Trabajo Minería de Datos: Preprocesamiento y Clasificación

Juan José Sierra González, Carlos Manuel Sequí Sánchez, Francisco González López

ID Grupo: **Biba er Beti**

1. Introducción

La práctica final de la asignatura Minería de Datos: Preprocesamiento y Clasificación consistía en participar en una serie de competiciones de Kaggle repartidas entre los miembros del grupo. Dichas competiciones incluían árboles de decisión, Support Vector Machine (SVM), K-Nearest Neighbours (KNN) y RIPPER. Por contar este grupo con sólo tres miembros, sólo se participó en las competiciones de árboles, SVM y KNN.

El objetivo de una competición de Kaggle es que los participantes, utilizando todos los mismos conjuntos de train y test, generen un modelo a partir de los datos de train que prediga los valores de la clase de cada uno de los ejemplos de test. En la tabla de resultados de Kaggle se muestra el porcentaje obtenido para un porcentaje de las instancias de test, y al finalizar la competición se revelan los resultados del porcentaje restante, por lo que es importante aportar soluciones que sean capaces de predecir de forma generalizada cualquier nuevo conjunto de datos (como si de un problema real se tratase).

2. Descripción del dataset.

2.1. Train.

El conjunto de datos de train consta de 9144 instancias, cada una de las cuales consta de 50 atributos (X1 a X50) con valores reales clasificadas en dos clases distintas (0 y 1) en una última columna (la número 51, nombrada "C").

2.2. Test.

El conjunto de datos de test consta de 3919 instancias con la misma cantidad de atributos que los datos de train pero esta vez sin la columna que indica la clase de cada una de las instancias con el fin de validar los valores obtenidos con el modelo para este conjunto en la plataforma de competición Kaggle.

3. Preprocesamiento

Primeramente nos centramos en la parte más gruesa del proyecto, tanto por tiempo empleado para su realización como por densidad de código desarrollado para la labor.
Como todos hemos utilizado en mayor o menor medida técnicas de preprocesamiento similares, realizaremos a continuación un listado genérico de los procedimientos usados por todos los componentes del grupo, se hayan incluido o no en el modelo final de cada uno de los problemas basándonos en la eficacia de dichos metodos.

Técnicas de preprocesamiento utilizadas:

- Identificación de datos duplicados.

- Identificación de datos con (lo que parecen) valores erróneos.
- Identificación de outliers mediante detección por distancia intercuartil.
- Imputación de datos de training.
- Identificación y tratamiento de datos ruidosos mediante filtros de ruido.

Clasificación:

- Optimización de hiperparámetros de los modelos.
- Validación cruzada con K particiones.
- Selección de variables y combinaciones.

3.0.1. Identificación de datos duplicados

Este preprocesamiento no consiste más que en buscar aquellos datos cuyos valores de todos sus atributos coinciden. No hemos detectado una gran cantidad de elementos repetidos, pero aun así, resulta conveniente deshacerse de dichas instancias.

3.0.2. Identificación de valores erróneos

Si observamos los datos, podremos ver una tendencia general a que aparezcan en algunas variables valores negativos muy grandes en valor absoluto (comparado con la mayor parte de los datos). Como tienen valores muy similares (-68*10E3), cabría pensar que estos valores se corresponden con algún tipo de error, y parecería una buena idea identificarlos como datos anómalos.

3.0.3. Identificación de outliers por distancia intercuartil

Esta técnica consiste en, para cada variable, calcular la distancia entre el primer y el tercer cuartil "IQR", y marcar como outliers aquellos valores en un entorno del primer y tercer cuartil proporcional a este tamaño. De forma simple, marcamos como outliers aquellos valores que cumplan *alguna* de las siguientes condiciones. Sea x un valor, Q1 y Q3 el primer y tercer cuartil, respectivamente, y lambda un parámetro que fijaremos para considerar outliers valores más cercanos o lejanos a los datos centrales:

$$IQR = Q3 - Q1$$

$$x < Q1 - \lambda * IQR$$

$$x > Q3 + \lambda * IQR$$

Esto se aplica variable a variable, de forma que los valores de unas variables no influyen a la hora de determinar cuando los valores de otras se consideran outliers.

A continuación, en la Figura 1, podemos observar la representación de los valores extremos detectados en el atributo X1 del conjunto de datos, así como la representación de este mismo atributo sin dichos outliers en la Figura 2 (una vez eliminados).

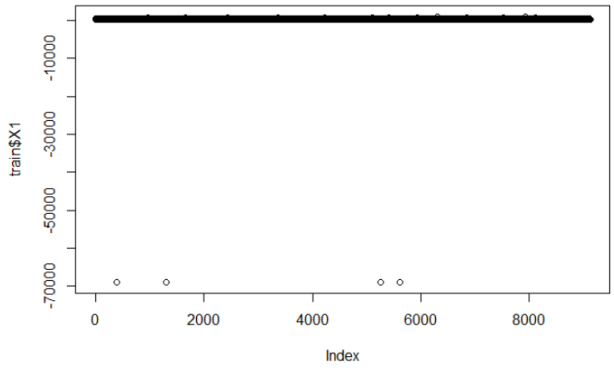

Figura 1: Gráfica dónde se muestran los datos y outliers del problema.

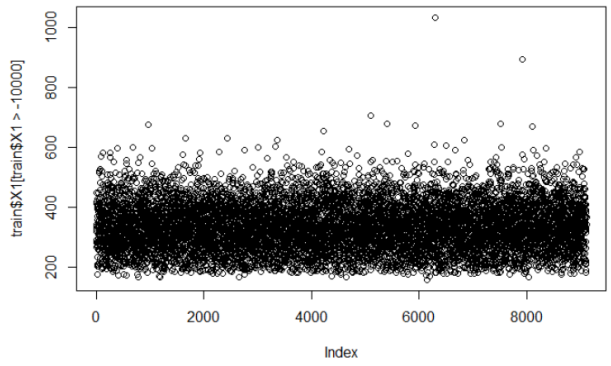

Figura 2: Outliers extremos por rango intercuartil inferior eliminados.

3.0.4. Imputación de datos de training

Para imputar los valores perdidos se han utilizado dos técnicas distintas, dando resultados diferentes a cada uno de los algoritmos, y que ha derivado en que cada uno de los componentes del equipo haya acabado quedándose con el que mejor se adaptaba a su caso.

- **MICE (Multivariate Imputation by Chained Equations)**: Este método crea múltiples imputaciones para cada variable que ha de ser reemplazada, y se basa en el resto de variables que se encuentran en el dataset para decidir qué nuevo valor se adecúa más a cada caso. Además, este método permite incluir un número de imputaciones y de iteraciones por imputación que permitan a cada caso ser más significativo y no regirse únicamente por una componente aleatoria.
- **Amelia**: Este otro método también crea imputaciones múltiples de los valores perdidos pero lo hace utilizando por dentro una implementación del algoritmo EMB o Esperanza-Maximización agregado con Bootstrap para la generación de incertidumbre. Amelia asume que las variables de los datos que le llegan tienen forma de normal multivariante.

3.0.5. Identificación y tratamiento de datos ruidosos mediante filtros de ruido

Este conjunto de técnicas se ha basado en la librería de R "NoiseFiltersR", y se han utilizado los 3 métodos que aparecen en el material de teoría; "EF", "CVCF" e "IPF".

1. filtro EF: *Ensemble filter*. Funciona montando varios clasificadores, haciendo una validación cruzada de los datos, y posteriormente cada clasificador "vota" a cada muestra según si ha podido clasificarla correctamente en validación cruzada (como parte de cada test) o no. De esta forma, los ejemplos que son difíciles de clasificar (los clasificadores han fallado mucho) se marcan como ruidosos.

2. filtro CVCF: *Cross Validated Committees Filter*. Similar al anterior, pero usando ensembles de árboles de decisión, y clasificando todos los ejemplos del dataset (no solo los del fold seleccionado para validación).

3. filtro IPF: *Iterative partitioning Filter*. Consiste en aplicar el filtro anterior, el filtro CVCF, de forma iterativa hasta alcanzar un criterio de parada, que suele ser que durante un cierto número de iteraciones no se hayan eliminado más de un porcentaje fijado de ejemplos del conjunto.

3.0.6. Optimización de hiperparámetros de los modelos

Como su propio nombre indica, esta técnica consiste en buscar un set de hiperparámetros para el entrenamiento del modelo que maximice la métrica que estamos usando para evaluar cada modelo de clasificación. Normalmente usaremos el error medio de validación cruzada con K particiones.

3.0.7. Validación cruzada con K particiones

Esta técnica consiste en separar los datos en K particiones de tamaño similar, de forma que se realizan K entrenamientos distintos utilizando K-1 particiones para entrenar, y una partición para validar (rotando la partición de validación, de forma que todas se validan al menos una vez). Normalmente se observa el resultado promedio. Con esto conseguimos aliviar el efecto que podría tener la selección al azar de los datos mediante un esquema de asignación de muestras para entrenamiento/validación estático.

3.0.8. Selección de variables y combinaciones

Este método trata de buscar un conjunto de variables (podemos excluir variables que consideramos poco informativas para el proceso de clasificación), incluyendo transformaciones (e.g. logarítmica, exponencial) y combinaciones entre variables (e.g. productos entre variables) con el objetivo de mejorar la métrica de clasificación buscando una mejor relación entre las variables de entrada del modelo y la variable respuesta que queremos predecir.

3.1. Árboles (Carlos Manuel Sequí Sánchez)

En este tipo de problema partimos conociendo la siguiente información sobre los árboles de decisión:

- No se requiere conocer la distribución de sus variables.
- Mantienen robustez frente a outliers.
- No les afectan transformaciones monótonas aplicadas a los datos.
- Seleccionan de forma automática los atributos más relevantes para el aprendizaje.

A continuación los resultados de cada experimento.

3.1.1. Tabla de resultados:

3.1.2. Balanceo.

Podemos observar primeramente que se trata de un conjunto de datos con un gran desbalanceo a favor de la clase etiquetada con el valor cero (5995 instancias frente a 3149 instancias de la clase etiquetada con el valor 1).
Esta situación, me incitó a pensar en el uso de algoritmos de balanceo tales como Tomek Links, CNN o SMOTE para el tratamiento de los datos.
En esta vía de investigación para tratar de balancear el dataset, el algoritmo más utilizado ha sido Tomek links, con el fin de aplicar una técnica de undersampliing sobre las instancias de la clase mayoritaria. Este método consiste básicamente en eliminar las instancias de la clase mayoritaria más cercanas a las instancias de la clase minoritaria. Como puede observarse en la tabla de resultados, los valores de acierto obtenidos indican que no es un buen método para utilizar ante el dataset obtenido (ni tan siquiera repitiendo el algoritmo varias veces para conseguir reducir en mayor medida la clase mayoritaria evitando su sobreaprendizaje en comparación con las instancias pertenecientes a la clase minoritaria).
Además de Tomek-Links, también he realizado pruebas con algoritmos como condensed nearest neighbors (CNN) y el famoso SMOTE(Synthetic Minority Over-sampling Technique) basado en vecindarios, para intentar, en este caso, aumentar la cantidad de instancias de la clase minoritaria. El resultado con estos algoritmos fue incluso peor que con tomek links. Incluso he llegado a probar un algoritmo basado en la concatenación de Tomek-Links seguido de Cnn, sin éxito algun claro esta (ubOSS - one side selection).

3.1.3. Instancias duplicadas.

Con el fin de evitar redundancia de información, tal como se ha comentado en la parte de preprocesamiento, procedí a comprobar la existencia de valores duplicados, lo que me llevó de forma directa al siguiente punto, la detección de outliers, ya que las instancias detectadas como elementos duplicados, parecían más bien datos extremos erróneos.

3.1.4. Detección de outliers.

Aún sabiendo que los árboles de decisión poseen gran robustez ante los valores extremos de un dataset, me decidí por encontrar y eliminar aquellos valores anormales que pudiesen causar cualquier tipo de reducción de veracidad en la información que nos aportaban los datos. Para ello la etapa queda dividida en dos partes:

- primeramente, tal como dije en el punto anterior, eliminé dichas instancias duplicadas que alcanzaban concretamente la misma cifra (-68931) para todos y cada uno de sus atributos, lo cual parece lógico, pues no tiene sentido ni el altísimo valor que representa ese número ni el hecho de que todos los atributos de una variable tengan ese mismo valor (más aún tratándose de instancias con un total de 50 atributos).

- En segundo lugar, tal como se ha descrito en la etapa de preprocesamiento, eliminé los “outliers"situados fuera de los cuartiles uno y tres más el rango intercuartil multiplicado por un factor lambda. He puesto entrecomillado el término *outliers* en este caso porque, tal como puede apreciarse en la Figura 2, realmente existen muy pocos outliers como tal (parece que solamente dos instancias a lo sumo).

Tal como es de esperar, sabiendo la robustez ante outliers que posee un algoritmo de árboles, y además, teniendo en cuenta que a simple vista (en el plot de datos) existen muy pocos valores anómalos en el dataset, el uso de la técnica de eliminación de outliers mediante el rango intercuartil (IQR) no ha aportado prácticamente nada a la hora de ayudar al modelo a extraer conocimiento de manera óptima, por lo que fue eliminado del modelo final con el que conseguí los mejores resultados.

3.1.5. Comprobación e imputación de valores perdidos.

Nos disponemos, tras la detección de valores anómalos, a buscar en el conjunto de datos valores perdidos (valores NA en nuestro caso, ya que, a la hora de leer los datos, nos hemos encargado de transformar los caracteres representantes de valores perdidos ("?") en valores NA).
Con un par de simples consultas en RStudio podemos comprobar que absolutamente todos los atributos contienen varias instancias con valores perdidos, y que no hay una instancia en todo el dataset que tenga más de un atributo con valor NA, por lo que sabemos que estos no están concentrados en pocas instancias, si no que estan repartidos.
A la hora de escoger un algoritmo de imputación de valores perdidos, me decanté finalmente por el uso de Amelia, ya que en un principio, el paquete MICE no me daba ningun tipo de problemas porque justo antes de realizar la imputación de valores perdidos, realizaba la eliminación de valores extremos mediante el uso de la teoría basada en el rango intercuartil, pero conociendo la robustez ante este tipo de datos por parte de los algoritmos de árboles, decidí eliminar este estudio del experimento y fue entonces cuando MICE me produjo fallos. ¿Qué tipo de fallos? Básicamente, al eliminar este tipo de preprocesamiento, de alguna forma MICE, al realizar internamente la matriz de correlaciones entre atributos, detectaba algunos con alta correlación y es sabido, que MICE no realiza la imputación de valores para valores pertenecientes a atributos altamete correlados, por lo que tras "imputar valores" con MICE, mi dataset seguía teniendo valores NA. Amelia no posee este tipo de comportamiento, por lo que me funcionaba sin problemas y además, sabiendo que también asume que los valores perdidos de mi dataset son missing at random (MAR, es decir, que los valores perdidos dependen de los datos observados y no de los no observados), me decanté finalmente por su uso.

3.1.6. Eliminación de ruido.

Esta etapa, en la búsqueda de un porcentaje de acierto mejor por parte de mi modelo, ha sido totalmente crucial (tal como podemos observar en la tabla de resultados), ya que, al parecer, el dataset en objeto de estudio posee una gran cantidad de datos ruidosos que hacen que el aprendizaje resulte más complicado. Para la eliminación de ruido, de los tres algoritmos distintos explicados en la etapa de preprocesamiento pertenecientes al paquete NoiseFiltersR, he escogido el filtro EF (Ensemble Filter), ya que he realizado pruebas con los otros dos filtros (CVCF e IPF) y nunca han llegado a ser tan exitosas como EF.

3.1.7. Selección de características.

Aún sabiendo que algoritmos basados en árboles de decisión son capaces de escoger por sí solos las características más relevantes en las que basarse para la obtención de un modelo adecuado a la hora de aprender unos datos para realizar una correcta clasificación, ante la desesperación de incrementar el porcentaje de acierto, me decidí por utilizar técnicas de selección de características basadas en dos distintas:

- **Correlación entre atributos:** a partir de la cual se eliminan los atributos que guardan una alta correlación con alguno de los demás atributos de las instancias con el fin de evitar la redundancia de información y facilitar el aprendizaje. En esta vía de estudio he hecho uso de funciones como *cor* y *finCorrelation*.
- **Correlación entre cada atributo con la clase:** a partir de la cual se estudia cuales son los atributos más relevantes a la hora de decidir a qué clase pertenece una instancia, de esta forma a la hora de utilizar el algoritmo de aprendizaje, podemos indicarle mediante una lista, dichos atributos para ignorar el resto y centrarse en la información realmente relevante. En esta segunda vía de estudio he hecho uso de un algoritmo basado en Random Forest, el cual mediante el uso de pesos devuelve una lista con los atributos más importantes.

3.1.8. Selección del algoritmo final de aprendizaje.

Primeramente comentar que para cualquier modelo creado a partir de cualesquiera de los algoritmos que he probado, he basado mis subidas a la plataforma Kaggle haciendo previamente uso de una 10 fold-cross validation, es decir, para evitar perder cualquiera de las subidas diarias a la plataforma, he hecho uso de dicha técnica para, más o menos, guiar mis esperanzas de éxito al subir los resultados obenidos por el modelo a Kaggle.

En cuanto al algoritmo utilizado para el aprendizaje de cada uno de los modelos, he de decir que he hecho severas pruebas con los algoritmos rpart(classification an regression trees), ctree(conditional inference trees), J48 (C4.5 pruned or unpruned trees) y LMT(logistic model trees). Comencé primeramente realizando pruebas básicas con los algoritmos ctree, rpart y J48, llegando a la conclusión tras varias subidas de que en este caso, J48 ofrecía mejores resultados que los otros dos algoritmos.
Una vez escogido dicho algoritmo me propuse optimizar sus dos únicos parámetros para obtener una mejor, aunque leve, predicción de los datos de test. Los valores de estos parámetros fueron extraídos concretamente utilizando validación cruzada con el dicho algoritmo, previa aplicación del algoritmo en sí, es decir, primeramente obtenemos los valores de configuración optima para los parámetros C y M de J48 (valor de confianza y número mínimo de instancias en las dos rámas más populares, respectivamente), y de forma posterior aplicamos el algoritmo J48 sobre el dataset con dicha configuración óptima hayada previamente.
Fue prácticamente al final de la competición cuando descubrí que con el uso de LMT (logistic model trees) el modelo se ajustaba de mejor manera a los datos, aportándome la mejor de mis soluciones en la competición por encima de cualquier otra en la que hubiera utilizado el algoritmo C4.5 (además de todo el grueso del preprocesamiento aplicado). A continuación podemos ver el árbol obtenido por este algoritmo.

Árbol obtenido por el modelo final.

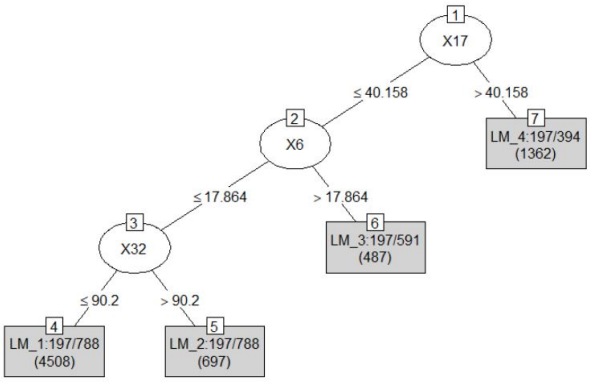

Figura 3: Árbol generado por el modelo final.

Este es el árbol obtenido por el modelo final utilizando el algoritmo LMT. En él podemos apreciar el uso de tan solo 3 características de las 50 existentes en el aprendizaje de los datos (X17, X6 y X32), podemos ver tambien la cantidad de instancias existentes en cada uno de los nodos hoja (representadas con un número entre paréntesis), así como la proporción de error de clasificación en cada nodo hoja (LM_X: <error>)

3.2. K-Nearest Neighbours (Juan José Sierra González)

Para comenzar el análisis del algoritmo KNN merece la pena recordar que, debido a su condición de comparar individuos del conjunto de datos mediante distancias, necesita tener sus datos normalizados para poder llevar a cabo las operaciones sin que los valores de orden más grande influyan más que aquellos de orden menor.

Sin embargo, por querer comenzar las subidas desde lo más básico, empezando por un preprocesamiento simple sin mucho pulimento para poco a poco ir perfeccionándolo, subí por accidente las dos primeras soluciones a Kaggle sin aplicar normalización. Por este motivo, aunque el preprocesado aplicado es reutilizado en las siguientes subidas, estas no deberían ser fiables pues son propensas a sesgos.

La primera subida de todas aplicaba un preprocesamiento muy sencillo: en primer lugar eliminaba las filas que tenían valores perdidos, a continuación clasificaba como nuevos valores perdidos (NA) los outliers (tanto simples como extremos) y entonces los imputaba usando la librería **MICE**. Los valores perdidos venían indicados como "?" en el dataset original, pero en la lectura de los datos se cambió a NA. Para evitar que el resultado estuviese totalmente determinado por la elección de la semilla, generaba 3 imputaciones con un total de 3 iteraciones por imputación, y me quedaba con aquella que tuviese más correlación con las otras, por considerarla más genérica. Además, para ejecutar el algoritmo propiamente dicho utilizaba el paquete **Caret** de R y, mediante la función **train**, aplicaba el método KNN y generaba los modelos. En particular, escogí desde un primer momento probar con 10 K impares a partir de 5, es decir, entre 5 y 23, y la función train me devolvía el mejor de todos. Para escoger el mejor modelo, indiqué a la función train que validase cada modelo generado utilizando el método **cross-validation** con un total de 5 divisiones del conjunto de entrenamiento. Por último, calculé las predicciones de los datos de test utilizando la función **predict**. El resultado igualón al de la última posición.

Para la segunda subida el preprocesamiento añadido fue eliminar las filas duplicadas, y además, en este caso imputar todos los valores perdidos, tanto los NA originales como aquellos generados manualmente procedentes de valores que eran considerados outliers. El resultado de dicha subida no mejoró lo anterior.

Para la siguiente subida pasé de imputar todos los outliers a solamente los outliers extremos, partiendo de la base de que existen muchos datos considerados como outliers que sin embargo pueden formar parte de un amplio grupo que no se encuentra muy alejado del núcleo principal de datos. Además, en esta subida ya sí realicé una transformación de centrado y escalado, pero sin embargo por error sólo lo hice sobre los datos de train y no también sobre los de test antes de predecirlos. A pesar de ello, esta subida ha sido la que ha obtenido una mejor puntuación global y que en el momento me permitió subir hasta el quinto puesto.

En la cuarta subida añadí el preprocesamiento de centrado y escalado utilizando la función **preProcess**, también del paquete Caret, y que permite guardar la configuración de dicho preprocesamiento para poder utilizarlo más adelante en test. En este caso sí se normaliza también el test y sin embargo los resultados empeoran ligeramente, a pesar de seguir el procedimiento correcto.

Para la siguiente entrega apliqué por primera vez una selección de atributos, en este caso eliminando aquellas variables que tuvieran un índice de correlación mayor a 0.75 o menor a -0.75, es decir, con una correlación bastante elevada. El resultado fue muy similar al anterior, sin llegar a superarlo.

En la sexta entrega decidí aplicar la técnica **Principal Component Analysis**, o comúnmente conocida como **PCA**. Esta técnica crea nuevas variables no correladas entre sí que tienen que explicar un umbral mínimo de la varianza original de los datos. Esto a menudo permite reducir la dimensionalidad del problema, ya que se puede representar una información similar en muchas menos variables. Aplicar PCA en mi caso no aportó mucho; aunque obtuve mejores resultados que en la subida anterior, estos no superaron la mejor subida hasta el momento.

Para la séptima subida quise utilizar algún método de oversampling o undersampling. Si bien el desbalanceo de clases no es exagerado, sí que existe una cantidad de datos de la clase 0 superior a la clase 1, un 65'6 % pertenecen al tipo 0 mientras que un 34'4 % lo son al 1. Movido por esta situación decidí utilizar **SMOTE** como herramienta para generar nuevos datos artificiales de la clase minoritaria (oversampling) basados en los ya existentes. Los resultados fueron decepcionantes y empeoraron significativamente.

En la octava subida cambié el oversampling que tan mal resultado dio por undersampling, es decir, eliminé instancias de la clase mayoritaria para tratar de balancear las clases. Para ello utilicé el método "down" dentro del atributo "sampling" en la función anteriormente citada **preProcess**. Además, dada su poca incidencia en el porcentaje de clasificación obtenido, decidí eliminar PCA del preprocesado, volviendo a dejar solamente el centrado y escalado. El resultado fue mejor que el reflejado con oversampling, aunque nada que incitase a continuar por esa vía de investigación.

Para la novena subida decidí probar otro método de imputación de valores perdidos distinto a MICE, en este caso se trata de **Amelia**. Además, de forma complementaria se incluyó en el preprocesamiento de los datos la eliminación de aquellas filas que contuviesen únicamente valores perdidos y/o outliers. Mantener estas instancias e imputarlas completamente sería en esencia entrenar con variables generadas "aleatoriamente" y que no tienen un fundamento real. Utilizar Amelia no dio mejores soluciones que MICE así que opté por seguir utilizando el método de imputación anterior.

En las subidas 10, 11 y 12 decidí probar los principales métodos de filtrado de ruido que se encuentran en el paquete **NoiseFiltersR**, implementado y mantenido por el grupo de investigación SCI2S. Los métodos utilizados fueron **EF** (Ensemble Filter), **CVCF** (Cross-Validated Committees Filter) e **IPF** (Iterative Partitioning Filter). De todos ellos, aunque el aumento en el conjunto de train era muy considerable (lógico dado que se está perdiendo ruido, es decir, elementos que intrínsecamente empeoran el clasificador), los resultados en Kaggle no mostraban mejora alguna. Esto también es entendible dado que a pesar de todo el ruido existe en el conjunto de test y por supuesto en los datos reales. Dicho esto, como nota destacable, el clasificador con CVCF ha resultado ser mi mejor subida en el marcador privado, y sin embargo no fue incluida entre mis 10 candidatas a mejor solución.

En la decimotercera subida, a causa de los resultados poco positivos obtenidos en los intentos recientes, decidí volver al código de la subida 4 (la mejor que tenía hasta el momento, sin contar la que contenía el error de no normalizar el test) y a partir de ahí cambiar algunas cosas. Por supuesto se eliminó el filtrado de ruido, y además probé sin hacer selección de características en base a correlación entre ellas por si al final estaba quitándome alguna información valiosa. Como cambio principal en esta subida, decidí eliminar únicamente los outliers extremos del extremo inferior, basándome en la distancia intercuartil. Como se puede observar en la Figura 1, existen unos outliers exagerados en el extremo inferior, y sin embargo no se aprecia ninguno de igual magnitud en el superior. De hecho, una vez eliminados dichos outliers, el aspecto de los datos mejora sustancialmente (ver Figura 2), pudiendo entenderse que los "outliers" superiores pueden ser una extensión más o menos normal de los datos. Los resultados volvieron a la normalidad pero no mejoraron.

En la decimocuarta subida probé lo mismo que en el caso anterior pero haciendo selección de características que superasen (en valor absoluto) un 0.95 de correlación. Es decir, mi intención era eliminar únicamente aquellas que fuesen estricta o prácticamente similares. El resultado empeoró ligeramente por lo que tampoco se mantuvo esta idea.

Para la subida 15 decidí cambiar la función que estaba utilizando para ejecutar el algoritmo KNN y utilizar en su lugar el paquete **kknn**, que tiene una función con el mismo nombre. Esto vino en parte inspirado debido a que quise utilizar un algoritmo de selección de características (CFS) que devolvía como resultado una fórmula, y la función train de Caret no podía trabajar con ella, mientras que este paquete sí lo hacía. La función CFS la obtuve del paquete **FSelector**, pero los resultados obtenidos con ella y la nueva forma de ejecutar KNN fueron desastrosos y los di por apartados para futuras subidas.

En la subida 16 y 17 decidí utilizar un nuevo método de undersampling (que como había especificado antes, me aportó mejores resultados que oversampling). En este caso probé con el conocido algoritmo **Tomek-Links**, que reduce dimensionalidad de la clase mayoritaria en las fronteras entre ambas clases. De-

bido a su comportamiento porcentual, tiene sentido aplicar Tomek-Links más de una vez, para cada vez reducir más variables de la clase mayoritaria y además seleccionar nuevas "fronteras" cada iteración. Es por eso que la subida 16 aplica Tomek-Links una sola vez mientras que la subida 17 lo aplica dos. El resultado en ambas es peor que lo que se había obtenido con anterioridad así que descarté seguir por esta línea.

Para la subida 18, observando los datos de nuevo y comprobando lo absurdo de los valores extremadamente bajos (del orden de -68000) que aparecían en algunas variables, decidí considerar todas las instancias con **al menos** un valor de dicho orden como basura, y las eliminé del conjunto de entrenamiento. No funcionó tan bien como esperaba y resultó restar acierto al modelo.

El modelo de la subida 19 se hizo eliminando las variables altamente correladas mediante el método **corr** de la función preProcess. Este podía incluirse junto a las transformaciones de centrado y escalado, por lo que además es una transformación que luego se puede realizar a los datos de test de forma análoga. Al incluir este método el modelo mejoró con respecto a las subidas anteriores, salvo de nuevo a la subida errónea.

La última subida, la vigésima, fue volviendo a eliminar los outliers extremos superiores además de los inferiores, y haciendo uso del método para eliminar variables altamente correladas que se heredó de la subida anterior. En el marcador público empeoró ligeramente, aunque a posteriori he podido comprobar que esta ha sido una de las mejores subidas, y la mejor de las que he escogido sin contar con la tercera que no tenía un preprocesamiento bien realizado.

3.3. Support Vector Machine (Francisco González López)

Los resultados obtenidos y una breve descripción indicando las particularidades de cada entrega marcada para el leaderboard privado se puede encontrar en la tabla 3. Más abajo se comenta cada solución y la influencia de las técnicas utilizadas.

3.3.1. Nota sobre la validación cruzada

Cabe comentar respecto de los resultados de validación cruzada; aunque pongamos en pie de igualdad todos los valores obtenidos, **no podemos considerarlos directamente comparables**. Al utilizar técnicas de eliminación/recuperación de instancias, cada conjunto de acciones y parámetros de preprocesamiento nos van a dejar diferentes conjuntos de datos para entrenar (y hacer validación cruzada). Si eliminamos la mayor parte de ejemplos "más diferentes" de una base de datos, es de esperar que los ejemplos que quedan se parezcan más entre sí, y por tanto la validación cruzada reporte mejores resultados (que luego no vamos a ver reflejados en los resultados del test). En general, cuanto más "agresivo" sea el preprocesamiento, mejor resultado obtenemos en validación cruzada, pero esto no se traduce en una mejoría (incluso empeora) en los resultados de test.

3.3.2. Partes comunes

Como se puede observar en la **tabla 3**, todas las soluciones presentadas implementan 2 técnicas de preprocesamiento.

Por un lado, tenemos la **eliminación de duplicados** . Esta técnica consiste simplemente en buscar las entradas con exactamente los mismos valores, y eliminarlas. Es un preprocesamiento muy básico, que tiene un coste muy pequeño, y que estamos seguros de que no va a influir negativamente, ya que la información redundante no suele guardar ningún valor teniendo en cuenta la cantidad de variables (50) que estábamos tratando.

Por otro lado, la **eliminación de valores $<$ -50000** me pareció una buena idea, ya que como se muestra en la **figura 1**, en la mayoría de variables encontramos algunos de estos outliers que están tan seprados de la mayoría de valores, y que pueden afectar fuertemente a otros métodos que utilicen medidas de centralidad

como la media (no tanto las medianas). Como todos tenían valores similares, he considerado que se trataba de algún tipo de error en el dato, ya sea como sensor defectuoso, problema de lectura o codificación, etc.

Finalmente, **selección de variables y ajuste de hiperparámetros**. Después de realizar algunas pruebas, decidí que como no obtenía mejoras significativas en los resultados, y la asignatura se centra en aplicar técnicas de preprocesamiento, decidí dejar fijos los parámetros del modelo (SVM del paquete "e1071") por defecto (kernel=radial, coste=1, sigma=1/número ejemplos). Algo que no he probado ha sido las transformaciones y combinaciones explícitas entre variables, pero de nuevo, no lo he considerado tan interesante dado el marco teórico.

3.3.3. Detección de outliers por rango intercuartil(outliers (solo por abajo) por IQR)

En un principio, esta técnica se probó eliminando outliers tanto por encima como por debajo de los cuartiles, pero sin embargo la eliminación de los valores por arriba resultó ser muy detrimental para los resultados. Posteriormente, la observación de los datos gráficamente parece indicar que mientras que los outliers por abajo sí que parece que se pueden eliminar, los que este método detecta por arriba tienen una densidad y distancia que parece que sí que tienen información relevante para la clasifiación. Por tanto, en general la decisión de eliminar outliers ha sido únicamente aquellos valores menores que **el primer cuartil menos 3 (parámetro lambda) veces el rango intercuartiles**. En cualquier caso, esta técnica se puede considerar bastante rudimentaria y no parece haber aportado mucho respecto a otras mejoras.

3.3.4. Filtros de ruido (filtro EF, filtro CVCF, filtro IPF)

Como podemos ver en la **tabla 3**, los filtros de ruido están presentes en las mejores soluciones alcanzadas, tanto en validación cruzada como en test público y privado. En primer lugar, respecto del filtro IPF, podemos ver que obtiene muy buenos resultados en validación cruzada, pero no en los set de test. Probablemente esto se deba a que elimina muchas más instancias que el resto (ya se ha comentado este problema al principio de la sección), ya que se trata de una aplicación iterativa del filtro CVCF. Por otro lado, el filtro EF y el filtro CVCF obtienen los mejores resultados en el leaderboard público y privado respectivamente. La pega de utilizarlos es que son modelos complejos, que se basan en ensembles de clasificadores, y esto hace que sean poco interpretables, además de que al no hacer clasificación con ellos, sino detectar ruido, la interpretación es más indirecta. Como el modelo que usa el filtro EF es el que reporta mejores resultados en el leaderboard público, es el que habríamos elegido si hubiéramos tenido que elegir un único modelo, pero esto hubiera hecho que hubiéramos tenido resultados peores en el conjunto de test privado final (aunque no por mucho).

3.3.5. Balanceo: balanceo con Tomek links

El dataset que tenemos tiene una proporción de clases desigual; el 65 % de las muestras tienen etiqueta negativa, y el 35 % restante tienen etiqueta positiva. Esto hace que las muestras negativas estén sobrerepresentadas, de forma que el clasificador puede tender a intentar ajustar mejor la clase mayoritaria (en este caso, la negativa). Para intentar aliviar el efecto del desbalanceo, se puede intentar eliminar enlaces de Tomek, que consiste (en la variante utilizada) en eliminar aquellos ejemplos de la clase mayoritaria que forman un enlace de Tomek, formado por dos datos que son vecinos más cercanos el uno del otro.

Sin embargo el balanceo del set de datos no parece ser una buena idea por ninguna de las métricas que utilizamos, de forma que no lo incluiría en un modelo final. Esto me resultó extraño, ya que los modelos SVM no se benefician mucho por tener un gran número de muestras (por construcción, las muestras importantes son los vectores de soporte, que son un subconjunto relativamente reducido).

3.3.6. Imputación test: imputar test con knn

Una de las técnicas que me parecían más interesantes es la imputación (reconstruir instancias incompletas) de datos de test mediante los datos de training. Hay que tener cuidado al hacer esto, ya que no podemos imputar los datos de test, ni detectar outliers, utilizando los mismos datos de test. Por ello, para la detección de outliers en test (que dejan algunas instancias de test incompletas para su posterior imputación, ya que

no podemos "eliminar" una instancia de test), utilizamos los umbrales y valores obtenidos del conjunto de entrenamiento.

Posteriormente, utilizamos una imputación con KNN (del paquete "DMwR2"), de forma que los valores perdidos en cada instancia de test se reconstruyen mediante votación de las K instancias de entrenamiento más cercanas (en el espacio de valores que conocemos de cada muestra). Se ha usado K=3.

Esta técnica, aunque esperaba obtener algún resultado positivo, ya que me parecía que en el conjunto de test hay muestras anómalas, pero sin embargo, no ha sido así. Comparando en todos los casos la predicción sobre el test imputado y sin imputar, en ninguno he encontrado diferencia significativa, como mucho un cambio de 1 valor en los 5-6 experimentos de los que esta técnica ha formado parte. En conclusión, la imputación con knn del test no afecta a los valores de la predicción obtenida en los experimentos planteados.

3.3.7. Imputación training: inputar train con mice

La imputación de los valores de training (es decir, reconstrucción de los valores que hemos eliminado marcando como outliers) se ha hecho con el paquete "mice", utilizando siempre los parámetros por defecto, excepto generando una imputación en lugar de varias como hace mice por defecto. Probando a inputar train con mice por separado del resto de técnicas (no está incluido en la **tabla 3**), su efecto (cuando se incluye) parece ser el más importante de cara a la mejora en los resultados de test (tanto público como privado).

En este aspecto hubiera sido interesante probar otros métodos de imputación, como los incluidos en el paquete "Amelia" de R, que permiten hacer imputación multivariante (mice funciona variable a variable).

3.3.8. Conclusiones y modelo final

De cara a los mejore resultados obtenidos en conjunto, el mejor modelo sería el 3 de la **tabla 3**. Sin embargo, como tenemos que guiarnos por los valores obtenidos en la clasificación de la parte pública del set de test, nos decantaríamos por el modelo 2 de la **tabla 3**.

Una vez considerados simplemente los valores de precisión de test, podemos pasar a ponderar hasta qué punto nos interesa complicar el preprocesamiento para mejorar los resultados. Si hubiera que elegir un procedmiento, me decantaría por uno más simple, como serían los utilizados en los modelos 1 y 6 de la **tabla 3**. La razón es que del conjunto de operaciones que se han probado, estas 3, eliminación de duplicados , valores $<$ -50000, y aquellos marcados como outliers (solo por abajo) por IQR, resultan ser las más simples, y reportan unos resultados marginalmente peores que los obtenidos aplicando preprocesamientos más complejos (detección de ruido, etc.), por lo que creo que puede merecer la pena sacrificar un poco de calidad en el resultado final a cambio de una reducción de complejidad en el procedimiento.

Como mejoras a este modelo podría ser interesante hacer un estudio más profundo de cada variable, así como una selección y transormación de las variables más extensa, que ya he comentado que he dejado un poco de lado en favor de explorar diferentes opciones de preprocesamiento de los datos.

Nº Subida	Descripción	% Acierto Público	% Acierto Privado
1	Algoritmo para predicción: rpart, eliminación de outliers extremos mediante IQR, imputación con MICE.	0.85706	0.86173
2	Algoritmo para predicción: ctree junto con PCA centrado y escalado como control de preprocesamiento, eliminación de outliers extremos mediante IQR, imputación con MICE	0.86217	0.86836
3	Algoritmo para predicción: J48(C4.5) con parámetros del algoritmo optimizados, eliminación de outliers mediante IQR, imputación con MICE	0.85145	0.85663
4	Algoritmo para predicción: J48(C4.5) con parámetros del algoritmo optimizados, imputación con Amelia, eliminación de ruido (EF sin consenso)	0.88463	0.88928
5	Algoritmo para predicción: J48(C4.5) con parámetros del algoritmo optimizados, eliminación de outliers mediante IQR, imputación con Amelia, eliminación de ruido (EF sin consenso), selección de características mediante correlación entre estas	0.88769	0.89336
6	igual que anterior pero modificando el threshold en el estudio de la correlación entre atributos para la selección de características	0.88871	0.88673
7	Algoritmo para predicción: J48(C4.5) con parámetros del algoritmo optimizados, imputación con Amelia multicore, eliminación de ruido (IPF), selección de características mediante correlación entre estas	0.87953	0.87704
8	Algoritmo para predicción: J48(C4.5) con parámetros del algoritmo optimizados, imputación con Amelia multicore, eliminación de ruido (EF), selección de características mediante correlación entre estas y mediante el uso de Random Forests	0.89127	0.88877
9	Algoritmo para predicción: J48(C4.5) con parámetros del algoritmo optimizados, imputación con Amelia multicore, eliminación de ruido (EF), uso de Tomek Links para técnica de undersampling sobre la clase mayoritaria selección de características mediante correlación entre estas y mediante el uso de Random Forests	0.87289	0.88673
10	Algoritmo para predicción: LMT, imputación con Amelia multicore, eliminación de ruido (EF)	0.90709	0.90561

Cuadro 1: Análisis de las subidas para el algoritmo de árboles de decisión

Nº Subida	Descripción	% Acierto Público	% Acierto Privado
1	Imputación de outliers con MICE y eliminación de NA	0.87442	0.86938
2	Eliminación de duplicados e imputación de todos los NA	0.87340	0.87142
3	Eliminación sólo de outliers extremos, normalización sólo en train	0.91066	0.91173
4	Transformaciones y normalización con preProcess, tanto en train como en test	0.90658	0.91020
5	Selección de características por |correlación| >0.75	0.90607	0.91122
6	Aplicación de PCA	0.90658	0.90918
7	Utilización de SMOTE para oversampling	0.82899	0.83367
8	Reducción mediante undersampling y eliminación de PCA	0.87748	0.88520
9	Eliminación de filas llenas de outliers e imputación de valores perdidos con Amelia	0.90505	0.90867
10	Filtrado de ruido con EF	0.89892	0.90969
11	Filtrado de ruido con CVCF	0.89892	0.91275
12	Filtrado de ruido con IPF	0.90045	0.90969
13	Vuelta a subida 4, eliminación de outliers inferiores y sin filtrado de ruido ni selección de características	0.90607	0.90918
14	Selección de características por |correlación| >0.95	0.90096	0.90714
15	Selección de características con filtro CFS y función kknn	0.83818	0.82244
16	Empleo de Tomek-Links para undersampling	0.89688	0.89693
17	Empleo de Tomek-Links dos veces para undersampling	0.88310	0.88622
18	Eliminación de instancias con al menos un valor basura (-68000)	0.89994	0.90765
19	Selección de características con el método “corr” de preProcess	0.90811	0.90918
20	Eliminación de nuevo de todos los outliers extremos manteniendo el método de selección de características “corr”	0.90505	0.91071

Cuadro 2: Análisis de las subidas para el algoritmo KNN

ID Subida	Descripción	% Acierto CV	% Acierto Público	% Acierto Privado
0	Eliminando duplicados , filtro EF	80.62074	0.92189	0.91989
1	Eliminando duplicados , valores < -50000, outliers (solo por abajo) por IQR	81.00527	0.92496	0.92704
2	Eliminando duplicados , valores < -50000, outliers (solo por abajo) por IQR, filtro EF, inputar train con mice, imputar test con knn	89.9387	**0.92547**	0.92755
3	Eliminando duplicados , valores < -50000, outliers (solo por abajo) por IQR, filtro CVCF, inputar train con mice, imputar test con knn	89.78938	0.92445	**0.92959**
4	Eliminando duplicados , valores < -50000, outliers (solo por abajo) por IQR, filtro IPF, inputar train con mice, imputar test con knn	**94.0577**	0.92087	0.91989
5	Eliminando duplicados , valores < -50000, outliers (solo por abajo) por IQR, filtro EF, balanceo con Tomek links	80.62074	0.91679	0.91479
6	(Final) Eliminando duplicados , y valores < -50000	80.0129	0.92394	0.92551

Cuadro 3: Lista y breve descripción de entradas seleccionadas en la competición de SVM. Los valores incluidos son el acierto de validación cruzada sobre el set de entrenamiento, el acierto en el set de test público, y el acierto en el set de test privado. Únicamente se han incluido las entradas seleccionadas al final de la competición.