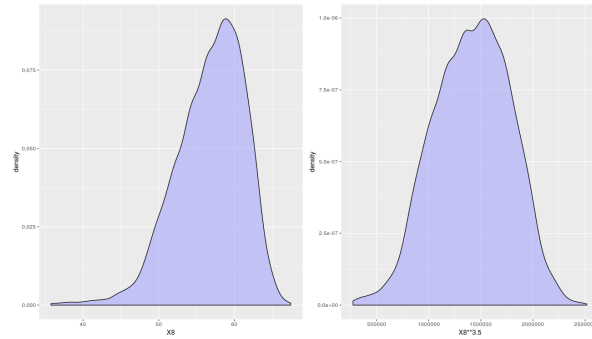


Figura 14: Funciones de densidad variable 8

- X_{10} : Se eleva a 2.5 el logaritmo de los datos:

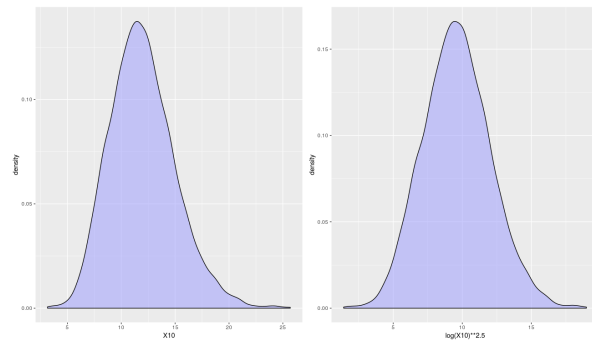


Figura 15: Funciones de densidad variable 10

- X_{19} : Se eleva a 0.75:

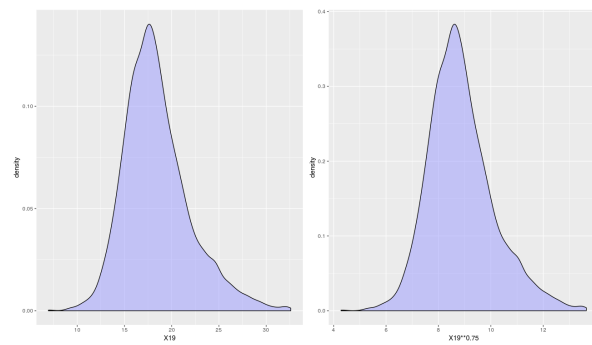


Figura 16: Funciones de densidad variable 19

- X_{24} : Se eleva a 0.23:

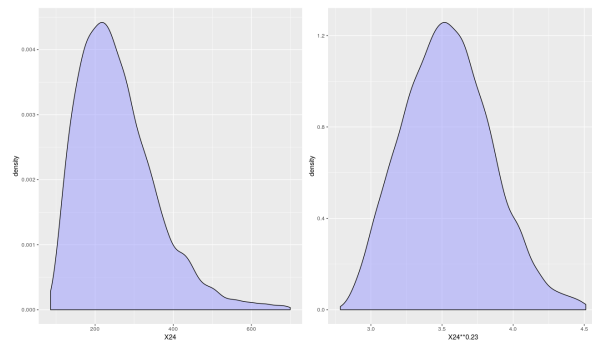


Figura 17: Funciones de densidad variable 24

- X_{28} : Se aplica el logaritmo:

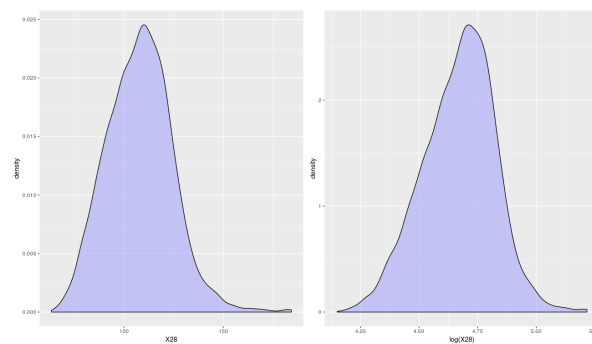


Figura 18: Funciones de densidad variable 28

- X_{31} : Se aplica el logaritmo:

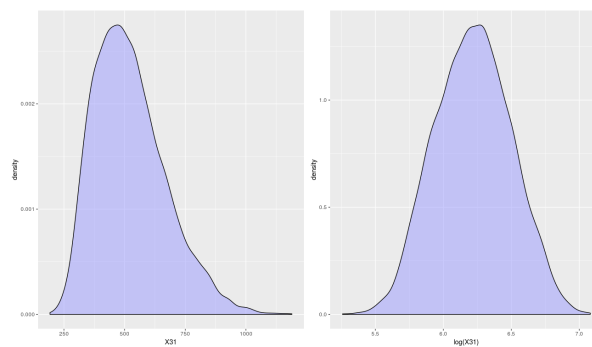


Figura 19: Funciones de densidad variable 31

- X33: Se aplica la raíz cuadrada:

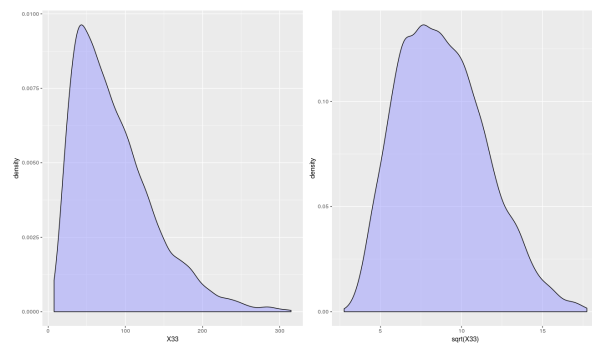


Figura 20: Funciones de densidad variable 33

- X40: Eleva a 0.75:

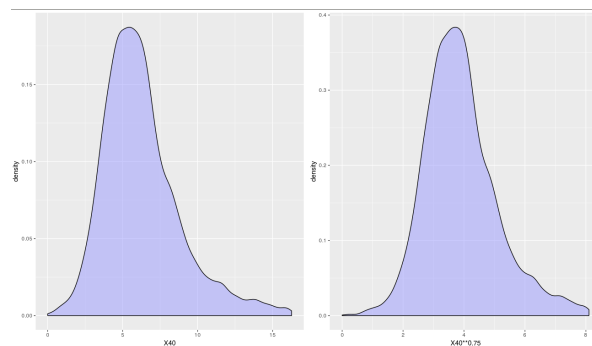


Figura 21: Funciones de densidad variable 40

- X44: Se aplica la raíz cuadrada:

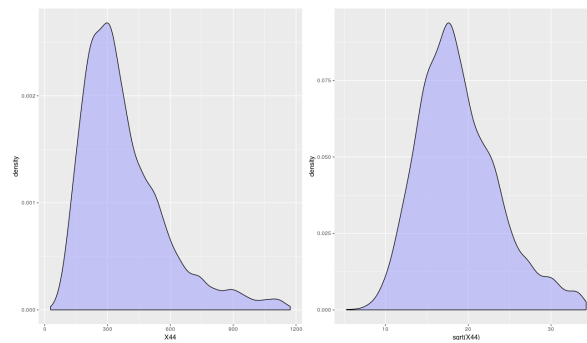


Figura 22: Funciones de densidad variable 44

3.5. Balanceo de clases

En la Figura 9 podemos apreciar el desbalanceo de clases de los datos proporcionados. Este desbalanceo aumenta al aplicar algunos de los métodos de filtrado de ruido (Figura 23). Añadiendo estos hechos a la sensibilidad de algunos de los clasificadores elegidos al desbalanceo de las clases, se hace crucial el tratamiento de este problema.

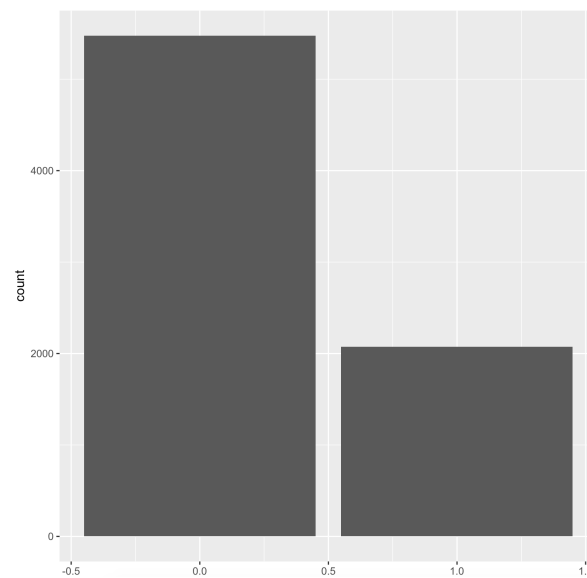


Figura 23: Distribución de clases tras preprocesamiento.

Durante el desarrollo de los métodos de balanceo de datos utilizaremos la siguiente notación:

1. **Clase mayoritaria:** aquella a la cual pertenecen la mayoría de los datos. En nuestro caso la clase 0.
2. **Clase minoritaria:** aquella a la cual pertenece la menor parte de los datos. En nuestro caso la clase 1.

El problema del balanceo de datos se puede abordar desde dos enfoques diferentes:

- **Oversampling:** consiste en aumentar el número de instancias pertenecientes a la clase minoritaria, consiguiendo así un balanceo entre ambas clases y aumentando el número de datos total.
- **Undersampling:** consiste en disminuir el número de instancias de la clase mayoritaria consiguiendo el mismo objetivo, pero disminuyendo el número de datos disponibles.

En este trabajo se han utilizado métodos de balanceo de ambas categorías.

3.5.1. Oversampling - Smote

El **SMOTE**, de sus siglas en inglés *Synthetic Minority Over-sampling Technique* es una de las técnicas más utilizadas para oversampling, motivo por el cual decidimos elegirlo.

El funcionamiento consiste básicamente en encontrar aquellas regiones de datos de la clase mayoritaria donde se encuentren instancias de la clase minoritaria. El problema de estas regiones es que los modelos tienden a prescindir de la información de la clase minoritaria. Así, si aumentamos la presencia de datos minoritarios en esta zona, influirán más en el modelo pues no prescindirá de dicha información.

Esto causará efectos en el modelo, por ejemplo en los árboles de clasificación se producirán nuevas particiones a partir de la información añadida.

(bibliografía -> <https://arxiv.org/pdf/1106.1813.pdf>)

3.5.2. Undersampling - Tome K Links

La idea básica de esta metodología es básicamente muy parecida, acabar con la “supremacía” de la clase mayoritaria en aquellas zonas donde se encuentran instancias de la clase minoritaria. A diferencia de SMOTE, se eligen instancias de la clase mayoritaria que serán eliminadas. Para la elección de estas instancias, se eligen aquellas que estén más próximas a instancias de la clase minoritaria, pues son aquellas que más eclipsan la información que estas instancias aportan.

Por la naturaleza de las metodologías utilizadas para solucionar el desbalanceo tanto por oversampling como undersampling, se espera un buen funcionamiento especialmente en los **árboles de clasificación**.

4. Modelos

4.1. KNN

El modelo knn, es un modelo de clasificación que consiste en clasificar una instancia en función de la clase a la que pertenezcan sus k vecinos más cercanos. Donde radica la complejidad del método es en determinar ese concepto de cercanía, que dependerá de la métrica utilizada.

En nuestro caso después de haber hecho el EDA inicial, nos preocupamos por hacer una subida de resultados únicamente habiendo hecho imputación con random forest, un filtrado de ruido utilizado IPF y una normalización de los datos. Utilizando esta puntuación como base empezamos a preocuparnos por los distintos aspectos que comentaremos uno a uno.

1. **Imputación:** en este apartado utilizamos directamente métodos más avanzados que los de moda, mediana o media que resultan bastante básicos. En primera instancia empezamos utilizando una imputación por random forest, pero dado el tiempo que necesitaba cada ejecución, nos decantamos por acabar utilizando la imputación ofrecida por el paquete 'Amelia' con los parámetros por defecto.
2. **Filtrado de ruido:** aquí utilizamos siempre IPF. El resto de funciones que filtraban ruido resultaron ser demasiado costosas a nivel computacional, por lo que nos decantamos por esta opción. Siendo que mejoraba considerablemente los resultados y tenía un tiempo de ejecución razonable.
3. **Corrección de outliers:** como se ha descrito previamente utilizamos un método para sustituir outliers univariantes por valores NA (en el atributo del que era outlier), para después imputar los datos nuevamente. Este método podía plantearse como un proceso iterativo, ya que cuanto mejor fueran los datos que corregíamos, mejores correcciones podíamos seguir haciendo. Sin embargo, observamos que iterar demasiadas veces este proceso podía resultar en deshacernos de outliers que, en el caso de knn al menos, eran valiosos a la hora de clasificar nuevos ejemplos. Por lo tanto terminamos iterando este proceso de 2 a 3 veces en las últimas pruebas. Además el método para detectar outliers univariantes era determinar como tales a los valores que se saliesen del intervalo intercuartílico más un coeficiente por el rango intercuartílico. Este coeficiente empezó siendo 3, se probó con 1,5 pero perdíamos mucha información, y acabó siendo 5 para tratar de eliminar solo valores verdaderamente extremos.
4. **Selección de variables:** se intentó utilizar random forest para determinar el peso de las variables y se siguieron dos caminos. El primero sin demasiado éxito en test, pero con leves mejoras en train fue eliminar variables cuyo peso no superaran un umbral, que determinamos en función de cómo se distribuían estos pesos. El segundo fue determinar previamente el número de variables con las que nos quedaríamos y dar el corte independientemente de cómo se distribuyeran los pesos. Esta segunda idea tuvo mayor éxito siendo que la selección de 40 características mejoró el accuracy en el test público. Se probó con 35 y 30 atributos y los resultados empeoraron. También se probó a utilizar del orden de 10 a 20 variables, y los resultados empeoraban.
5. **Interacciones y transformaciones de variables:** en base a las gráficas de las funciones de densidad se pensó en hacer transformaciones logarítmicas de algunas variables con una alta curtosis. Además introducimos interacciones polinómicas entre las variables con mayor peso en el apartado anterior. Ambas estrategias no dieron muchos resultados, pero esto sucedió en la recta final de la competición y debido a la escasez de tiempo no se hizo de manera exhaustiva. Siendo este punto quizás donde se podría haber encontrado una mejora significativa del modelo, si se le hubiera dedicado más tiempo.
6. **Balanceo de clases:** dado que existía un pequeño desbalanceo entre clases y que al filtrar el ruido con IPF, resultaba que el desbalanceo se pronunciaba más. Se utilizó SMOTE sobre el conjunto de training con el ratio de desbalanceo inicial, pero no se tuvo demasiado éxito. El modelo knn si bien es cierto que puede verse afectado por el desbalanceo, cuando éste no es tan pronunciado, no tiene por qué ser el caso. A no ser que se escojan valores muy alto de k.
7. **Elección de k:** inicialmente siempre que se ejecutaba el modelo con una cross validation, se obtenían los mejores resultados para valores de k elevados, del orden de 20 a 30. Estos valores demasiado altos no funcionaban tan bien en test, y al utilizar valores demasiado bajos tampoco se mejoraba enormemente la situación. Finalmente comprobamos que los valores más robustos, cuando

probábamos distintos métodos tanto en train como posteriormente observando los resultados en test, eran aquellos entorno al $k = 11$. Por lo que la mayoría de las subidas se hicieron con $k \in \{9, 11, 13\}$. Valores muy altos y muy bajos de k siempre fracasaban en cuanto a accuracy.

8. **Uso de otras métricas:** se planteó utilizar tanto distancias de la literatura, como la distancia de Minkowski (con $1 \leq p \leq \infty$), como estas mismas distancias pero dando más peso a algunas variables utilizando la información que ofrecían algunos selectores de variables. En el primer caso los experimentos realizados no resultaron demasiado satisfactorios, y en el segundo no se encontró una manera satisfactoria de utilizar estos pesos sin que desequilibrase en exceso la importancia de las variables (ya que los valores de los pesos que daban los selectores eran demasiado heterogéneos y se buscaba algo más sutil).

En la siguiente tabla se muestran los resultados obtenidos en train (con 10-cv), test público y test privado, con algunas de las estrategias ideadas y expuestas anteriormente.

| Método | Train | Test_pub | Test_priv |
|--------------------------------------|-----------|----------|-----------|
| RfImpute | 0.763744 | 0.71056 | 0.71326 |
| RfImpute+IP+corrección | 0.9208487 | 0.90658 | 0.91326 |
| RfImpute+IPF+corrección train y test | 0.9208487 | 0.91015 | 0.91173 |
| * + selección | 0.9252549 | 0.90250 | 0.91275 |
| * + selección Xi.sq | 0.9215437 | 0.90301 | 0.90561 |
| * + smote ratio 0.4 | 0.9230338 | 0.90556 | 0.91224 |
| * + smote ratio 0.5 | 0.9210779 | 0.90250 | 0.90663 |
| * + smote ratio 0.6 | 0.920816 | 0.90148 | 0.90051 |
| * + rfselector 40 atributos | 0.9353580 | 0.91424 | 0.90459 |

- Selección Xi.sq es la selección de atributos utilizando el test Ξ^2 .
- Donde "*" indica el preprocesamiento de aplicar imputación por Amelia, filtrado de ruido por IPF y corrección de outliers con 3 iteraciones y con el coeficiente de detección de outliers por IQR en 5. Además de aplicar la corrección de outliers en test con imputación por knn.
- En la última subida con la selección de atributos por random forest, se corrigen primero los outliers y luego se aplica el filtrado IPF.

En vista de los resultados obtenidos llegamos a la conclusión de que la corrección de outliers y el filtrado de ruido mejoraban considerablemente el rendimiento del modelo. Además la imputación por Amelia resultó ser muy eficiente en términos de resultados y tiempo. Con respecto a la selección de atributos, quitar atributos no mejora los resultados demasiado y en ocasiones los empeoraba. Sin embargo, si se busca un modelo con una complejidad menor o se quiere reducir el tiempo de computación no es mala idea, porque tampoco empeora el rendimiento considerablemente.

4.2. Árboles de decisión

Los árboles de decisión son un método usado en distintas disciplinas como modelo de predicción. Estos son similares a diagramas de flujo, en los que llegamos a puntos en los que se toman decisiones de acuerdo a una regla.

En el campo del aprendizaje automático, hay distintas maneras de obtener árboles de decisión, la que usaremos en esta ocasión es conocida como CART: Classification And Regression Trees. Esta es una

técnica de aprendizaje supervisado. Tenemos una variable objetivo (C) y nuestra meta es obtener una función que nos permita predecir, a partir de variables predictoras (independientes), el valor de la variable objetivo para casos desconocidos.

La implementación particular de CART que usaremos es conocida como Recursive Partitioning and Regression Trees o RPART. De allí el nombre del paquete que utilizaremos en nuestro ejemplo.

De manera general, lo que hace este algoritmo es encontrar la variable independiente que mejor separa nuestros datos en grupos, que corresponden con las categorías de la variable objetivo. Esta mejor separación es expresada con una regla. A cada regla corresponde un nodo.

4.2.1. Preprocesamiento

Lo primero que se realizó con el algoritmo r-part fue ejecutarlo directamente sin realizar ningún preprocesamiento sobre los datos. Con esto se obtuvo un acierto de 0.84948. Esto nos sirvió de referencia para saber que cualquier preparación de los datos que se haga antes de pasarlos por el algoritmo rpart debe de mejorar este resultado.

Empezando ya con la parte de preprocesamiento propiamente dicha, se empezaron probando filtros básicos. Concretamente primero se aplicó el filtro de ruido **IPF**, por su coste computacional bajo. Esto supuso una mejora respecto al anterior resultado con un valor del 0.86275. Tras esto, se aplicó distintas técnicas de imputación de los valores perdidos con **Random Forest**, **Amelia** y **KNN**. Random Forest fue el que aportaba mejores resultados que junto con **IPF** se obtuvo un resultado 0.87704.

Aún así, después nos dimos cuenta de que para que terminarse de imputar había que hacer la imputación dos veces. Como se ha descrito en los otros modelos, la idea era imputar los valores perdidos con **Random Forest**, sustituir por NA los valores de datos considerados outliers univariantes e imputar de nuevo y para acabar, eliminar ruido con **IPF**. Haciendo doble imputación con Random Forest junto con **IPF** se obtuvo el mejor resultado 0.89081.

Sobre estos pasos que han conducido al mejor resultado se pensó aplicar técnicas de **oversampling y undersampling**. Aunque los datos no fueron del todo malos (0.88265) generaban bastante sobreajuste al modelo y no mejoraba los del mejor preprocesado hecho hasta el momento. Seguidamente sobre el preprocesamiento que hasta ahora había dado los mejores resultados se probaron diversos métodos de selección de características. Al igual que con el modelo SVM, en primer lugar, mediante **chi cuadrado** se probaron con 40, 35 y 30 características. Se llegó a la conclusión que al reducir demasiado el número de variables estábamos perdiendo precisión en el modelo. Además, a diferencia al modelo SVM, no aportó en ningún caso mejoras en el modelo. Luego se utilizó **random forest**, aunque este dio resultados bastantes similares a los obtenidos (0.88877) no mejoraban al mejor modelo.

4.2.2. Evaluando el modelo

Del entrenamiento de nuestro modelo obtenemos el siguiente resultado.

```
1) root 7471 1979 0 (0.735109088 0.264890912)
  2) X17< 40.6823 6036 759 0 (0.874254473 0.125745527)
    4) X32< 88.88057 4857 221 0 (0.954498662 0.045501338)
      8) X6< 26.82724 4778 161 0 (0.966303893 0.033696107)
        16) X1>=293.2635 4118 40 0 (0.990286547 0.009713453)
          32) X23< 11051.76 4027 26 0 (0.993543581 0.006456419)
            64) X41>=4667.684 4019 23 0 (0.994277183 0.005722817)
              128) X32< 87.16321 3946 15 0 (0.996198682 0.003801318)
                256) X27< 20092.43 3935 13 0 (0.996696315 0.003303685)
                  512) X6< 15.34434 3730 4 0 (0.998927614 0.001072386) *
                    1024) X6>=15.34434 205 9 0 (0.956097561 0.043902439)
                      2048) X8< 59.44896 174 1 0 (0.994252874 0.005747126) *
                        4096) X8>=59.44896 31 8 0 (0.741935484 0.258064516)
                          8192) X22< 26.20092 24 3 0 (0.875000000 0.125000000) *
                            16384) X22>=26.20092 7 2 1 (0.285714286 0.714285714) *
                              32768) X27>=20092.43 11 2 0 (0.818181818 0.181818182) *
                                65536) X32>=87.16321 73 8 0 (0.890410959 0.109589041)
                                  131072) X30< 23.86427 66 3 0 (0.954545455 0.045454545) *
                                    262144) X30>=23.86427 7 2 1 (0.285714286 0.714285714) *
```

Figura 24: Esquema de nuestro árbol de clasificación

Lo anterior muestra el esquema de nuestro árbol de clasificación. Cada inciso nos indica un nodo y la regla de clasificación que le corresponde. Siguiendo estos nodos, podemos llegar a las hojas del árbol, que corresponde a la clasificación de nuestros datos.

Todo lo anterior resulta mucho más claro si lo visualizamos, aunque no es suficiente por el tamaño del árbol.

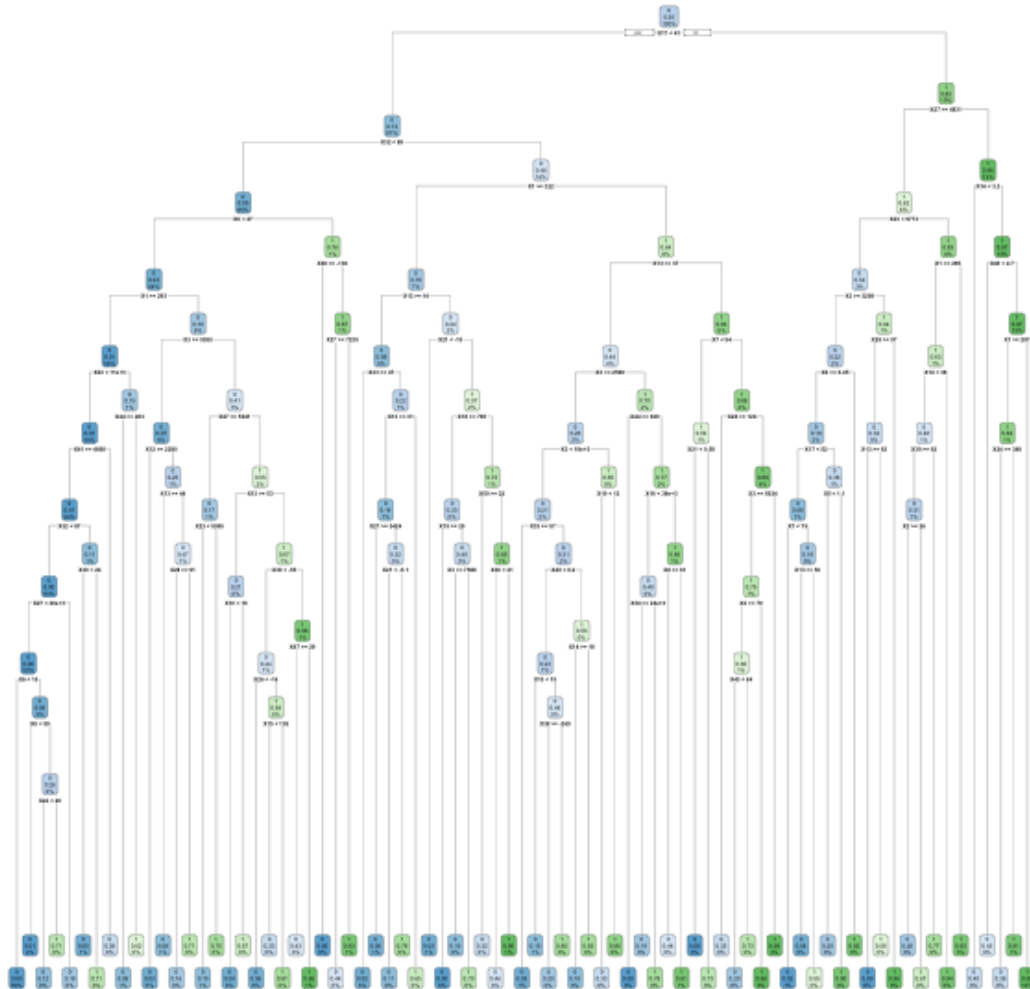


Figura 25: Árbol de decision sin podar

Uno de los principales problemas de los árboles de decisión es su tendencia al overfitting: se ajustan tan bien al train que capturan no sólo la “señal” existente en el train set, sino tambien el “ruido”, de manera que su rendimiento es mucho peor con el test set.

Para reducir este problema, y para intentar mejorar la accuracy, se recurrio, además de las distintas técnicas de preprocesado, a una técnica conocida como pruning o “podado” del árbol: eliminaremos las ramas del árbol que no contribuyen a capturar “señal”.

Para encontrar el balance entre la profundidad y complejidad del árbol con respecto a la capacidad predictiva del modelo en datos de test, se hace hecho crecer el árbol de decisión hasta su mayor extensión y luego hemos buscado el menor error de cross-validation (xerror) en el modelo para posteriormente podar el árbol.

Podemos visualizar un gráfico [29](#) para cross-validation del modelo realizado:

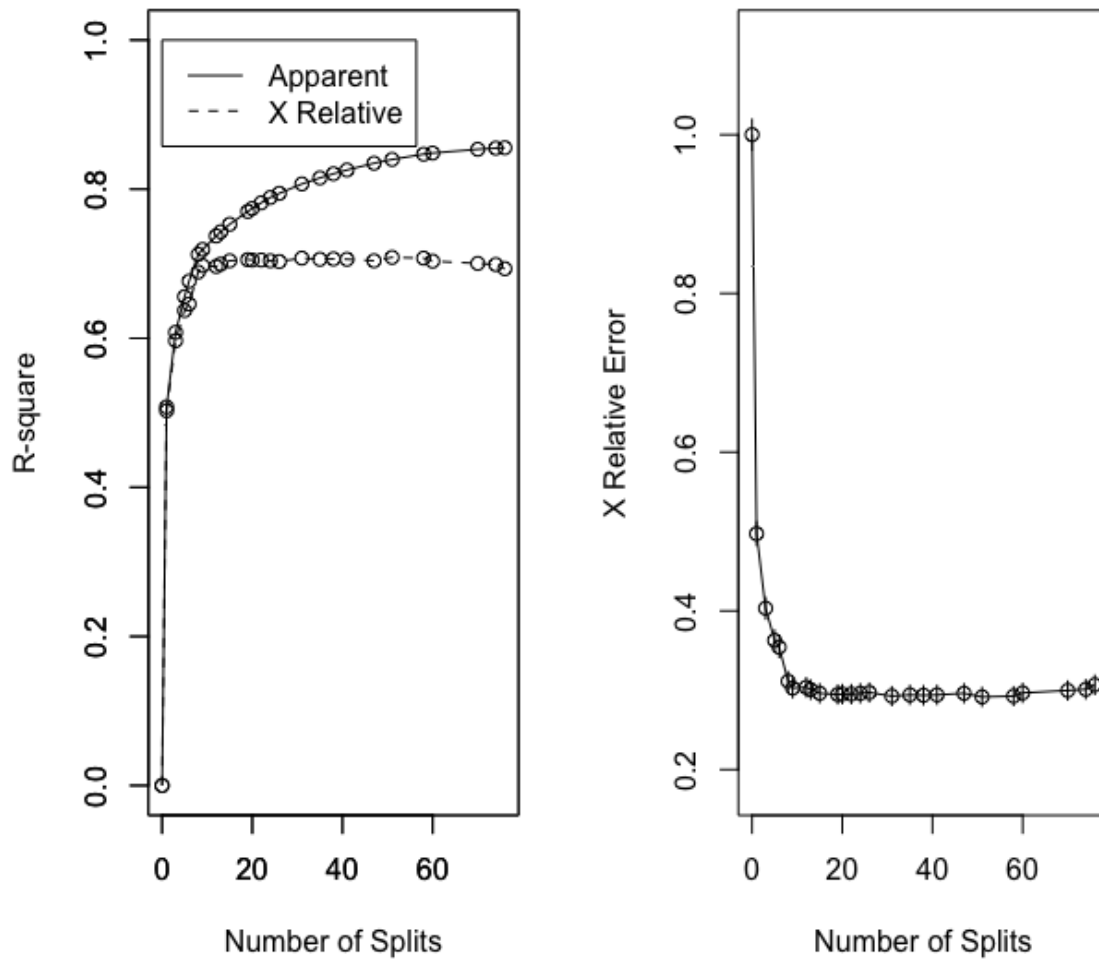


Figura 26: Árbol de decision con poda

A la vista del gráfico del R-cuadrado [29](#) es evidente que estamos sobreestimando la variable C. Tendremos que trabajar el modelo realizando podas de las ramas de nuestro árbol.

Utilizaremos los parámetros de complejidad (CP) como una penalización para controlar el tamaño del árbol.

La información sobre el CP se puede visualizar en el gráfico [27](#):

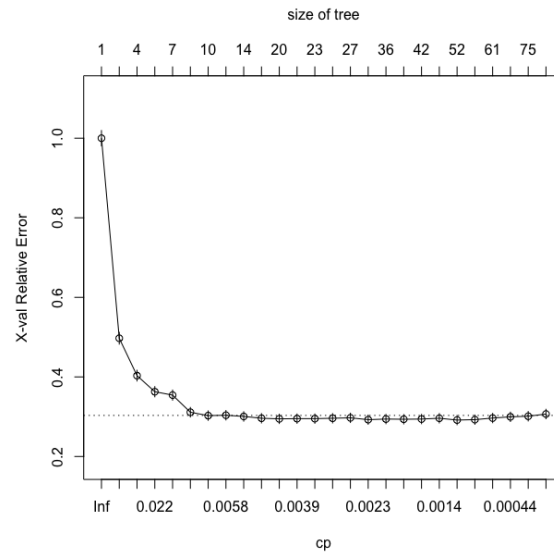


Figura 27: CP

El eje x inferior representa el CP, el eje y es el error relativo y el eje x superior es el tamaño del árbol. Por tanto, escogeremos el valor CP con el mínimo error relativo para podar el árbol.

El resultado final se visualiza en la imagen [29](#).

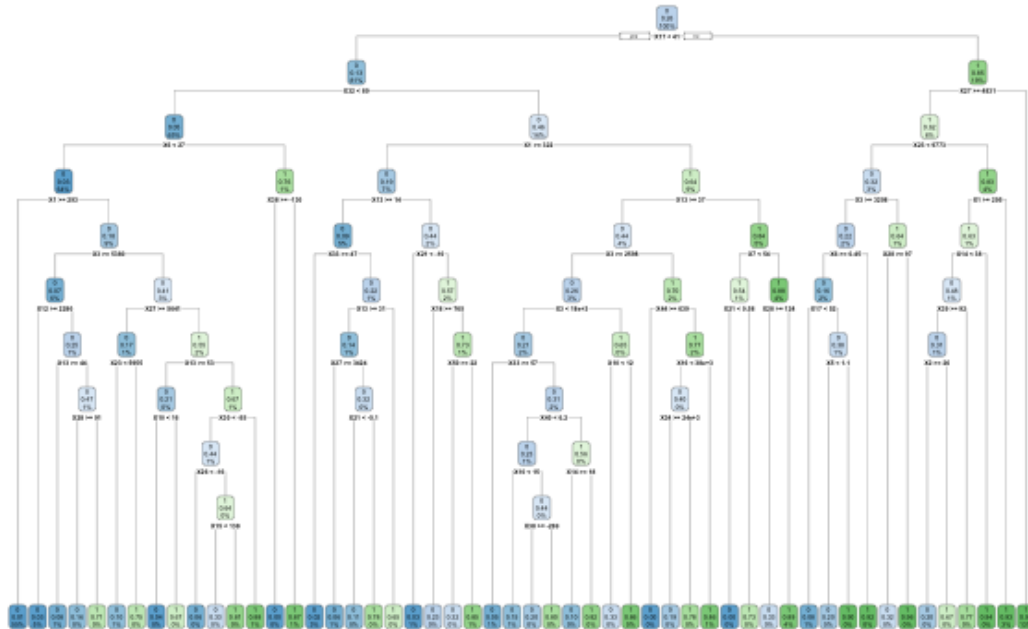


Figura 28: Árbol de decision con poda

A simple vista se puede observar una reducción considerable del tamaño del árbol. Dentro del rectángulo de cada nodo se nos muestra qué proporción de casos pertenecen a cada categoría y la proporción del total de datos que han sido agrupados allí.

5. Máquinas de Soporte Vectorial (SVM)

Las máquinas de soporte vectorial (SVM) aplicadas a problemas de clasificación binaria tratan de dividir los datos de entrenamiento en dos subconjuntos mediante un hiperplano. Así, cada uno de estos subconjuntos se corresponderá con cada una de las clases.

El conjunto en el cual el SVM tratará de encontrar el hiperplano óptimo (minimizando el error de

clasificación) dependerá del Kernel elegido y la optimización del plano de los parámetros (C y ϵ) elegidos. El tratamiento de estos parámetros se desarrollará en los apartados dedicados a ellos.

5.1. Parámetros

Durante el desarrollo de la competición se probaron diferentes parámetros influyentes en el método. Para la elección de estos parámetros se utilizaron los datos con la imputación de valores perdidos que se explicará a continuación y filtrado del ruido básico con IPF.

A lo largo del preprocesado se comprobó que siguieran siendo los óptimos para los datos preprocesados sin producirse cambios en los parámetros óptimos.

- **Elección del kernel:** la elección del kernel puede ser crucial para el funcionamiento del SVM dado que condiciona el espacio al que se transforman los datos. Para esta práctica se consideraron tres de los kernels más utilizados:
 - **Lineal:** buen funcionamiento para datos separables linealmente.
 - **Polinomial:** se eligió probar con grado 2 para evitar el sobreaprendizaje pues grados de polinomios más altos adaptan demasiado el resultado a los datos de train. Comunmente utilizado en procesamiento del lenguaje natural.
 - **Radial:** uno de los más utilizados.

De los kernels utilizados aquel que mejor resultado demostró en validación cruzada fue el kernel radial, por lo que será el que utilizamos para todas las pruebas.

- **Elección del factor de penalización C :** que determina la penalización aplicada por violar el margen. Un alto valor de este parámetro producirá alto sobreaprendizaje.

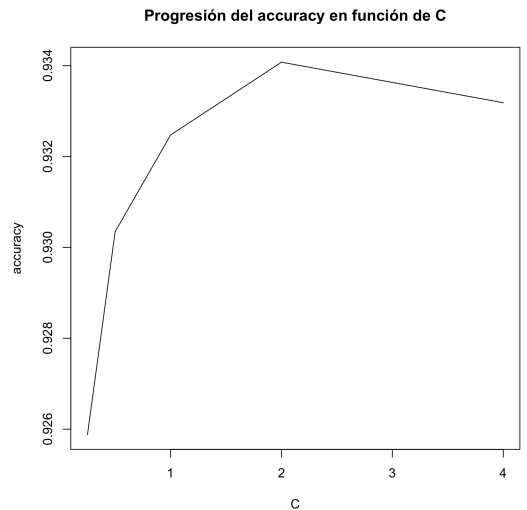


Figura 29: Progresión del accuracy en función de C

5.2. Preprocesamiento

1. En primer lugar, tras el análisis exploratorio de los datos (EDA) se consideró que el primer paso sería la **imputación de valores perdidos** dado que el método no funciona ante la presencia de ambos. En lugar de probar con diferentes imputaciones basadas en medidas estadísticas decidimos probar directamente con imputación basada en randomForest.
2. **Filtrado de ruido:** por la eficiencia computacional y teórica se elige utilizar el filtrado basado en IPF para todas las pruebas.
3. **Corrección de outliers:** para la corrección de los outliers se ha utilizado el método iterativo explicado en la Sección 3.3. Este método se ha planteado con un enfoque iterativo, por tanto, se ha probado con 2,3,4 y 5 iteraciones, consiguiéndose los mejores resultados con **3 iteraciones**.

Este tratamiento de los outliers produjo la mayor subida en la clasificación de la competición.

4. **Selección características:** durante todo el proceso se observó un alto sobreaprendizaje del método. Por tanto, el siguiente paso natural será la selección de características. Se probaron selectores de diferentes enfoques:
 - Selector con enfoque de filtro **chi cuadrado**. Se probó con 40, 35 y 30 características. Al ver que se estaba perdiendo mucha precisión, se decidió no seguir disminuyendo el número de características.
 - Con la selección de 40 características se consiguió otra subida en la clasificación. Por tanto, se decidió ahondar en la selección de características considerando un enfoque embebido, dado que en teoría reunía las ventajas de los otros dos enfoques. Se probó con el basado en Random Forest, aunque no se consiguieron mejoras.
5. **Balanceo de clases:** como ya se ha observado en las secciones de balanceo de clases, el desbalanceo de las etiquetas tras aplicar el preprocesamiento utilizado era muy exagerado. Por tanto, el siguiente paso fue plantearse cómo solucionar este problema. Para ello, se aplicaron:
 - Smote y Tome k Link.
 - Smote con 100 de *percOver*.
 - Smote con 50 de *percOver*.

Aunque estos datos producen mejoras significativas en local, los resultados en test son catastróficos. Esto se puede deber a la replicación de datos que producen alto sobreajuste en los datos.

6. **Transformaciones de los datos:** se aplicaron las transformaciones comentadas en la sección de preprocesamiento.

5.3. Resultados obtenidos

Mostraremos en una tabla los resultados más significativos obtenidos durante la competición, ya sea porque mostraron mejoras significativas o por la idea conceptual aplicada.

| Método | Test público | Test privado |
|--|----------------|----------------|
| Rf.impute + IPF | 0.88565 | 0.89642 |
| Impute + IPF + Corrección outliers (*) | 0.92445 | 0.92755 |
| (*) + 40 características chi.squared | 0.92189 | 0.92193 |
| (*) + 35 características chi.squared | 0.92598 | 0.92142 |
| (*) + 30 características chi.squared | 0.90301 | 0.90357 |
| (*) + Smote 300 + Tome k Links 150 | 0.92291 | 0.91989 |
| (*) + Smote 100 | 0.90199 | 0.89795 |
| (*) + Smote 50 | 0.88820 | 0.87959 |
| (*) + Transformaciones | 0.92291 | 0.92346 |

5.4. Conclusiones

Como podemos ver a lo largo del desarrollo se han probado varias técnicas básicas de preprocesamiento vistas en clase. Sin embargo, la que mejor resultados ha proporcionado (una mayor subida y mejor resultados en test privado) fue la técnica pensada por nosotros en particular para este caso. Al detectar que muchos outliers univariantes estaban provocando la eliminación de instancias potencialmente útiles, se desarrolló la corrección de outliers. Por tanto, podemos ver la importancia de estudiar los datos concretos y no aplicar algoritmos de preprocesado de forma inmediata.

También, resaltar la complejidad de los datos en cuestión pues muchos de los algoritmos utilizados que teóricamente deberían proporcionar mejoras no han mostrado tal comportamiento. Por ejemplo, la selección de características teóricamente debería reducir el sobreaprendizaje en train, sin embargo, el mejor resultado en test privado se ha obtenido utilizando todas las características.

También, destacar que hay ciertos procedimientos que no son significativos para el SVM como por ejemplo el balanceo de clases.

Finalmente, aunque muchas de las técnicas no produjeran mejoras, el estudio realizado para este algoritmo ha proporcionado buenos resultados, situándonos en el puesto 4 de la competición.

6. Clasificador basado en reglas (RIPPER)

RIPPER (Repeated Incremental Pruning to Produce Error Reduction) es un clasificador basado en reglas. Para nuestro caso (2 clases), considera una clase positiva y una negativa y establece reglas para la clase positiva. En nuestro caso, se considera que la clase 0 es la negativa y 1 la positiva. Esto puede tener sentido, pues creamos reglas para la positiva, que en nuestro caso, es la que cuenta con menos representación; cuando no se cumplan ninguna de estas reglas, por defecto, se predice la clase 0 (que tiene más representación).

La estrategia de Ripper para la creación de reglas es de lo general (regla vacía) a lo específico (top-down). Así, añade conjunto que maximicen la ganancia de información. Además, se emplean técnica de poda eficaces (poda usando incremental reduced error pruning).

6.1. Preprocesamiento

Las ideas seguidas para el preprocesamiento (los resultados se mostrarán después) fueron las siguientes:

- Una primera idea era usar Ripper simplemente imputando los valores perdidos con **Random Forest** (también se ha probado esta idea con **Amelia** aunque los resultados eran peores).
- Tras esto, la idea era hacer lo mismo, pero eliminando ruido con **IPF**.
- A partir de este momento, se sigue la idea comentada previamente para imputar los valores perdidos y eliminar ruido. Esto también pasó por dos fases:
 - Eliminar los valores perdidos **imputando** con Random Forest, a continuación eliminar el ruido con **IPF** y tras esto, sustituir los valores de aquellos datos considerados **outliers univariantes** por el valor NA e **imputar de nuevo**.
 - Una segunda idea fue modificar el orden de esta idea: **imputar** los valores perdidos, tras esto, sustituir por NA los valores de datos considerados **outliers univariantes** e **imputar de nuevo**; y para acabar, eliminar ruido con **IPF**.
- En preprocesamiento suele ser importante también reducir la cantidad de variables con la que se trabaja. Se ha probado también, después de este trato de los valores perdidos indicado anteriormente, a seleccionar características, empleando tanto un selector **embedded basado en Random Forest**, como un selector Filter: **chi.squared**.
- Para facilitar la creación de reglas, se consideró buena idea llevar a cabo una discretización del conjunto de datos. Para ello se ha llevado a cabo discretización con **AMEVA** y con **CAIM** (la primera funcionó mejor). La mayor dificultad que tenía esto era que había que obtener los puntos de corte que se establecen en el train, y había que discretizar manualmente el test empleando estos puntos de corte.
- Cuando se trata con árboles en general (al final, las reglas siguen el esquema de un árbol), el desbalanceo puede llevar a peores resultados. Aunque tratar el desbalanceo no garantice mejores resultados, puede ser una buena idea. Así, se ha empleado la función **oversample** del paquete **imbalanced**.
- Para finalizar, con el mejor resultado obtenido, se probó a aplicar lo mismo sobre las variables con las **transformaciones** indicadas en el apartado de transformaciones.

6.2. Parámetros RIPPER

Por término general, podemos modificar tres parámetros:

- **NumOpt**: Número de optimizaciones, es decir. número de vueltas de las optimizaciones.
- **NumFolds**: Número de subconjuntos (folds) por repetición. Tengamos en cuenta que cada fold se usa como conjunto de poda.
- **MinWeights**: Número de pesos mínimo de instancias por partición.

Cuando ya se obtuvieron resultados imputando correctamente los valores perdidos, tratando el ruido, seleccionando características y discretizando, para evitar que se perdieran grandes tiempos proban-

do distintos valores para estos parámetros, se empleaba JRip dando los siguientes valores a estos parámetros:

- $NumOpt = 6$
- $NumFolds = 5$
- $MinWeights = 6$

Parece que con estos valores se obtienen, por término general, buenos resultados. Indicar sólo que esto se empezó a utilizar tras haber realizado ya muchas pruebas, hasta entonces se empleaban, según el caso, los mejores parámetros atendiendo a los resultados de la validación cruzada.

6.3. Resultados

Se muestra a continuación una tabla con los resultados más destacados obtenidos:

| MÉTODO | TEST_PUB | TEST_PRIVADO |
|--|----------|---------------|
| Rf_impute | 0.87697 | 0.87500 |
| Rf_impute + IPF | 0.89280 | 0.89540 |
| Impute + IPF + filtrar_univ | 0.89586 | 0.88724 |
| Impute + filtrar_univ + IPF (*) | 0.89586 | 0.9000 |
| (*) + selec 20 variables | 0.89025 | 0.88826 |
| (*) + selec 30 variables | 0.89285 | 0.88922 |
| (*) + selec 35 variables | 0.89331 | 0.89438 |
| (*) + selec 42 variables | 0.88718 | 0.88469 |
| (*) + CAIM + selec 30 variables | 0.87442 | 0.88163 |
| (*) + AMEVA + selec 34 variables | 0.88106 | 0.88826 |
| (*) + oversample(DBSMOTE y ratio=0.9) | 0.88412 | 0.89438 |
| (*) + transformaciones (- variables 15 y 16) | 0.89586 | 0.89336 |
| (*) + transformaciones + oversample (ADASYN) | 0.87595 | 0.87857 |

6.4. Conclusiones

Podemos observar como, a la hora de imputar, los resultados van siendo los esperados. Para comenzar, no eliminar ruido da resultados bastante peores que eliminarlo. Por otra parte, la idea para imputar los datos y tratar los outliers univariantes parece prometedora. Sin embargo, el primer orden que se estableció (imputar, eliminar ruido y después tratar los outliers univariantes) no da resultados mejores que realizar solo imputación y eliminación de ruido.

Al observar la cantidad de datos que eran considerados como ruido (y eliminados), se podía observar como se eliminaban bastantes datos. La idea al cambiar el orden es pues tratar de eliminar solo outliers multivariantes, de modo que los univariantes se traten, en la medida de lo posible, de la manera indicada anteriormente (sin llegar a eliminarlos). Al realizar los cambios en este orden se obtienen los mejores resultados en esta competición.

Se probaron otras ideas que parecían también prometedoras. Sin ir más lejos, la selección de características suele llevar a mejores resultados. Para ello, se alternan los dos selectores indicados anteriormente, en que los resultados son bastante similares (se subía a la competición el que daba mejores resultados

en cross-validation). Sin embargo, probando distintos números de características señaladas, no se han llegado a mejores resultados que empleando todas las características.

Por otra parte, discretizando los datos tampoco se han obtenido resultados mucho mejores. Realmente, podría considerarse en este punto que Ripper sabe como dividir correctamente los valores de cada variable (empleando en las reglas mayor o menos que cierto valor para cada variable) sin necesidad de una previa discretización.

Otra idea era el trato del desbalanceo. A pesar de que pudiera parece que se podrían conseguir mejores resultados, no era el caso. El motivo puede encontrarse en varios puntos, quizás uno a tener en cuenta sea que en el train, a pesar del trato del ruido, podían seguir existiendo valores tales que, al tratar el desbalanceo, hayan cobrado más importancia y lleven a peores resultados en test (porque en validación cruzada se obtenían mejores resultados).

Para finalizar, con las transformaciones realizadas sobre las variables tampoco se obtienen mejores resultados. Aunque cada variable transformada tenga una distribución más cercana a una normal, esto no tiene que ser sinónimo de mejores resultados. En este caso, los resultados son algo peores.

En resumen, la mejor idea al final es imputar datos con Random Forest, tratar outliers univariantes convirtiendo los valores que los hacen univariantes por NA y volver a imputar, eliminar ruido con IPF y mantener las 50 variables.

7. Conclusión

A modo de conclusión, resaltar la importancia de del preprocesamiento a la hora de aplicar algoritmos de clasificación simples. Una buena preparación de los datos, producirá un mejor comportamiento de los algoritmos y por tanto, mejores resultados.

En un problema real, será crucial dedicar el tiempo necesario a estudiar los datos y sacar conclusiones de ellos antes de aventurarnos a aplicar un algoritmo sin más. También este estudio puede ser útil para decidir qué algoritmo elegir pues cada uno presenta sus propias limitaciones como se ha visto a lo largo de la práctica.