# Minería de Datos: Preprocesamiento y Clasificación
# Memoria de trabajo

Cristian González Guerrero

3 de septiembre de 2017

## Índice

# 1. Introducción

Esta es la memoria del trabajo final de la asignatura *Minería de Datos: Preprocesamiento y Clasificación*, impartida en el *Máster en Ciencia de Datos e Ingeniería de Computadores* (DATCOM). Dicho trabajo consiste en la realización de las tareas propias de la minería de datos en el ámbito del aprendizaje supervisado. Esto se materializa en la aplicación de las técnicas de preprocesamiento y clasificación adquiridas en la asignatura a un conjunto de datos de gran tamaño propuesto por los profesores.

La realización de este trabajo se ha llevado a cabo usando *RStudio* [1, 2], *en su versión 0.99.903 - R version 3.3.2 (2016-10-31)* [3]. La mayor parte de los gráficos han sido generados usando *ggplot2*[4, 5], *en su versión 2.2.1*.

Este trabajo, realizado de forma original por Cristian González Guerrero, se presenta para su valoración en la evaluación extraordinaria de Septiembre de 2017.

# 2. Presentación y visualización de los datos

El conjunto de datos de trabajo ha sido proporcionado en el fichero `datcom2016.csv`. Éste corresponde a un dataframe con una dimensión de 20000 filas y 51 columnas que no presentan valores omitidos. Esto corresponde a 20000 muestras en las que se han medido 50 propiedades. Además, estos registros han sido clasificadas según la variable `class`, que presenta dos clases: `positive` y `negative`. Con respecto a los atributos de entrada, el conjunto de datos cuenta con 9 variables categóricas y 41 variables numéricas, de las cuáles 22 toman valores enteros y las 19 restantes toman valores continuos. Un resumen de las variables puede verse en la tabla 1.

No se ha proporcionado información adicional sobre los atributos medidos, por lo que no podrán realizarse hipótesis sobre las relaciones entre éstos y la clase de los datos. Por este motivo, las columnas serán tratadas como variables de entrada genéricas, que serán nombradas con una equis mayúscula y el número de columna. Tendremos por tanto las variables de entrada `X1`, `X2`, ..., `X50`.

Al observar la frecuencia de cada clase, resulta evidente que nos encontramos ante un problema de clasificación binaria no balanceado (*imbalanced*). Como se muestra en la tabla 2 y en la figura 1, existen muchas más ocurrencias de la clase `negative` que de la clase `positive`. Esto, que a simple vista puede no parecer un problema, hará que el tratamiento de los datos tenga que hacerse con sumo cuidado, tanto en las etapas de preprocesamiento como en la clasificación. Entre las precauciones a tomar, habrá que tener en cuenta que el *accuracy* no será una buena medida en la clasificación, por lo que tendremos que usar otras medidas, como el AUC (*area under the curve*). Otra solución será usar un muestreo sobre la distribución para generar los modelos, o bien usar técnicas como *bagging* o *boosting*.

Las distribuciones de frecuencias de los atributos del conjunto de datos se presenta en las figuras 2 y 3. Como puede observarse, hay variables cuya distribución depende fuertemente de la clase, mientras que en otras, las distribuciones de ambas clases son muy parecidas. Esta dependencia de la clase hace que algunas variables sean muy interesantes, puesto que algunos de sus valores determinan en gran medida la clase de pertenencia. En cambio, las variables que presentan igual frecuencia para ambas clases deberán ser eliminadas. Este tema se trata en mayor profundidad en el epígrafe 3.5.

| Attribute | Type | Domain | Alias |
|---|---|---|---|
| separation | integer | [6, 1171] | X1 |
| propensity | numeric | [-1.89, 2.64] | X2 |
| length | integer | [50, 1244] | X3 |
| PredSS_r1_.1 | factor | {C, E, H, X} | X4 |
| PredSS_r1 | factor | {C, E, H} | X5 |
| PredSS_r1_1 | factor | {C, E, H} | X6 |
| PredSS_r2_.1 | factor | {C, E, H} | X7 |
| PredSS_r2 | factor | {C, E, H} | X8 |
| PredSS_freq_central_H | numeric | [0.00, 1.00] | X9 |
| PredSS_freq_central_E | numeric | [0.00, 1.00] | X10 |
| PredSS_freq_central_C | numeric | [0.00, 1.00] | X11 |
| PredCN_freq_central_0 | numeric | [0.00, 1.00] | X12 |
| PredRCH_freq_central_0 | numeric | [0.00, 1.00] | X13 |
| PredRCH_freq_central_1 | numeric | [0.00, 0.71] | X14 |
| PredSA_freq_central_0 | numeric | [0.00, 1.00] | X15 |
| PredSA_freq_central_4 | numeric | [0.00, 1.00] | X16 |
| PredRCH_r1_.1 | factor | {0, 1, 2, 3, 4, X} | X17 |
| PredRCH_r1 | integer | [0, 4] | X18 |
| PredRCH_r1_1 | integer | [0, 4] | X19 |
| PredRCH_r2_.1 | integer | [0, 4] | X20 |
| PredRCH_r2 | integer | [0, 4] | X21 |
| PredRCH_r2_1 | factor | {0, 1, 2, 3, 4, X} | X22 |
| PredCN_r1_.1 | factor | {0, 1, 2, 3, 4, X} | X23 |
| PredCN_r1 | integer | [0, 4] | X24 |
| PredCN_r1_1 | integer | [0, 4] | X25 |
| PredCN_r2 | integer | [0, 4] | X26 |
| PredSA_r1 | integer | [0, 4] | X27 |
| PredSA_r2_.1 | integer | [0, 4] | X28 |
| PredSA_r2 | integer | [0, 4] | X29 |
| PredSA_r2_1 | factor | {0, 1, 2, 3, 4, X} | X30 |
| PredSS_freq_global_H | numeric | [0.00, 0.96] | X31 |
| PredCN_freq_global_0 | numeric | [0.01, 0.79] | X32 |
| PredRCH_freq_global_0 | numeric | [0.06, 0.79] | X33 |
| PredRCH_freq_global_4 | numeric | [0.00, 0.63] | X34 |
| PredSA_freq_global_0 | numeric | [0.00, 0.60] | X35 |
| AA_freq_central_A | numeric | [0.00, 0.62] | X36 |
| AA_freq_central_D | numeric | [0.00, 0.60] | X37 |
| AA_freq_central_E | numeric | [0.00, 0.60] | X38 |
| AA_freq_central_I | numeric | [0.00, 0.43] | X39 |
| AA_freq_central_F | numeric | [0.00, 0.40] | X40 |
| PSSM_r1_.4_A | integer | [-11, 7] | X41 |
| PSSM_r1_.4_N | integer | [-12, 9] | X42 |
| PSSM_r1_0_D | integer | [-12, 9] | X43 |
| PSSM_r1_1_W | integer | [-13, 13] | X44 |
| PSSM_r2_0_A | integer | [-12, 7] | X45 |
| PSSM_central_.2_A | integer | [-13, 7] | X46 |
| PSSM_central_.2_T | integer | [-13, 9] | X47 |
| PSSM_central_0_H | integer | [-12, 12] | X48 |
| PSSM_central_2_D | integer | [-14, 9] | X49 |
| PSSM_central_2_V | integer | [-15, 8] | X50 |
| class | factor | {positive, negative} | class |

Cuadro 1: Resumen de los atributos y la variable clase en el conjunto de datos de estudio.

| Class | Absolute frequency ($n_i$) | Relative frequency ($f_i$) |
|---|---|---|
| positive | 5000 | 25 % |
| negative | 15000 | 75 % |

Cuadro 2: Frecuencia absoluta y relativa de las instancias de cada clase.

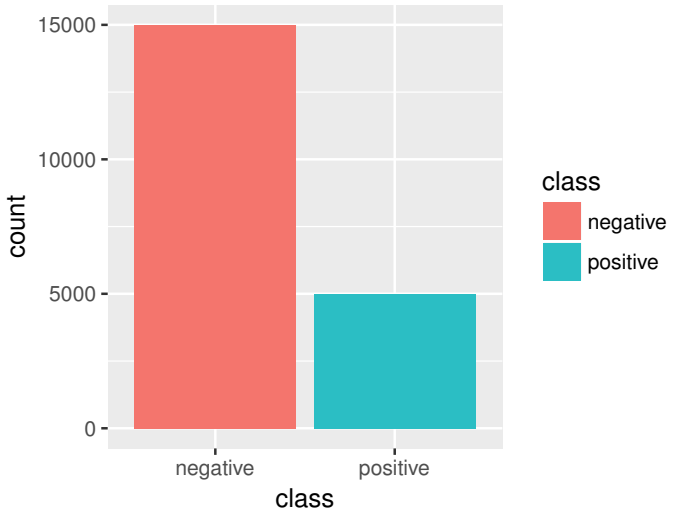



Figura 1: Conteo de ocurrencias en la variable de salida.

# 3. Preprocesamiento

En esta sección se trata el preprocesamiento de los datos, con el objetivo de conseguir unos datos de calidad que puedan servir de entrada a los algoritmos de entrenamiento en la etapa de clasificación. Los principales objetivos del preprocesamiento son:

1. Eliminar impurezas de los datos, recuperando información perdida, eliminando ruido y atendiendo a las inconsistencias que puedan presentarse en los mismos.

2. Reducir el tamaño del conjunto de datos, seleccionando atributos relevantes, eliminando registros duplicados y anomalías, discretizando variables o seleccionando instancias.

3. Obtener datos de calidad, que conduzcan a modelos de calidad.

Estos tres objetivos serán tratados en los epígrafes siguientes.

## 3.1. Particionado del conjunto de datos

En primer lugar, hay que tener en cuenta que los modelos aprendidos tendrán que ser evaluados con datos que no hayan sido vistos previamente. Para ello, se usarán datos distintos en la fase de aprendizaje y en la fase de test. Estos datos serán extraídos del conjunto de datos original usando la técnica de validación cruzada *k-fold cross validation*.

En el caso que nos ocupa, se ha decidido realizar una partición del conjunto de datos original en 10 particiones o *folds*. Este número es un valor bastante típico en los problemas de clasificación, y permite tener una cantidad manejable de sets de datos. De este modo, se usarán 9 particiones como set de entrenamiento (*training set*) y la partición restante como set de validación (*test set*). A la hora de evaluar los resultados de la clasificación se usarán 10 conjuntos de entrenamiento y de validación distintos, basados en las particiones previamente establecidas, tal y como especifica la técnica *k-fold cross validation*.

Para la creación de las 10 particiones mencionadas previamente se ha hecho uso de la función `createFolds`, disponible con el paquete `caret`. Esta función nos asegura que los datos estarán bien mezclados y que cada partición contará con una proporción similar de instancias de cada una de las

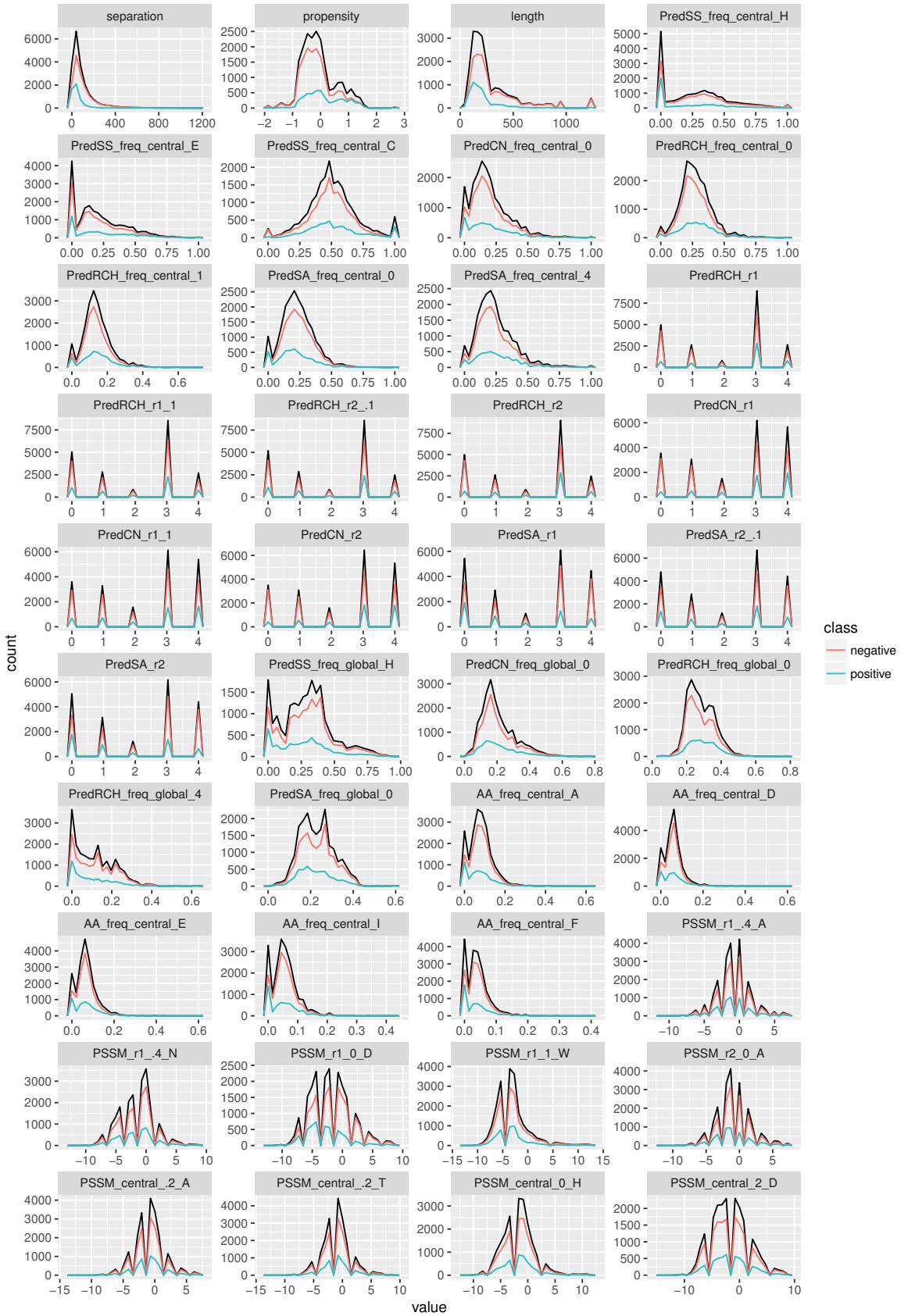



Figura 2: Distribuciones de frecuencias de las 40 primeras variables cuantitativas del set de datos. Los trazos en negro corresponden a la suma de las dos clases.

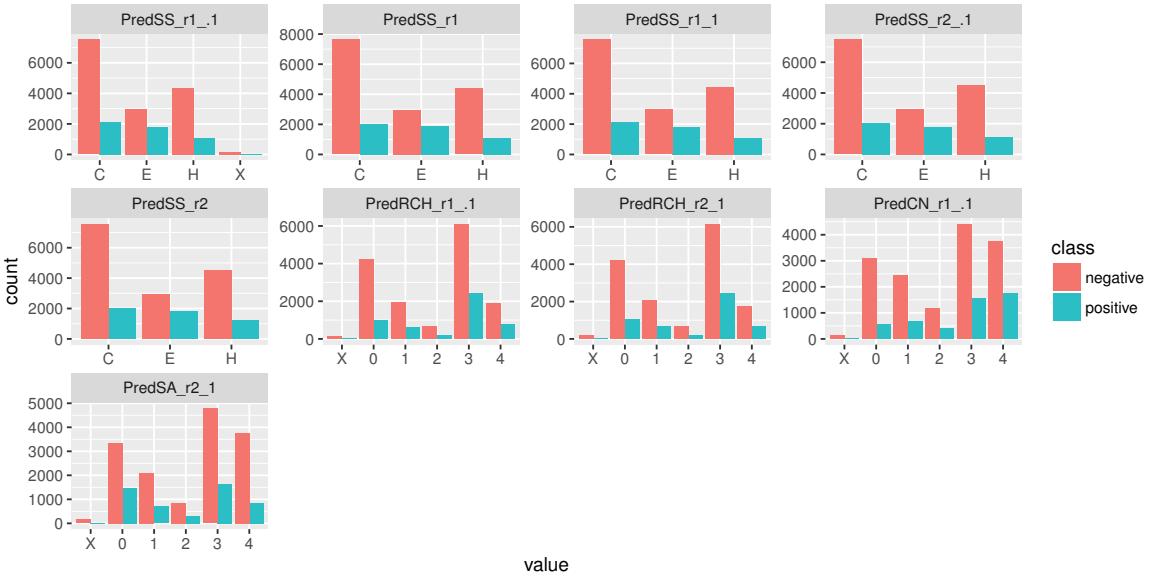



Figura 3: Distribuciones de frecuencias de las variables cualitativas del set de datos.

clases. Estas particiones han sido guardadas en archivos CSV para asegurar la repetibilidad de los procesos que se realizan a continuación. Con el mismo propósito, se proporciona un script para cargar el set de datos completo junto con estas particiones.

Cabe mencionar que ciertas etapas del preprocesamiento deberán realizarse exclusivamente sobre los datos de entrenamiento. Por ejemplo, la eliminación de anomalías o de ruido son procesos que deben realizarse exclusivamente en el *training set*, ya que de lo contrario estaríamos falseando las medidas de *performance*. En cambio, la normalización de los datos o la selección de características deberán efectuarse sobre todo el conjunto de datos.

## 3.2. Imputación de valores perdidos

Puesto que el conjunto de datos proporcionado no presenta valores perdidos, no será necesaria su imputación.

## 3.3. Detección de datos anómalos

El principal objetivo de la detección de anomalías es identificar atributos que presentan valores extraños, es decir, que no se ajustan a la norma o a la tendencia general del conjunto. Estos registros anómalos suelen representar errores en las medidas, por lo que deberán ser eliminados con el fin de evitar que interfieran con el resto de observaciones. No obstante, hay que tener en cuenta que estas anomalías pueden representar datos reales que presentan características extrañas. Este último caso deberá ser minuciosamente estudiado en el caso que nos ocupa, puesto que nos enfrentamos a un problema de clasificación no balanceada. Por este motivo, debemos cuidar que los registros de la clase minoritaria no sean eliminados por considerarse anómalos.

Un registro puede ser considerado anómalo si alguno de sus atributos presenta un valor extremo, o si el conjunto de los atributos presenta una combinación anómala de valores. En el primer caso el dato en cuestión es denominado *valor atípico* o *outlier*.

Estos dos casos deberán ser estudiados de forma separada, con el objetivo de eliminar tantas instancias anómalas como sea posible.

### 3.3.1. Datos anómalos en una variable

La detección de datos anómalos en una variable se ha llevado a cabo usando la función `rm.outlier` del paquete `outliers`. Con esta función se ha analizado cada variable continua presente en el dataset, eliminando los registros que presentaban valores anómalos. El criterio por el cuál se ha decidido

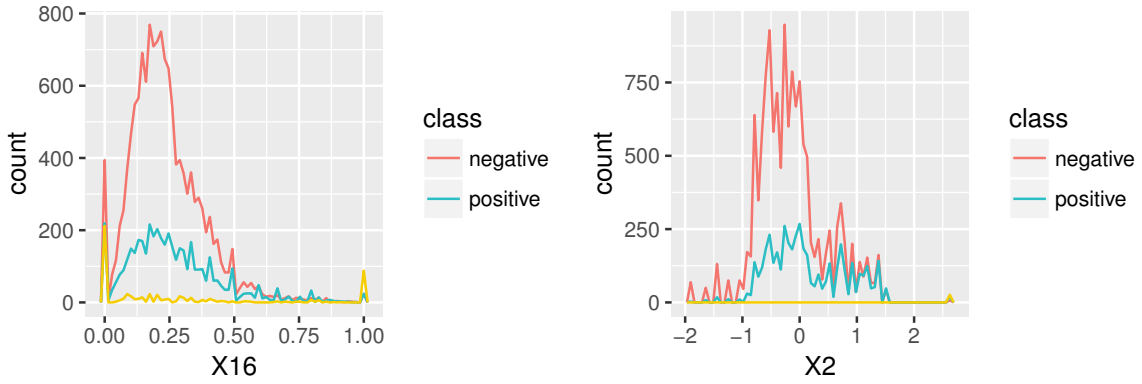



Figura 4: Ejemplos de detección de anomalías en una variable. Las líneas anaranjadas representan la distribución de las anomalías detectadas.

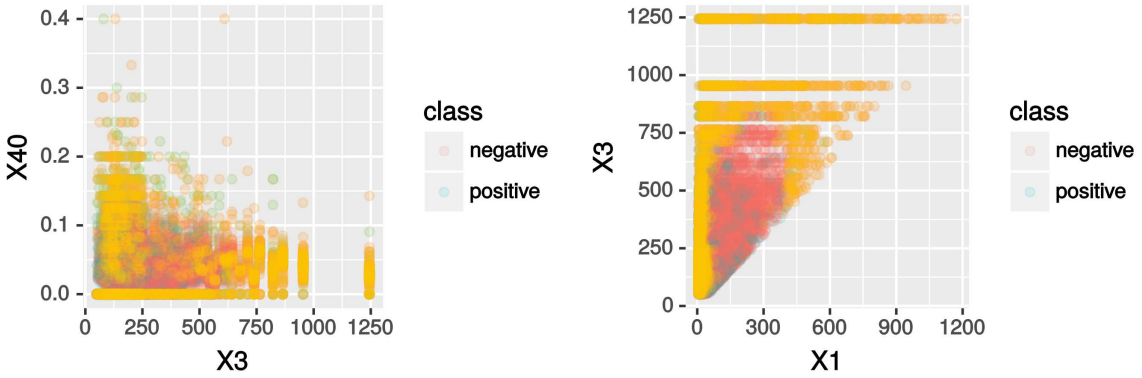



Figura 5: Ejemplos de detección de anomalías en varias variables. Los puntos anaranjados representan las anomalías detectadas.

eliminar los registros anómalos en vez de imputarles un valor promedio se basa en que nuestro conjunto de datos cuenta con numerosas filas. Por este motivo, la eliminación de alguna de ellas no supone una gran pérdida de información, siendo preferible a imputar otro valor de cara a la detección de anomalías en múltiples variables. Además, la eliminación de filas puede ayudar a reducir el tamaño del conjunto de datos, acelerando el proceso de entrenamiento en la fase de clasificación.

Como puede observarse en la figura 4, la mayoría de anomalías detectadas están en los extremos de las distribuciones, como cabía esperar.

### 3.3.2. Datos que no se ajustan a la tendencia general de los datos

La detección de datos anómalos en múltiples variables se ha realizado mediante la función `uni.plot` del paquete `mvoutlier`. Para llevarla a cabo, ha sido necesario seleccionar 10 variables numéricas. En este caso, se han elegido las 10 variables cuantitativas más representativas, según el test $\chi^2$ de significancia con respecto a la clase proporcionado por la función `chi.squared` del paquete `FSelector`. Esta selección es necesaria, puesto que la función `uni.plot` no admite más de diez atributos.

Como puede observarse en la figura 5, la mayoría de las anomalías detectadas se encuentran cerca de los bordes de las distribuciones.

### 3.3.3. Análisis de la eliminación de anomalías

Una vez realizado el proceso de eliminación de registros anómalos es necesario comprobar que la proporción de la clase de salida no se haya visto modificada. Esto es crucial en clasificación no balanceada, puesto que los datos asociados a la clase minoritaria podrían ser detectados como anómalos.

| Class | $f_i$ prior outlier elimination | $f_i$ after outlier elimination |
|---|---|---|
| positive | 25.0 % | 24.15 % |
| negative | 75.0 % | 75.85 % |

Cuadro 3: Frecuencias relativas de cada clase antes y después de la eliminación de anomalías.

En particular, es muy interesante que los atributos asociados a la clase minoritaria produzcan valores extremos, pues facilitaría la tarea de clasificación.

En el caso que nos ocupa, la proporción de ocurrencias de cada clase no se ha visto significativamente modificada, como puede observarse en la tabla 3. Algo que llama la atención es que se han eliminado alrededor del 17 % de los registros, lo cuál puede afectar significativamente a la etapa de clasificación.

## 3.4. Discretización de los datos

La discretización de los datos supone llevar una serie de valores, normalmente de una variable continua, a valores discretos. Este proceso simplifica los datos, de forma que se tiene una información más sencilla sobre cada variable. Además, esta nueva información suele ser más relevante que los valores iniciales, puesto que los puntos de corte son seleccionados de forma que las variables discretas resultantes recojan la información más relevante posible acerca de cada tramo discretizado.

El proceso de discretización es, además, un requisito para algunos métodos de clasificación, que sólo pueden trabajar con valores discretos. Un ejemplo de esto son las reglas de asociación.

La implementación de la discretización de los datos se ha llevado a cabo usando el la función `disc.Topdown` del paquete `discretization`. Esta función realiza una discretización top-down usando alguno de los algoritmos que implementa. En el caso que nos ocupa, se ha optado por usar el algoritmo CAIM (*class-attributo interdependence maximization*), que selecciona los puntos de corte atendiendo a la dependencia entre la variable clase y la discretización a realizar sobre el atributo. De esta forma, se esperan conseguir intervalos que recojan mucha información sobre la clase a la que pertenece el registro.

En nuestro caso, la discretización ha dado como resultado variables discretas con dos valores únicamente. Esto se pone de manifiesto en las figuras 6 y 7, que muestran el resultado de la discretización en dos pasos:

En la figura 6 puede observarse la elección de los puntos de corte en algunas de las variables del conjunto de datos. Como puede observarse, estos puntos de corte han sido seleccionados donde la proporción entre la clase `positive` y `negative` tiene un cambio más importante.

La figura 7 muestra la proporción de cada una de las clases en las variables ya discretizadas. Como puede observarse, la diferencia de proporción entre cada clase es bastante significativa. De este modo, cabría esperar que usando simplemente estas variables en la entrada de un algoritmo de clasificación se podrían conseguir un resultados bastante razonables.

### 3.4.1. Valoración de la discretización

Dado que en el problema tratado existen ya datos discretos, cabe cuestionarse si una discretización de esos datos realmente beneficiará en la clasificación o bien si simplificará demasiado las variables de entrada. La figura 8 muestra los efectos de la discretización sobre cuatro variables que toman pocos valores enteros. Como puede observarse al comparar las densidades de puntos de cada clase en los dos diagramas de dispersión, los valores discretizados parecen recoger muy bien la esencia de las distribuciones originales.

## 3.5. Selección de características

La selección de características es una etapa fundamental en la reducción del tamaño del conjunto de datos. Consiste en detectar las características más relevantes para el proceso de clasificación. De

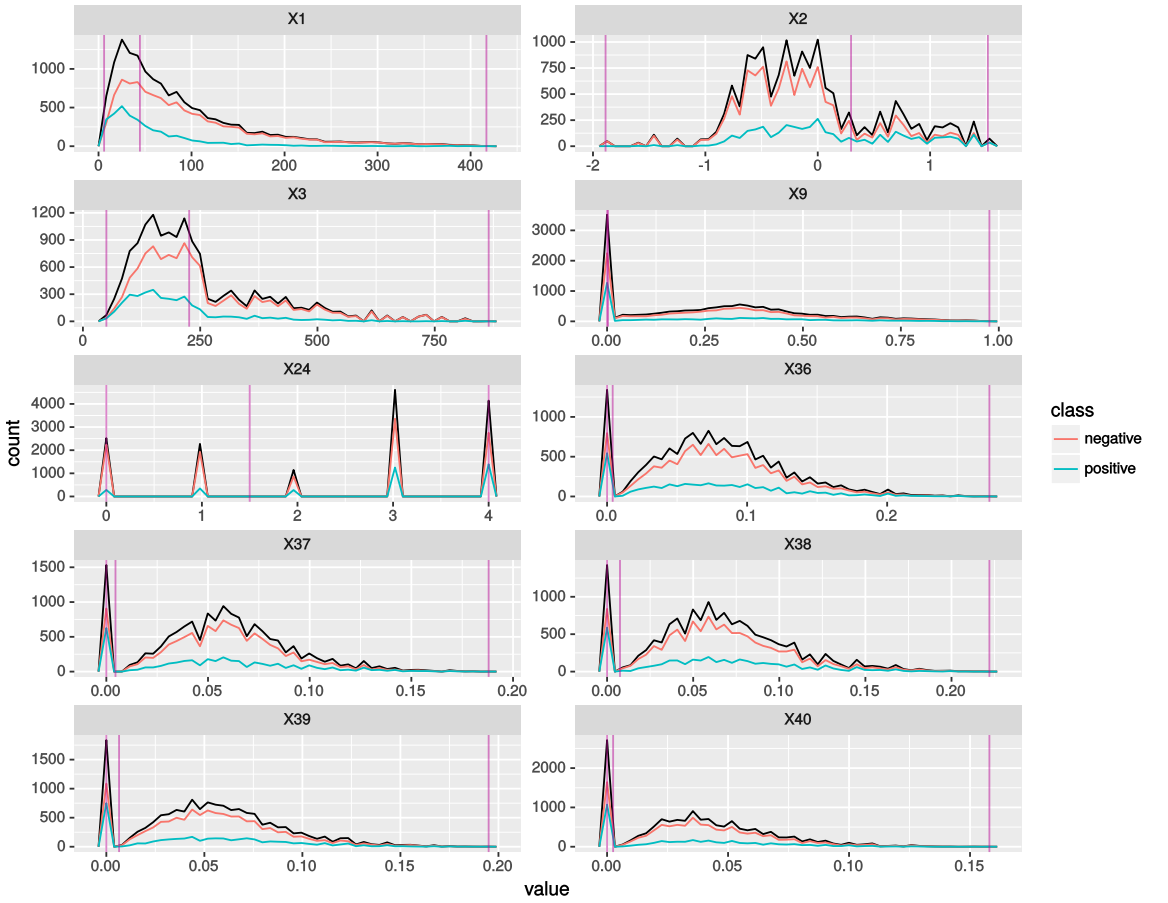



Figura 6: Distribución de frecuencias de las 10 variables más determinantes junto con los puntos de corte establecidos por la discretización. Las 10 variables más determinantes se han obtenido con un test $\chi^2$.

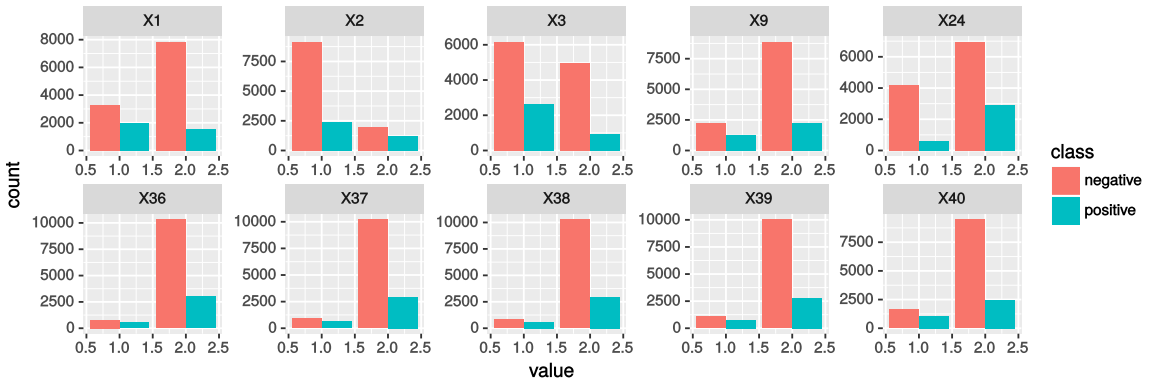
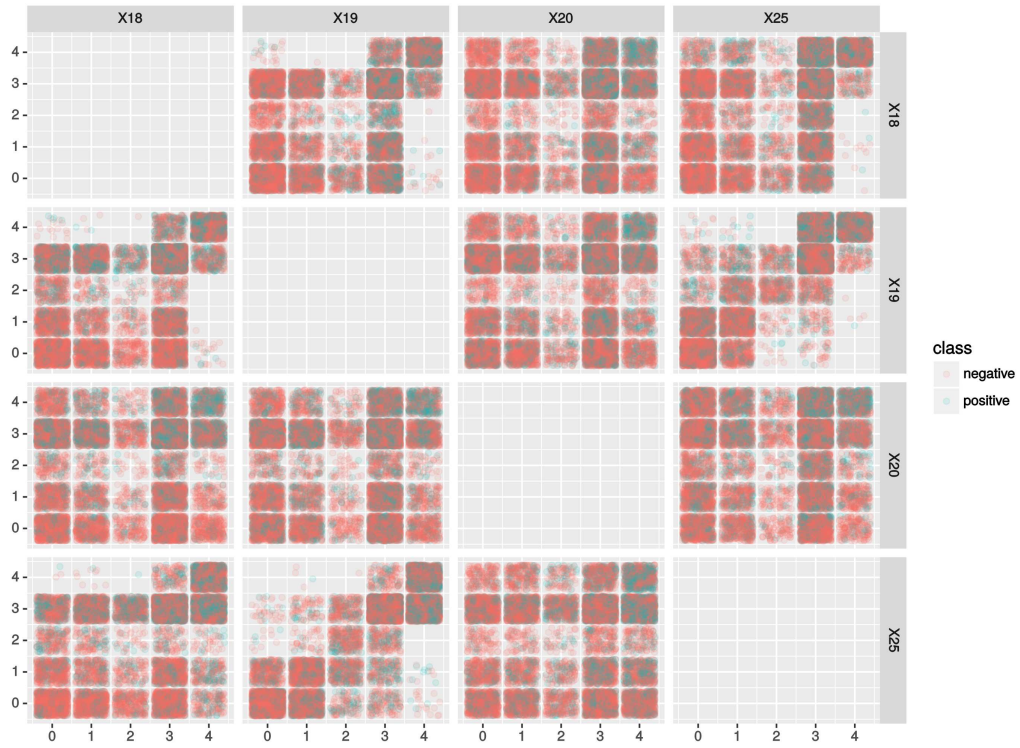
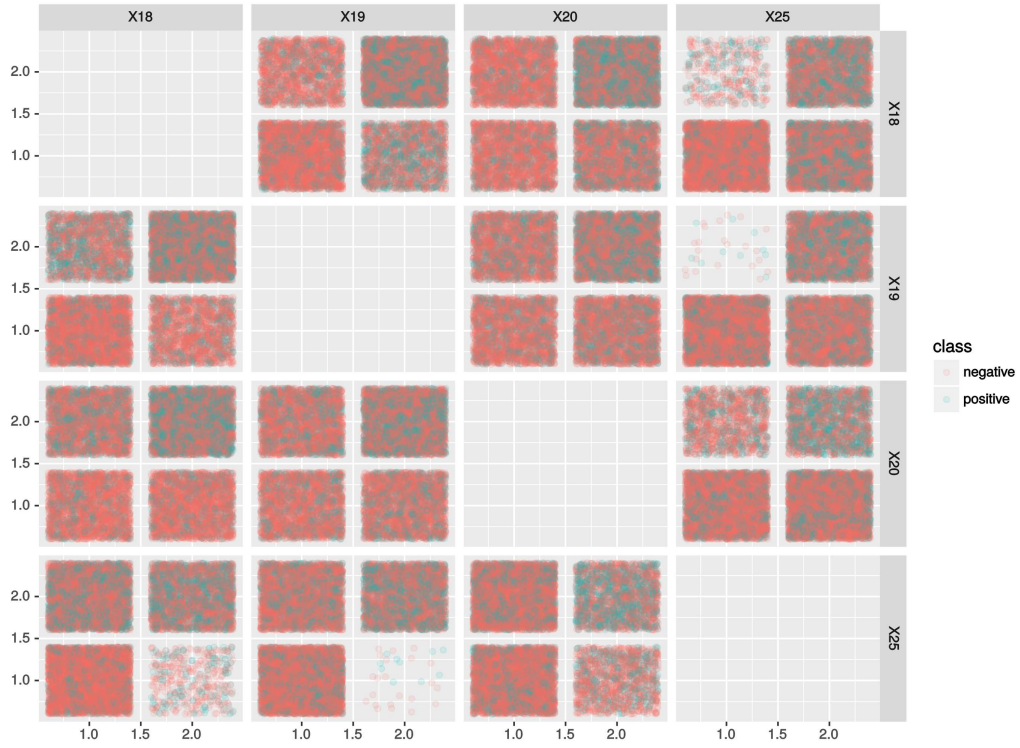



Figura 7: Frecuencias de las 10 variables más determinantes discretizadas. Las 10 variables más determinantes se han obtenido con un test $\chi^2$.




(a) Diagramas de dispersión antes de la discretización.




(b) Diagramas de dispersión después de la discretización.

Figura 8: Efectos de la discretización sobre algunas de las variables de entrada enteras. Los diagramas de dispersión se han construido usando la opción *jitter*, que esparce los valores para permitir una mejor visualización.

| Attribute | Importance |
|---|---|
| X1 | 0.277 |
| X3 | 0.197 |
| X24 | 0.191 |
| X2 | 0.188 |
| X26 | 0.182 |
| X27 | 0.180 |
| X40 | 0.177 |
| X5 | 0.174 |
| X18 | 0.172 |
| X29 | 0.172 |
| X6 | 0.166 |
| X21 | 0.166 |
| X43 | 0.162 |
| X4 | 0.160 |
| X39 | 0.160 |
| X8 | 0.158 |
| X9 | 0.157 |

| Attribute | Importance |
|---|---|
| X37 | 0.156 |
| X7 | 0.153 |
| X38 | 0.143 |
| X36 | 0.136 |
| X10 | 0.132 |
| X23 | 0.130 |
| X13 | 0.129 |
| X12 | 0.124 |
| X34 | 0.115 |
| X25 | 0.114 |
| X14 | 0.109 |
| X15 | 0.104 |
| X11 | 0.100 |
| X17 | 0.100 |
| X30 | 0.095 |
| X16 | 0.089 |
| X22 | 0.086 |

| Attribute | Importance |
|---|---|
| X19 | 0.083 |
| X31 | 0.081 |
| X28 | 0.079 |
| X20 | 0.076 |
| X45 | 0.072 |
| X33 | 0.070 |
| X35 | 0.053 |
| X32 | 0.000 |
| X41 | 0.000 |
| X42 | 0.000 |
| X44 | 0.000 |
| X46 | 0.000 |
| X47 | 0.000 |
| X48 | 0.000 |
| X49 | 0.000 |
| X50 | 0.000 |

Cuadro 4: Importancia de cada variable según el test $\chi^2$ de significancia con respecto a la clase. Los valores señalados en rojo no aportan información sobre la clase.

este modo, se eliminan atributos que no tienen peso en la decisión de la clase, así como atributos redundantes, cuya información puede ser extraída de un atributo similar. Existen varias aproximaciones para abordar la selección de características.

### 3.5.1. Aproximación filter

Se basa en aplicar medidas estadísticas sobre los datos con el objetivo de detectar aquellos atributos que son irrelevantes o redundantes. El resultado es una lista de atributos y el peso o importancia de cada uno de ellos.

**Atributos irrelevantes** Hemos aplicado la función `chi.squared` del paquete `FSelector` para obtener un orden de significancia de las variables con respecto a la clase. La tabla 4 muestra los resultados de esta aproximación. Como puede observarse, hay nueve atributos que no aportan información en este problema de clasificación, por lo que pueden ser eliminados sin más consideraciones.

La aportación de información sobre la clase que un atributo proporciona puede entenderse como una diferencia en las distribuciones de probabilidades condicionales con respecto a la clase. Para verificar los resultados ofrecidos por la función `chi.squared`, hemos aplicado un test de Kolmogorov-Smirnov, comparando, para cada atributo, las distribuciones asociadas a cada clase. Con este test hemos podido comprobar que no puede refutarse la hipótesis de que las variables `X46`, `X47`, `X48`, `X49` y `X50` presenten distribuciones significativamente diferentes para cada clase con más de un 90 % de confianza ($p$-value$< 0{,}1$).

Estas conclusiones también pueden obtenerse a partir de medidas basadas en la entropía, que miden directamente la ganancia de información que aporta cada variable. Para esto, se ha aplicado la función `information.gain` del paquete `FSelector`. Los resultados de aplicar esta función pueden observarse en la tabla 5. Como puede comprobarse, el orden de las variables es similar al proporcionado por el test $\chi^2$, aunque algunas están cambiadas de orden. Otra posible medida de entropía es la relación de ganancia de información (*information gain ratio*), proporcionada por la función `gain.ratio`. Esta medida trata de evitar el sesgo producido por la ganancia de información, en especial cuando se trata de evaluar la importancia de un atributo en un árbol de decisión. Para ello, divide la ganancia de información entre la dispersión de los valores del atributo [**?**]. Esta medida proporciona resultados muy distintos a los conseguidos con la ganancia de información o el test $\chi^2$, por lo que habrá que considerar cuál de ellos proporciona una mejor aproximación al caso estudiado.

| Attribute | Importance |
|---|---|
| X1 | 0.057 |
| X3 | 0.029 |
| X24 | 0.028 |
| X26 | 0.026 |
| X2 | 0.024 |
| X27 | 0.024 |
| X18 | 0.023 |
| X29 | 0.022 |
| X40 | 0.021 |
| X21 | 0.021 |
| X5 | 0.021 |
| X43 | 0.019 |
| X6 | 0.019 |
| X4 | 0.018 |
| X8 | 0.017 |
| X39 | 0.017 |
| X9 | 0.017 |

| Attribute | Importance |
|---|---|
| X37 | 0.016 |
| X7 | 0.016 |
| X38 | 0.013 |
| X23 | 0.013 |
| X36 | 0.012 |
| X10 | 0.011 |
| X13 | 0.011 |
| X12 | 0.010 |
| X25 | 0.009 |
| X34 | 0.009 |
| X14 | 0.008 |
| X17 | 0.007 |
| X15 | 0.007 |
| X30 | 0.007 |
| X11 | 0.007 |
| X22 | 0.005 |
| X16 | 0.005 |

| Attribute | Importance |
|---|---|
| X19 | 0.005 |
| X28 | 0.005 |
| X31 | 0.005 |
| X20 | 0.004 |
| X45 | 0.004 |
| X33 | 0.004 |
| X35 | 0.002 |
| X32 | 0.000 |
| X41 | 0.000 |
| X42 | 0.000 |
| X44 | 0.000 |
| X46 | 0.000 |
| X47 | 0.000 |
| X48 | 0.000 |
| X49 | 0.000 |
| X50 | 0.000 |

Cuadro 5: Importancia de cada variable según la ganancia de información. Los valores señalados en rojo no aportan información sobre la clase.

Un resumen de los resultados obtenidos puede verse en la figura 11. En la misma, se representa la importancia de cada atributo atendiendo a las métricas mencionadas anteriormente. Los atributos han sido ordenados de más a menos importantes, en función de la suma de los distintos índices de importancia. Para realizar esta suma, cada índice de importancia se ha normalizado, con el objetivo de que todos tuvieran el mismo peso. Resulta llamativo que los primeros cuatro atributos (`X1`, `X2`, `X3` y `X24`) presentan un buen resultado en todas las métricas. A partir del quinto atributo (`X15`) en adelante, suele haber una o dos métricas con buenos resultados, siendo el resto bastante pobres. Cabe plantearse si es más conveniente atender a la suma de los distintos índices o a alguno en particular, puesto que cada uno de ellos afectará de forma distinta a cada algoritmo de clasificación. Siendo conservadores, concluimos que convendrá tomar los primeros 30 atributos, puesto que a partir del trigésimo primero (`X34`) todas las métricas producen resultados más bien pobres.

**Redundancias** Algunos de los atributos aportan información redundante, que puede obtenerse a partir de otros atributos. Por este motivo, para simplificar aún más el conjunto de datos, podemos prescindir de algunos de estos atributos. Hemos usado la función `findCorrelation` del paquete `caret` para eliminar aquellas columnas que presentan una fuerte correlación. De esta forma, hemos eliminado una variable de cada pareja de variables que presente una correlación mayor a 0,8, según el coeficiente de correlación lineal de Pearson. El resultado es un conjunto de datos con 25 atributos, que se muestra en la tabla 6.

### 3.5.2. Aproximación wrapper

Otra aproximación a la selección de características es la aproximación *wrapper*. En este tipo de aproximación, se prueban modelos diferentes con distintos subconjuntos de atributos, comprobando qué atributos producen un mejor modelo en función de alguna métrica. No se han llevado a cabo pruebas con este tipo de aproximación en este trabajo.

## 3.6. Detección de ruido

La etapa de detección de ruido pretende detectar y, en su caso eliminar, los registros que presentan una clase que no se corresponde con la clase esperada, atendiendo a los atributos que dicho registro presenta. Existen varias formas de realizar este proceso. Uno de los más usados es el denominado *Iterative Partitioning Filter*, que es muy útil cuando el problema tratado supone manejar grandes


Figura 9: Importancia de cada atributo según las distintas métricas. El color del borde de cada rectángulo indica el tipo de dato.

| Attribute | Type | Domain |
|---|---|---|
| X1 | integer | [6, 417] |
| X2 | numeric | [-1.89, 1.51] |
| X3 | integer | [50, 865] |
| X5 | factor | {C, E, H} |
| X6 | factor | {C, E, H} |
| X8 | factor | {C, E, H} |
| X9 | numeric | [0.00, 0.98] |
| X10 | numeric | [0.00, 0.96] |
| X12 | numeric | [0.00, 0.92] |
| X13 | numeric | [0.00, 0.89] |
| X14 | numeric | [0.00, 0.67] |
| X15 | numeric | [0.00, 0.90] |
| X21 | integer | [0, 4] |

| Attribute | Type | Domain |
|---|---|---|
| X22 | factor | {0, 1, 2, 3, 4, X} |
| X23 | factor | {0, 1, 2, 3, 4, X} |
| X26 | integer | [0, 4] |
| X27 | integer | [0, 4] |
| X29 | integer | [0, 4] |
| X30 | factor | {0, 1, 2, 3, 4, X} |
| X36 | numeric | [0.00, 0.27] |
| X37 | numeric | [0.00, 0.19] |
| X38 | numeric | [0.00, 0.22] |
| X39 | numeric | [0.00, 0.20] |
| X40 | numeric | [0.00, 0.16] |
| X43 | integer | [-12, 9] |
| class | factor | {positive, negative} |

Cuadro 6: Variables más relevantes para esta tarea de clasificación, después de la etapa de preprocesamiento. En los rangos mostrados se ha tenido en cuenta la eliminación de anomalías, pero no la discretización.

| Class | $f_i$ prior Noise Elimination | $f_i$ after IPF | $f_i$ after ENN |
|---|---|---|---|
| positive | 24.15 % | 8.04 % | 24.60 % |
| negative | 75.85 % | 91.96 % | 75.40 % |

Cuadro 7: Frecuencias relativas de cada clase antes y después de la eliminación de ruido.

cantidades de datos. Este método realiza varias particiones del conjunto de datos, genera un modelo sencillo en cada una de estas particiones y, a continuación, detecta instancias anómalas en el conjunto de datos. Para este último paso, los distintos modelos creados aportan una predicción de la clase a la que pertenece la instancia. Si la clase real del dato no se ajusta a lo predicho por la mayoría de los modelos, puede considerarse que el dato es ruidoso.

Hemos aplicado la función `IPF` del paquete `NoiseFiltersR` sobre el conjunto de datos obtenido en el paso anterior. El resultado, sin embargo, ha sido una descompensación aún más pronunciada de la clase minoritaria. La proporción entre estas clases se muestra en la tabla 7. Esta descompensación es inaceptable, y pone de manifiesto que el método aplicado no funciona correctamente con clases no balanceadas. Por este motivo, habrá que aplicar algún otro método para la eliminación de ruido.

El método seleccionado ha sido Edited Nearest Neighbors [6]. Este método elimina una instancia si ésta presenta una clase distinta a la de al menos dos de sus tres vecinos más cercanos. Se trata, por tanto, de una técnica de submuestreo, que pretende eliminar instancias de la clase mayoritaria muy cercanas al espacio de la clase minoritaria. El resultado de aplicar esta técnica para eliminación de ruido puede observarse en la tabla 7 y en la figura 10. En esta última, podemos observar que el ruido eliminado se superpone claramente con la clase minoritaria.

# 4. Clasificación

Una vez mejorados los datos a través del preprocesamiento, hay que ocuparse de la tarea de clasificación. En primer lugar, tendremos que definir la medida de bondad que deberá usarse en la verificación del modelo. Es conveniente que esta medida sea también usada durante el entrenamiento, de forma que se obtenga el mejor modelo posible. Hay que destacar que la métrica usada debe ser significativa para el problema de clasificación no balanceada que nos ocupa.

A continuación, exploraremos diversos métodos de clasificación, viendo sus ventajas y sus inconvenientes, así como la medida en que cada método produce un modelo adecuado que explique los datos

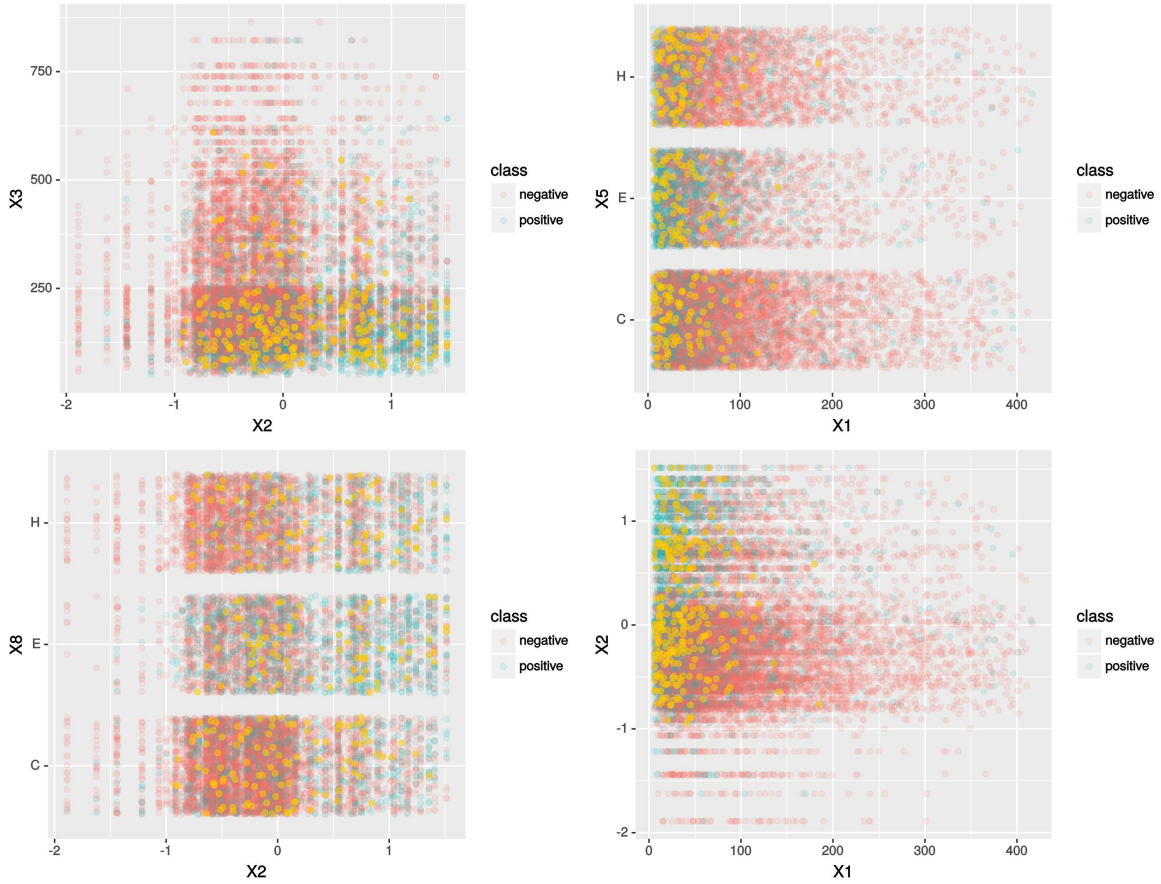



Figura 10: Diagramas de dispersión de algunas de las variables del conjunto de datos tras el preprocesamiento. Los puntos anaranjados representan instancias con ruido eliminadas.

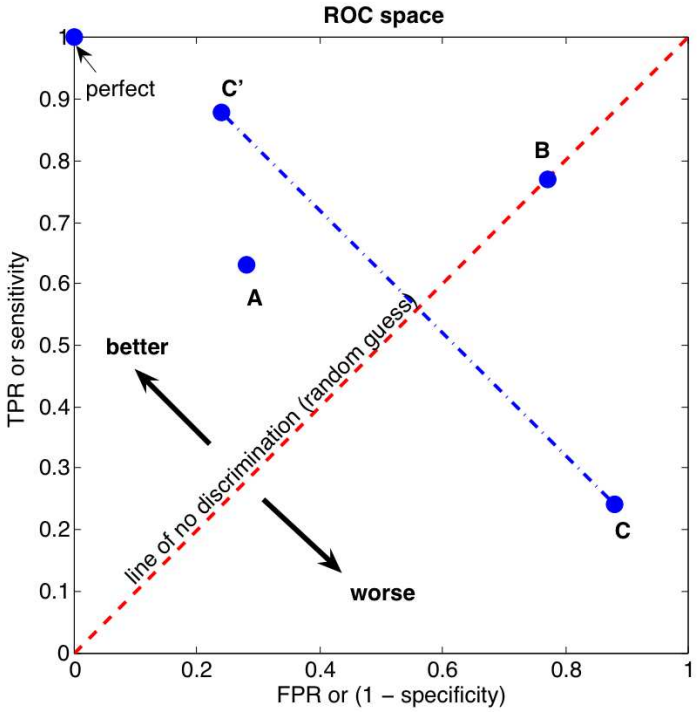



Figura 11: Curva ROC genérica. En rojo se representa el peor caso posible, en que la incertidumbre en la detección es total (clasificación aleatoria). La línea que pasa por la esquina superior izquierda representa el mejor caso, en que la sensibilidad y la especificidad son máximas. [7]

correctamente. Estos métodos también deben estar preparados para afrontarse a una clasificación no balanceada.

Comenzaremos, por tanto, con unos apuntes sobre este tipo particular de clasificación.

## 4.1. Apuntes sobre clasificación no balanceada

Puesto que el problema estudiado se trata de una clasificación no balanceada, ha habido que usar métricas de bondad que se ajustaran a esta situación. En concreto, se ha usado el AUC (*area under the curve*), que se refiere al área bajo la curva ROC (*receiver operating charasteristic*). Esta curva se construye representando la sensibilidad (*sensitivity*, o *true positive rate*) con respecto al opuesto de la especificidad ($1-$*specificity*, o *false positive rate*). El receptor, o, en nuestro caso, el clasificador, funcionará mejor cuanto más pronunciada sea la curva, y, por tanto, cuanto mayor sea el área bajo la misma (AUC). La figura 11 muestra una representación gráfica de esta curva, en la que se señalan también el mejor y el peor caso posible.

El uso del AUC soluciona el gran problema de otras medidas de bondad como el *accuracy*. Esta última, en concreto, es una medida muy poco deseable en clasificación no balanceada. Por ejemplo, en el caso estudiado, un modelo que siempre predijera la clase mayoritaria tendría un 75 % de acierto.

Otra forma de solucionar este problema es tratar de equilibrar las clases, convirtiendo el problema en una clasificación balanceada. Para esto, existen varios métodos, que pueden clasificarse como se expone a continuación.

### 4.1.1. Métodos basados en instancia (instance-based methods)

Estos métodos transforman el set de datos de entrenamiento para luego aplicar algoritmos de clasificación tradicionales. De esta forma, pretenden transformar el dataset original, que presenta muchas más observaciones de una clase que de la otra, en un dataset con un número parecido de observaciones de ambas clases. Existen dos formas principales de hacer esto.

**Submuestreo de las clases mayoritarias (downsampling)** Se trata de eliminar observaciones de la clase mayoritaria, aumentando así la proporción de observaciones de la clase minoritaria sobre el total.

Para ello, puede eliminarse la observación más cercana a cada observación de la clase minoritaria (Tomek-links). De esta forma se crea una separación clara entre los items de cada clase. También existen otros algoritmos más complejos, como *Condensed Nearest Neighbor (CNN)* o *Neighborhood Cleaning Rule (NCL)*, que están igualmente enfocados a resaltar las diferencias entre las dos clases.

Estos algoritmos son ampliamente utilizados en detección de anomalías, donde existe una gran diferencia en el número de instancias de las clases «normales» y «anómalas».

**Sobremuestreo de las clases minoritarias (upsampling)** Se trata de introducir observaciones artificiales, con el objetivo de aumentar la proporción de instancias de la clase minoritaria.

La técnica más usada es probablemente SMOTE (*Synthetic Minority Over-sampling Technique*) [8]. Esta técnica selecciona al azar una observación de la clase minoritaria. A continuación, selecciona los $k$ elementos más cercanos de la misma clase. Finalmente, genera items sintéticos en la línea que une la observación inicial y sus vecinos.

#### 4.1.2. Métodos basados en algoritmo (algorithm-based methods)

Estos métodos modifican los algoritmos de clasificación para dar más peso a las clases minoritarias, evitando así alterar el conjunto de datos. Existen varias formas de conseguirlo:

**Métodos sensitivos al coste** Consisten en asignar un coste elevado a los datos de las clases minoritarias, haciéndolos más relevantes y consiguiendo una mejor clasificación. Estos métodos sólo son aplicables a un grupo muy determinado de algoritmos que admiten pesos en el aprendizaje del modelo.

**Bagging** Consiste en aplicar el algoritmo de clasificación varias veces, cada vez con una muestra del dataset diferente, que puede incluir elementos duplicados. Estas clasificaciones deberán producir modelos con sobreajuste, que luego serán promediados. De este modo, en cada paso se conseguirá un modelo más efectivo, que habrá contado con la presencia de más observaciones de la clase minoritaria durante su construcción, mejorando así la calidad de la clasificación.

**Boosting** Consiste en aplicar el algoritmo de clasificación varias veces, dando cada vez más peso a los elementos que no fueron bien clasificados. De este modo, en cada paso el algoritmo se conseguirá un modelo más efectivo, que habrá sido entrenado dando más importancia a los elementos minoritarios.

### 4.2. Elección del algoritmo de clasificación

De entre los múltiples algoritmos y métodos de clasificación estudiados en la asignatura, empezaremos probando un modelo sencillo pero muy efectivo, como son los árboles de decisión, para pasar luego a otros modelos más complejos. Entrenaremos nuestros árboles de decisión atendiendo a tres criterios fundamentales.

1. Se hará uso de los métodos basados en instancia explicados previamente para tratar de rectificar el sesgo producido por la desproporción entre las clases. Con estos métodos se equilibrará la proporción entre las clases en el set de entrenamiento. Los resultados producidos por los modelos aprendidos a través de estas técnicas se compararán entre sí. También se compararán los resultados obtenidos al entrenar el modelo con los datos originales.

2. La variables de entrada en el entrenamiento serán tomadas a partir de las distintas etapas del preprocesamiento. Con esto se pretende verificar que cada una de las etapas mejora los resultados finales. En el caso de que una etapa empeore los resultados finales, será necesario ajustar los parámetros de la esta etapa, o bien prescindir de la misma.

3. El entrenamiento del modelo se llevará a cabo siguiendo la técnica de *repeated k-folds cross validation*, de forma que se repetirá la validación cruzada un número $n$ de veces, usando un particionado distinto cada vez. De esta forma, se buscará el modelo que mejor ajuste las $kn$ iteraciones. En todos los casos, la curva ROC será la métrica para medir el error, y el conjunto de test final, sobre el que se mide el AUC ni se preprocesa ni se usa en el entrenamiento.

Estos criterios también serán usados a la hora de generar modelos más complejos. No obstante, usaremos los resultados proporcionados por los árboles de decisión simples para ajustar parámetros en el preprocesamiento.

#### 4.2.1. Árboles de decisión

Para entrenar los árboles de decisión, nos ayudaremos del paquete `rpart`, que usaremos a través de las funciones de `caret`. De este modo, la mayor parte del código que desarrollemos para el entrenamiento de los árboles de decisión, también podrá ser usado en el entrenamiento de otros modelos, al ofrecer `caret` una interfaz común para todos ellos. Además, con caret se automatiza el proceso de validación cruzada y de poda (*prune*), gracias a la parametrización del entrenamiento que nos ofrece este paquete a través de la función `trainControl`.

Después del entrenamiento de los árboles de decisión siguiendo los parámetros indicados en la tabla 8, se ha procedido a analizar y representar los resultados. La figura 12 muestra una representación gráfica de los árboles generados a partir de los datos originales y de los datos después de haber realizado un sobremuestreo de la clase minoritaria. Como puede comprobarse, estos árboles son muy distintos. De entre estos dos árboles, solo el que ha sido obtenido a partir de los datos con sobremuestreo es capaz de detectar positivos en varias regiones del espacio. Esto tiene una gran relación con los resultados expresados en la tabla 9.

Esta muestra los resultados obtenidos con diversas medidas de bondad de la clasificación para los distintos datos de entrada ensayados. Como puede comprobarse, existen contradicciones entre las métricas AUC y *accuracy*: en el caso de los datos originales, el *accuracy* es máximo y el AUC es mínimo, justo al contrario que en el caso de submuestreo. Esto se debe a que el *accuracy* no tiene en cuenta la descompensación entre las clases mayoritaria y minoritaria al medir el número de aciertos. El AUC, en cambio, tiene en cuenta estas diferencias. Las curvas ROC obtenidas pueden observarse en la figura 13.

Los parámetros de la etapa de preprocesamiento mostrados en la tabla 8 han sido seleccionados para maximizar el AUC. Esta selección se ha llevado a cabo de forma manual, modificando los distintos parámetros y comprobando su efecto sobre cada uno de los modelos aprendidos. Los parámetros de la selección de datos, en cambio, han sido elegidos de forma arbitraria, ya que la partición elegida no debería afectar significativamente a la selección de parámetros. Una mejor aproximación habría sido iterar sobre las distintas particiones en el proceso de selección, tomando una tabla final de resultados para cada etapa de preprocesamiento. De esta forma, se podría validar, a través de un test de los rangos con signo de Wilcoxon, si existe una mejora significativa al aplicar cada etapa. Sin embargo, esta aproximación es computacionalmente muy costosa, y habría que repetirla para cada parámetro ensayado.

De entre las pruebas realizadas, ha llamado la atención que la eliminación de ruido proporcionada por IPF y por ENN lleva a resultados muy parecidos, que son, a su vez, muy parecidos a los resultados obtenidos sin la eliminación de ruido. Este es un hecho curioso, dado que las salidas que arrojan ambos métodos son muy diferentes entre sí. Sospechamos que esto puede estar sucediendo en el caso particular de los árboles de decisión, por lo que se ensayará la eliminación de ruido en otros modelos. Por otro lado, la eliminación de anomalías en múltiples variables ha llevado en todos los casos a modelos con menos capacidad predictiva. Sospechamos que esto se debe a un error en la parametrización de esta etapa. Tal vez las distribuciones de los datos no son las esperadas por la función encargada de detectar las anomalías; no se están seleccionando correctamente las variables de entrada; la métrica usada no sea la correcta; o se están detectando como anómalos datos que están relativamente cerca al centro de su distribución. En último lugar, la selección de atributos parece estar funcionando correctamente, aunque los atributos redundantes parecen ofrecer algo de información en la etapa de clasificación, por lo que se ha optado por no eliminarlos.

#### 4.2.2. Máquinas de soporte vectorial

Las máquinas de soporte vectorial *(support vector machines*, o *SVM)* son modelos de aprendizaje supervisado cuyo método de clasificación se basa en encontrar los hiper-planos que mejor separan las distintas clases. Estos hiper-planos se definen en espacios vectoriales de alta dimensionalidad, cuyas componentes están relacionadas con los atributos de los datos y una función de kernel. Esta función

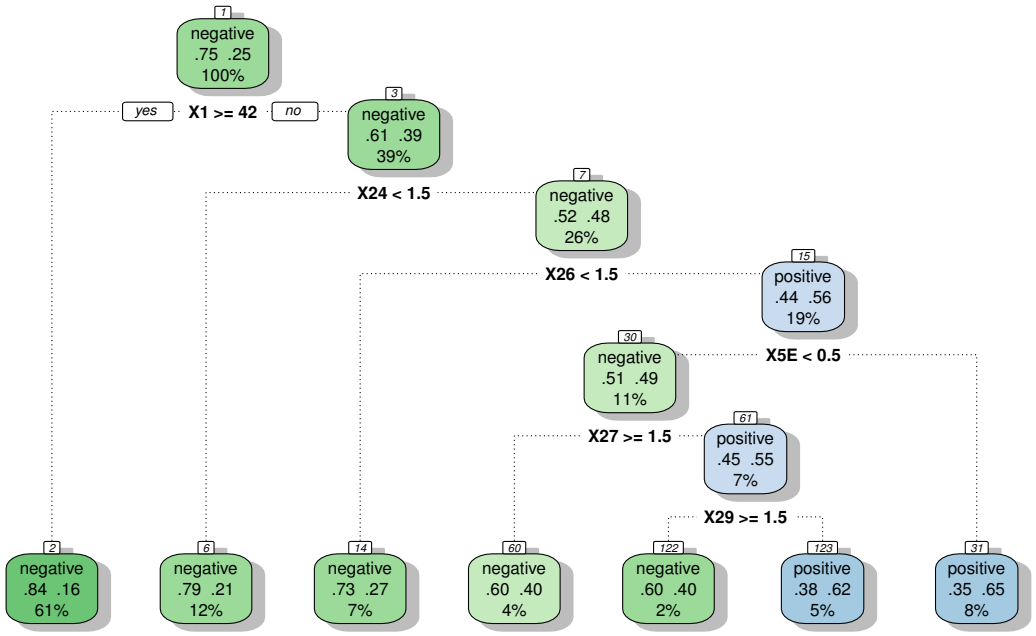
| Selección de datos | Parámetros |
|---|---|
| Número de particiones | 10 |
| Método de generación | Cargadas del disco (particiones previamente generadas en `dataset-partitioning.R`) |
| Partición para training | [1, 10] - {5} (todas menos la 5) |
| Partición para test | {5} |

| Etapa de preprocesamiento | Parámetros |
|---|---|
| Eliminación de anomalías | En una variable: Sí<br>En múltiples variables: No |
| Discretización | No |
| Selección de atributos | Eliminación de atributos poco significativos: Sí<br>Eliminación de redundancias: No<br>Número final de atributos: 30 |
| Eliminación de ruido | No |

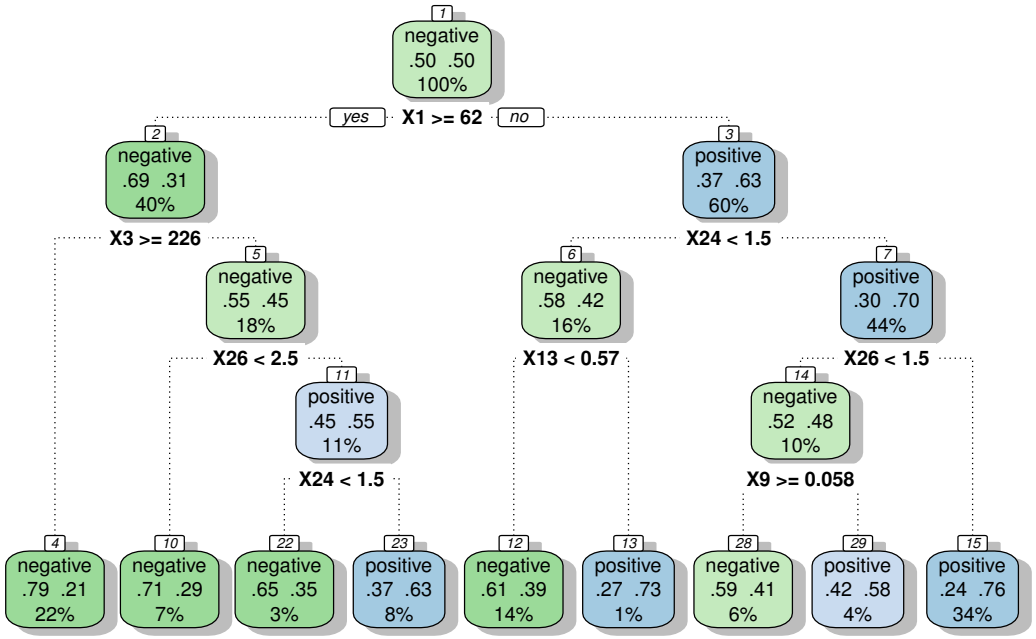
Cuadro 8: Parámetros de selección de datos y preprocesamiento usados para entrenar los árboles de decisión.

| | AUC | Accuracy | Kappa |
|---|---|---|---|
| original | 0.678 | 0.783 | 0.293 |
| downsampling | 0.748 | 0.719 | 0.360 |
| upsampling | 0.743 | 0.717 | 0.345 |
| SMOTE | 0.730 | 0.731 | 0.341 |

Cuadro 9: Resultados obtenidos con los árboles de decisión con los parámetros de la tabla 8. Se presentan los distintos procesamientos llevados a cabo sobre los datos de entrada y las distintas medidas realizadas.




(a) Árbol de decisión generado a partir de los datos sin modificar.




(b) Árbol de decisión generado a partir de los dos datos con *upsampling*.

Figura 12: Árboles de decisión que producen la mejor clasificación posible, generados a partir de los datos preprocesados. Los parámetros usados en el preprocesamiento pueden verse en la tabla 8.

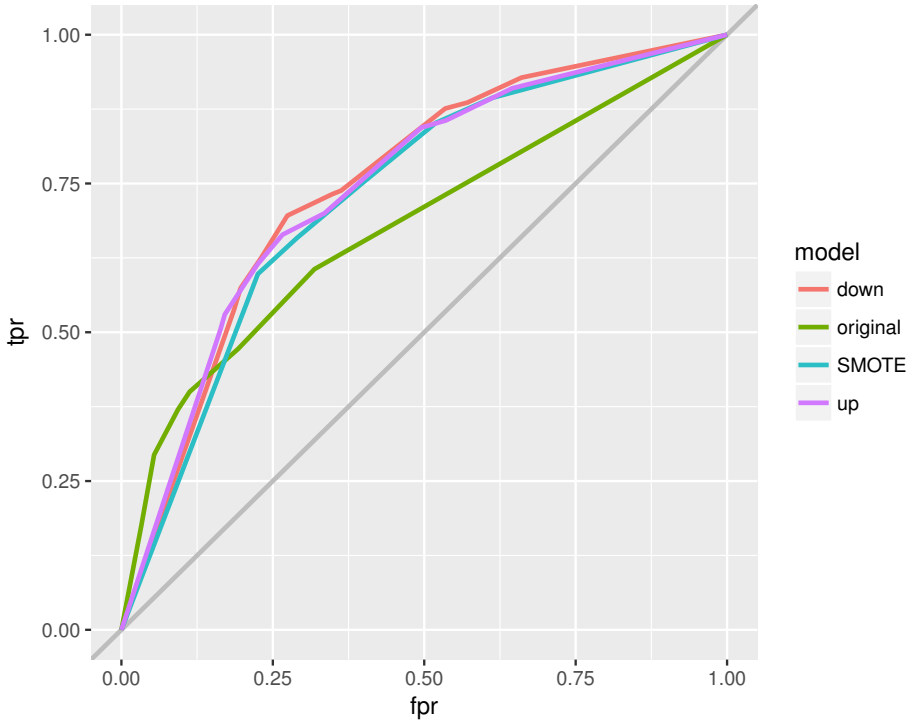



Figura 13: Curvas ROC para el mejor modelo encontrado usando árboles de decisión. Se han usado los parámetros de la tabla 8.

| | AUC | Accuracy | Kappa |
|---|---|---|---|
| original | 0.756 | 0.750 | 0.000 |
| downsampling | 0.787 | 0.704 | 0.354 |
| upsampling | 0.790 | 0.708 | 0.357 |

Cuadro 10: Resultados obtenidos con las máquinas de soporte vectorial con los parámetros de la tabla 8. Se presentan los distintos procesamientos llevados a cabo sobre los datos de entrada y las distintas medidas realizadas.

facilita la separación de las clases al representar los distintos puntos en un espacio vectorial de una dimensionalidad mayor que la del problema original.

Las máquinas de soporte vectorial pueden ser usadas en R gracias a la biblioteca `kernlab`, cuyas funciones están disponibles también a través de `caret`. Usando un kernel lineal y los parámetros de la tabla 8, se han obtenido los resultados mostrados en la tabla 10 y en la figura 14. Como puede comprobarse en estos, los resultados mejoran los de los árboles de decisión. No obstante, se trata de un método computacionalmente muy costoso.

Sería conveniente comprobar si otros tipos de kernel producen mejores resultados. También podrían cambiarse los parámetros del preprocesamiento, habilitando, por ejemplo, la eliminación de ruido.

#### 4.2.3. Random forest

En la tarea de clasificación, hay métodos más efectivos que combinan distintos modelos simples con el objetivo de conseguir una mejor aproximación. Estos métodos, llamados en conjunto métodos de ensamble (*ensamble methods*), son muy eficaces, aunque su entrenamiento tiene un gran coste computacional. De entre estas técnicas, una de las más populares y que mejores resultados obtiene es Random Forest.

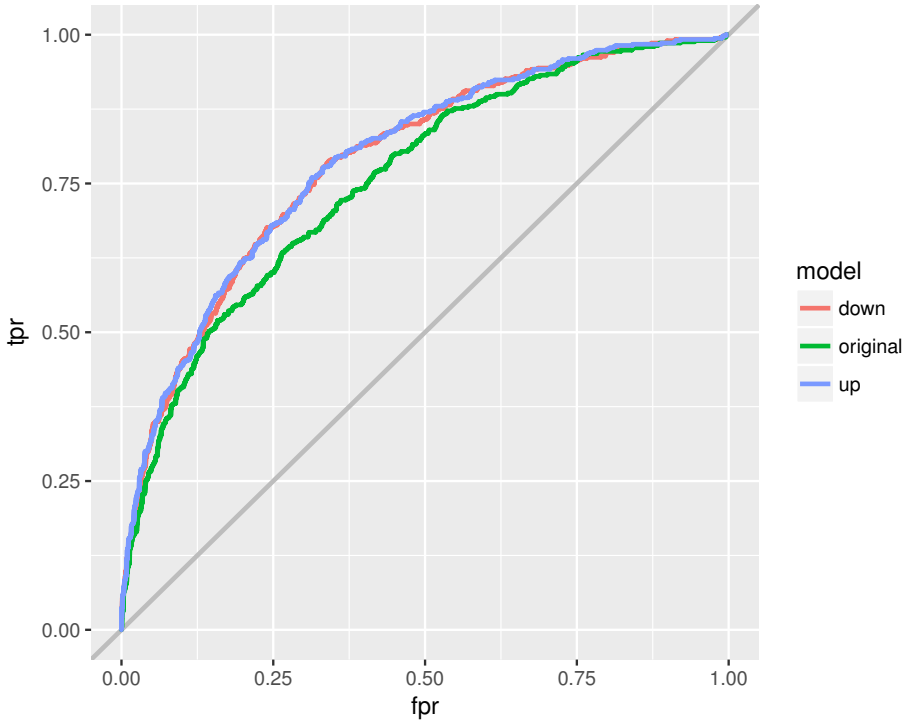



Figura 14: Curvas ROC para los modelos basados en SVM con los parámetros de la tabla 8.

| | AUC | Accuracy | Kappa |
|---|---|---|---|
| original | 0.810 | 0.808 | 0.396 |
| downsampling | 0.809 | 0.737 | 0.406 |
| upsampling | 0.811 | 0.800 | 0.442 |

Cuadro 11: Resultados obtenidos con los modelos basados en random forest con los parámetros de la tabla 8.

**Mejora sobre los árboles de decisión** Para comprobar que random forest supone una mejora sobre los árboles de decisión, se ha llevado a cabo el entrenamiento con los mismos parámetros, mostrados en la tabla 8. Los resultados producidos por este modelo, bajo estas mismas condiciones, se muestran en la tabla 11 y en la figura 15. Como puede observarse, los resultados superan a los de los árboles de decisión sencillos en varios aspectos. Por un lado, el AUC alcanzado es mejor en todos los casos. También lo son, como cabría esperar, el *accuracy* y el *kappa*. Por otro lado, observando las curvas ROC podemos decir que el sobremuestreo o el submuestreo no suponen una mejora significativa para este algoritmo, al menos en el caso tratado. Tal vez un caso en el que la diferencia de proporción fuera mayor requeriría de estos procesos.

**Pruebas con otros parámetros en el preprocesamiento** Como se mencionó anteriormente, la eliminación de ruido, que no mejoraba los resultados producidos por los árboles de decisión, podría mejorar los resultados al entrenar otros modelos. Se ha entrenado un modelo random forest después de haber eliminado ruido por el método IPF, como se indica en la tabla 12, donde pueden consultarse todos los parámetros utilizados. Los resultados son los mostrados en la tabla 13.

Al contrario de lo esperado, la eliminación de ruido no mejora el ajuste de este modelo a los datos de test. Sería necesario revisar los ajustes de esta etapa para tratar de conseguir mejores resultados.

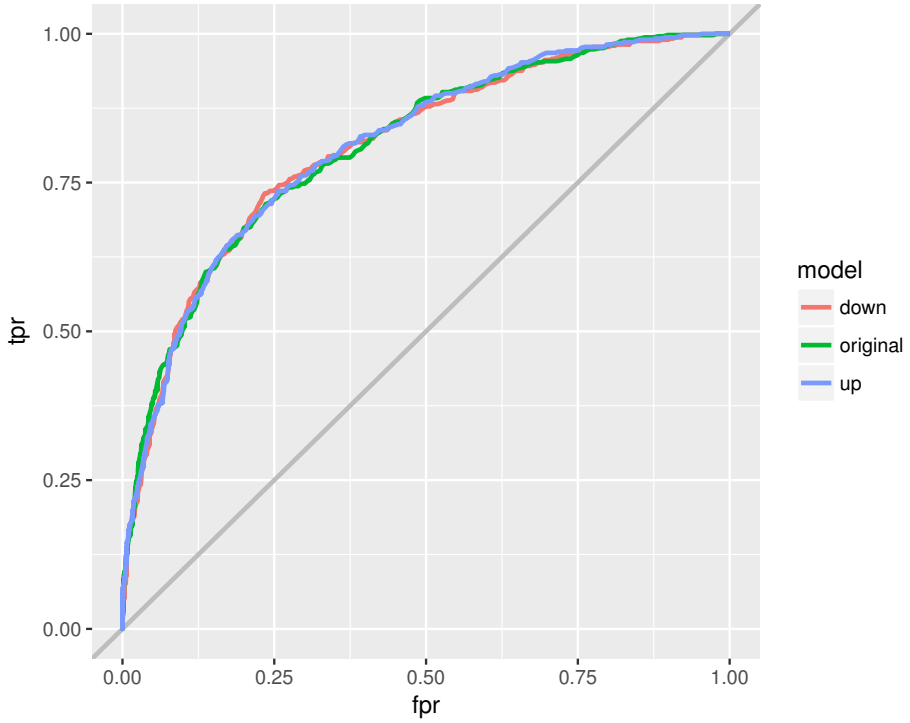



Figura 15: Curvas ROC para los modelos basados en random forest con los parámetros de la tabla 8.

| Selección de datos | Parámetros |
|---|---|
| Número de particiones | 10 |
| Método de generación | Cargadas del disco (particiones previamente generadas en `dataset-partitioning.R`) |
| Partición para training | [1, 10] - {5} (todas menos la 5) |
| Partición para test | {5} |

| Etapa de preprocesamiento | Parámetros |
|---|---|
| Eliminación de anomalías | En una variable: Sí<br>En múltiples variables: No |
| Discretización | No |
| Selección de atributos | Eliminación de atributos poco significativos: Sí<br>Eliminación de redundancias: No<br>Número final de atributos: 30 |
| Eliminación de ruido | IPF |

Cuadro 12: Parámetros de selección de datos y preprocesamiento usados para entrenar el modelo basado en random forest con eliminación de ruido.

| | AUC | Accuracy | Kappa |
|---|---|---|---|
| original | 0.797 | 0.806 | 0.382 |
| downsampling | 0.791 | 0.754 | 0.418 |
| upsampling | 0.802 | 0.800 | 0.433 |

Cuadro 13: Resultados obtenidos con los modelos basados en random forest con los parámetros de la tabla 12.

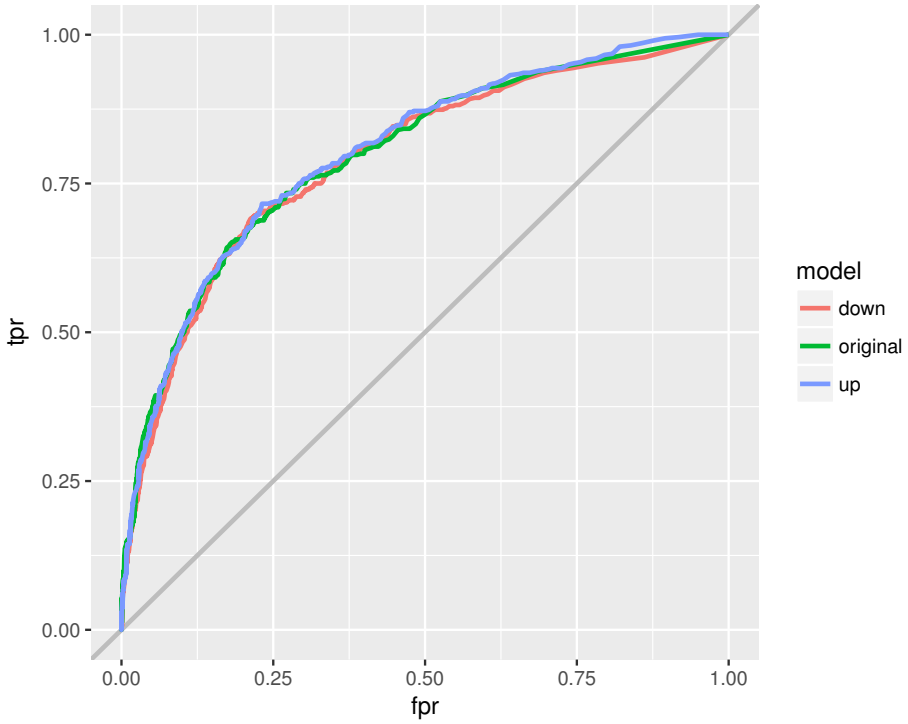



Figura 16: Curvas ROC para los modelos basados en random forest con los parámetros de la tabla 12.

## 4.3. Comparación de resultados

Atendiendo a los resultados obtenidos con los métodos estudiados, debemos considerar que random forest supera tanto a las máquinas de soporte vectorial con kernel lineal como a los árboles de decisión sencillos. Sin embargo, las SVM son muy superiores a los árboles de decisión sencillos, y su rendimiento se acerca al de random forest. Hay que decir que tanto el entrenamiento de random forest como el de SVM son computacionalmente muy costosos, mientras que los árboles de decisión no requieren tanto poder computacional.

También cabe mencionar que no todas las etapas de preprocesamiento han sido útiles, ya que algunas hacían que los resultados empeorasen con respecto a los producidos con los datos sin preprocesar. Como se ha mencionado anteriormente, es probable que estas etapas sean útiles y ayuden a mejorar los datos de entrenamiento si se ajustan correctamente sus parámetros. Para ello es necesaria una ampliación del trabajo de ajuste de parámetros, probando otros métodos en cada etapa del preprocesamiento y teniendo en cuenta otras características del conjunto de datos. Es posible que tener un conocimiento experto en la materia ayudara en esta tarea.

# Referencias


[1] Inc. RStudio. Rstudio - free and open-source ide for r. `https://www.rstudio.com/`, 2017.

[2] Jeffrey S Racine. Rstudio: A platform-independent ide for r and sweave. *Journal of Applied Econometrics*, 27(1):167–172, 2012.

[3] Ross Ihaka and Robert Gentleman. R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3):299–314, 1996.

[4] Hadley Wickham. ggplot2 - a plotting system for r. `http://ggplot2.org/`, 2013.

[5] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis (Use R!)*. Springer, 2009.

[6] Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3):408–421, 1972.

[7] Wikimedia Commons. File:roc space.png — wikimedia commons, the free media repository, 2014. [Online; accessed 30-August-2017 ].

[8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[9] José A. Sáez, Julián Luengo, Jerzy Stefanowski, and Francisco Herrera. Smote–ipf: Addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering. *Information Sciences*, 291:184 – 203, 2015.

[10] Elizabeth R DeLong, David M DeLong, and Daniel L Clarke-Pearson. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, pages 837–845, 1988.

[11] Pablo Morales, Julián Luengo, Luís P.F. Garcia, Ana C. Lorena, André C.P.L.F. de Carvalho, and Francisco Herrera. The NoiseFiltersR Package: Label Noise Preprocessing in R. *The R Journal*, 9(1):219–228, 2017.

[12] Donald Michie, David J Spiegelhalter, and Charles C Taylor. Machine learning, neural and statistical classification. 1994.

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

[14] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[15] Johannes Fürnkranz, Dragan Gamberger, and Nada Lavrač. *Foundations of rule learning*. Springer Science & Business Media, 2012.

[16] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

# A. Estructura y uso del código generado

El código desarrollado para este trabajo se ha estructurado en los siguientes ficheros:

1. Particionado y visualización de los datos

   **dataset-partitioning.R** se encarga del particionado del conjunto de datos proporcionado en diez particiones o *folds* que serán usadas en el proceso de validación cruzada. Estas particiones son guardadas en disco con el fin de permitir la reproducibilidad. Adicionalmente, proporciona funciones para acceder a los datos de entrenamiento y test de una forma cómoda.

   **load-data.R** se encarga de cargar los datos previamente guardados en la memoria no volátil. Además, proporciona algunas funciones útiles para acceder a partes de estos datos de forma cómoda.

   **initial-visualization.R** se encarga de la visualización inicial de los datos, mostrando el tamaño del conjunto de datos, la clase de cada atributo y la distribución en frecuencias de los mismos.

2. Preprocesamiento

   **outlier-removal-UNI.R** se encarga de detectar anomalías en una sola variable.

   **outlier-removal-MCD.R** se encarga de detectar anomalías en múltiples variables.

   **outlier-removal.R** se encarga de poner en común las salidas de los scripts anteriores, eliminando las anomalías del conjunto de datos.

   **discretization.R** se encarga de la discretización de los datos.

   **attribute-selection.R** se encarga de la selección de características, comenzando por la importancia de cada atributo para terminar con las redundancias.

   **noise-filter.R** se encarga del filtrado de los datos, eliminando instancias cuya clase está probablemente equivocada.

3. Clasificación

   **classsification-orig.R** lleva a cabo la clasificación con los datos originales (sin tratamiento para el problema de las clases no balanceadas). Este fichero debe ejecutarse siempre antes que cualquier otra clasificación.

   **classification-down.R** lleva a cabo la clasificación con los datos con submuestreo de la clase mayoritaria.

   **classification-up.R** lleva a cabo la clasificación con los datos con sobremuestreo de la clase minoritaria.

   **classification-smote.R** lleva a cabo la clasificación con los datos con submuestreo SMOTE.

   **classification-visualization.R** se encarga de la visualización de los resultados obtenidos en las clasificaciones anteriores.

4. Ejecución

   **mdpc-script.R** es un script que ejecuta de forma secuencial cada uno de los pasos de las etapas de preprocesamiento y de clasificación.

   **mdpc-options.R** es un script que genera una lista de opciones, a partir de la cuál se decide qué tipo de preprocesamiento y clasificación se llevan a cabo.

Cabe mencionar que cada uno de los scripts hace uso de variables globales que han sido generadas por los scripts anteriores. Por este motivo, los scripts deben ejecutarse de forma secuencial, en el orden descrito anteriormente. Los resultados obtenidos pueden verse afectados si estas variables globales son modificadas manualmente antes de ejecutar cualquiera de los scripts. Es por esto que se proporciona el fichero `mdpc-script.R`, con el que nos aseguramos de que los resultados son perfectamente reproducibles.

Las opciones para la importación de los datos, así como las opciones de preprocesamiento y de clasificación se definen en el fichero `mdpc-options.R`. Para reproducir los resultados expuestos en esta memoria, basta con copiar las tablas de parámetros con el formato adecuado en este fichero.

# B. Código fuente: Particionado y visualización del conjunto de datos

## B.1. Código fuente usado en el particionado de los datos

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: dataset-partitioning.R                                      #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(utils)
library(caret)

# Load the dataset
dataset = read.csv("data/datcom2016.csv")

# Set the random seed in order to be able to reproduce the results
set.seed(1)

# Create the folds for k-fold cross validation
folds = createFolds(
  y = dataset$class,
  k = 10
)

# Get the folds from dataset
split_up = lapply(
  folds,
  function(ind, dat) {
    dat[ind,]
  },
  dat = dataset
)

# Quick check for the structure
unlist(lapply(split_up, nrow))

# Function to quickly access to training data for a given fold
dataset.train = function(i) {
  return(do.call("rbind", splidatat_up[-i]))
}

# Function to quickly access to test data for a given fold
dataset.test = function(i) {
  return(split_up[[i]])
}

# Save the data (so it can be loaded afterwards)
for (k in 1:10) {
  write.csv(
    split_up[[k]],
    paste("data/datcom2016-10-", k, "-fold.csv", sep = "")
  )
}
```

End Of File 'dataset-partitioning.R'

## B.2. Código fuente usado en la adquisición de los datos

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: load-data.R                                                 #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(utils)
library(stats)


# Load the complete dataset
dataset = read.csv("data/datcom2016.csv")
names(dataset)[1:50] = paste("X", 1:50, sep = "")

# Load the 10-folds
split_up    = list()  # The k-folds (same to dataset.test)
for (k in 1:10) {
  filename = paste(
    "data/datcom2016-10-",
    as.character(k),
    "-fold",
    ".csv",
    sep = ""
  )
  x = read.csv(filename)
  rownames(x) = x$X
  x = x[,-1]
  names(x)[1:50] = paste("X", 1:50, sep = "")

  split_up[[k]] = x
}

# Function to quickly access to training set for a given fold
dataset.train = function(i) {
  return(do.call("rbind", split_up[-i]))
}

# Function to quickly access to test set for a given fold
dataset.test = function(i) {
  return(split_up[[i]])
}

# Create some useful sets
dataset.tra    = dataset.train(1)     # Training set
dataset.tra.cl = dataset.tra$class    #  |-> Class
dataset.tra.dt = dataset.tra[, -50]   #  \-> Data
dataset.tst    = dataset.test(1)      # Training set
dataset.tst.cl = dataset.tst$class    #  |-> Class
dataset.tst.dt = dataset.tst[, -50]   #  \-> Data

# Separate data into numeric and factor (useful for visualization)
subset.numeric.dt = function(df) {
  return(
```

```
    df[,sapply(df, class) != "factor", drop = F]
  )
}
subset.numeric.cl = function(df) {
  return(
    df$class
  )
}
subset.numeric    = function(df) {
  return(cbind(
    subset.numeric.dt(df),
    class = df$class
  ))
}
subset.factor.dt  = function(df) {
  return(subset(
    df[,sapply(df, class) == "factor", drop = F],
    select = -class
  ))
}
subset.factor.cl  = subset.numeric.cl
subset.factor     = function(df) {
  return(
    df[,sapply(df, class) == "factor", drop = F]
  )
}
subset.num.dt = function(df) {
  return(
    df[,sapply(df, class) == "numeric", drop = F]
  )
}
subset.num.cl = subset.numeric.cl
subset.num    = function(df) {
  return(cbind(
    subset.num.dt(df),
    class = df$class
  ))
}
subset.int.dt = function(df) {
  return(
    df[,sapply(df, class) == "integer", drop = F]
  )
}
subset.int.cl = subset.numeric.cl
subset.int    = function(df) {
  return(cbind(
    subset.int.dt(df),
    class = df$class
  ))
}

# Remove temporal variables
rm(filename, x)
```

End Of File 'load-data.R'

## B.3. Código fuente usado en la visualización preliminar de los datos

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: initial-visualization.R                                     #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################


# Load required libraries
library(utils)
library(ggplot2)
library(reshape)

# Load the dataset
dataset = read.csv("data/datcom2016.csv")

# Set the random seed in order to be able to reproduce the results
set.seed(1)


# Visualize the structure of the data
dim(dataset)

data.types = sapply(dataset[, -51], class)
table(data.types)
## Data contains 20000 registers from 50 attributes plus a class variable.
## 9 of the attributes are factors, whist the remaining 41 are numeric (
## 22 of which are integer, the other 19 being continous variables)


# Build a summary table for the document
mt = matrix(nrow = 50, ncol = 4)
mt[,1] = names(data.types)
mt[,2] = data.types
mt[,4] = paste("X", 1:50, sep ="")
for (i in 1:50) {
  if (data.types[i] == "factor") {
    mt[i,3] = paste("{", paste(levels(dataset[,i]), collapse = ", "), "}", sep = "")
  } else if (data.types[i] == "numeric") {
    mt[i,3] = sprintf("[%.2f, %.2f]", min(dataset[,i]), max(dataset[,i]))
  } else {
    mt[i,3] = sprintf("[%d, %d]", min(dataset[,i]), max(dataset[,i]))
  }
}


# Get additional summary of the data
str(dataset)
summary(dataset)

```

```r
## There are too many variables to understand at once. Let's do it
## little by little.

# Frequency of each class
table(dataset$class)
prop.table(table(dataset$class))

# Plot frequency of each class
ggplot(dataset) +
  geom_bar(aes(class, fill = class))

## We are facing an imbalanced problem


# Check for missing values
data.frame(missing.values = colSums(is.na(dataset)))

## There are no missing values


# Plot numeric data
## Distributions
dataset.numeric.dt = dataset[,sapply(dataset, class) != "factor"]
dataset.numeric.cl = dataset$class
dataset.numeric = cbind(
  dataset.numeric.dt,
  class = dataset.numeric.cl
)
dataset.factor  = dataset[,sapply(dataset, class) == "factor"]
myData = melt.data.frame(dataset.numeric)

## Density (relative to the class)
ggplot(myData) +
  geom_density(aes(value)) +
  geom_density(aes(value, color = class)) +
  facet_wrap(~variable, scales = "free")
## Frequency (more samples, more frequency)
ggplot(myData) +
  geom_freqpoly(aes(value)) +
  geom_freqpoly(aes(value, color = class)) +
  facet_wrap(~variable, scales = "free")


# Plots for the document
myData1 = melt.data.frame(
  cbind(dataset.numeric[,1:40], class = dataset.numeric.cl)
)
myData2 = melt.data.frame(
  dataset.factor,
  id.vars = "class"
)
ggplot(myData1) +
  geom_freqpoly(aes(value)) +
  geom_freqpoly(aes(value, color = class)) +
  facet_wrap(~variable, scales = "free", ncol = 4)
ggplot(myData2) +
  geom_bar(aes(value, fill = class), position = "dodge") +
  facet_wrap(~variable, scales = "free", ncol = 4)


# ATTRIBUTE-SELECTION APPROACH
```

```r

# Can attributes with the same distribution be informative?
ks.test(dataset$PredCN_r1, dataset$PredCN_r2)
## These two attributes may have the same distribution.
ks.test(
  subset(dataset, subset = class == "positive")$PredCN_r1,
  subset(dataset, subset = class == "positive")$PredCN_r2
)
ks.test(
  subset(dataset, subset = class == "negative")$PredCN_r1,
  subset(dataset, subset = class == "negative")$PredCN_r2
)
## Indeed, their relationship to positive and negative classes
## seem to be the same.
myData = subset(dataset, select = c(PredCN_r1, PredCN_r2, class))
myData = melt(myData)
ggplot(myData) +
  geom_density(aes(value)) +
  geom_density(aes(value, color = class)) +
  facet_wrap(~variable, ncol = 1)
## It could be thought that one of them could be redundant, thus
## being appropriate to omit it.
ggplot(dataset) + geom_bar(aes(class, fill = as.factor(PredCN_r1)))
ggplot(dataset) + geom_bar(aes(class, fill = as.factor(PredCN_r2)))
## They do seem to hold the same information.
cor(dataset$PredCN_r1, dataset$PredCN_r2)
## Nevertheless, the two variables are not correlated...
ggplot(dataset) +
  geom_jitter(aes(PredCN_r1, PredCN_r1, color = class), alpha = 0.1)
ggplot(dataset) +
  geom_jitter(aes(PredCN_r1, PredCN_r2, color = class), alpha = 0.1)
## And that is the most important part, because they are independent
   ↪ ,
## but they inform very well about the class.
## Having the two variables makes a great decission boundary,
## so none of them should be deleted.
## Conclusion: same distribution does not mean the variables should
   ↪ be removed...
```

End Of File 'initial-visualization.R'

# C. Código fuente: Preprocesamiento

## C.1. Código fuente usado en la detección de valores anómalos

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: outlier-removal-UNI.R                                       #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(outliers)
library(reshape)
library(ggplot2)

# Visualization
if (!exists("plot.enable")) {
  plot.enable = F
}

# Remove variables
if (!exists("rm.variables")) {
  rm.variables = F
}

# Set the random seed in order to be able to reproduce the results
set.seed(1)

# A row will be considered an outlier if it presents an
# anomaly on any variable or if it presents an anomalous
# combination of its values.

# UNIVARIATE OUTLIERS

# Remove univariate outliers (only from numeric data, not integers!)

## Look for outliers in every column
my.dataset.dt = subset.numeric.dt(dataset.tra)[, sapply(subset.
    ↪ numeric.dt(dataset.tra), class) == "numeric"]
my.dataset.cl = subset.numeric.cl(dataset.tra)
my.dataset    = cbind(my.dataset.dt, class = my.dataset.cl)
removed.rows  = 0
UNI.outliers  = c()
for (i in 1:ncol(my.dataset.dt)) {
    new.data     = rm.outlier(my.dataset.dt[,i], fill = F)
    UNI.outliers = c(UNI.outliers, which(!(my.dataset.dt[,i] %in%
        ↪ new.data)))

    if (plot.enable) {
      # Visualize the removed outliers
      print(
        ggplot(my.dataset, aes_string(
          (names(my.dataset)[i]),
          color = "class")) +
          geom_freqpoly(bins = 70) +
```

```
52              geom_freqpoly(data = subset.numeric(dataset.tra)[UNI.
                    ↪ outliers, ], color = "#FFCC00", bins = 70)
53        )
54      }
55  }
56
57  ## Get the rows containing outliers
58  UNI.outliers  = unique(UNI.outliers)
59
60  removed.rows  = length(UNI.outliers)
61  print(paste("Removed", removed.rows, "rows␣containing␣univariate␣
        ↪ outliers"))
```

End Of File 'outlier-removal-UNI.R'

```r
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: outlier-removal-MCD.R                                       #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(FSelector)
library(mvoutlier)
library(reshape)
library(ggplot2)

# Visualization
if (!exists("plot.enable")) {
  plot.enable = F
}

# Remove variables
if (!exists("rm.variables")) {
  rm.variables = F
}

# Set the random seed in order to be able to reproduce the results
set.seed(1)

# A row will be considered an outlier if it presents an
# anomaly on any variable or if it presents an anomalous
# combination of its values.



# MULTIVARIATE OUTLIERS

# Remove multivariate outliers (from both numeric and integer)

## Select the 10 most useful variables according to chi.squared test
weights.chi.squared = chi.squared(class~., subset.num(dataset.tra))
weights.chi.squared =
  weights.chi.squared[order(weights.chi.squared, decreasing = T), ,
    ↪ drop = F]

my.selection = names(subset.numeric.dt(dataset.tra)) %in% rownames(
  ↪ weights.chi.squared)[1:10]
my.dataset.10.dt = subset.numeric.dt(dataset.tra)[,my.selection]
my.dataset.10.cl = subset.numeric.cl(dataset.tra)
my.dataset.10    = cbind(my.dataset.10.dt, class = my.dataset.10.cl)

## Normalize the data
my.dataset.10.dt.scaled = scale(my.dataset.10.dt)

## Get the multivariate outliers
alpha.value = 0.05
alpha.value = 1 - ( 1 - alpha.value) ^ (1 / nrow(my.dataset.10.dt.
  ↪ scaled))
mvoutlier.plot = uni.plot(my.dataset.10.dt.scaled, symb = FALSE,
  ↪ alpha = alpha.value)
is.MCD.outlier = mvoutlier.plot$outliers
```

```
56
57 ## Get the rows containing outliers
58 MCD.outliers = which(is.MCD.outlier)
59
60 removed.rows = length(MCD.outliers)
61 print(paste("Removed", removed.rows, "rows␣containing␣multivariate␣
       ↪ outliers"))
62
63 ## Visualize removed outliers
64 if (plot.enable) {
65   ggplot(my.dataset.10, aes(X1, X3, color = class)) +
66     geom_point(alpha = 0.1) +
67     geom_point(data = my.dataset.10[MCD.outliers, ], alpha = 0.1,
          ↪ color = "#FFCC00")
68   ggplot(my.dataset.10, aes(X3, X40, color = class)) +
69     geom_jitter(alpha = 0.1) +
70     geom_jitter(data = my.dataset.10[MCD.outliers, ], alpha = 0.1,
          ↪ color = "#FFCC00")
71 }
```

End Of File 'outlier-removal-MCD.R'

```r
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: outlier-removal.R                                           #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(reshape)
library(ggplot2)

# Visualization
if (!exists("plot.enable")) {
  plot.enable = F
}

# Remove variables
if (!exists("rm.variables")) {
  rm.variables = F
}


# Remove outliers from the data
if (!exists("UNI.outliers")) {
  UNI.outliers = c()
}
if (!exists("MCD.outliers")) {
  MCD.outliers = c()
}
the.outliers = unique(c(UNI.outliers, MCD.outliers))
dataset.tra.preprocessed    = dataset.tra[-the.outliers, ]
dataset.tra.preprocessed.dt = dataset.tra.dt[-the.outliers, ]
dataset.tra.preprocessed.cl = dataset.tra.cl[-the.outliers]

## Check that the removal of outliers is independent to the class
   ↪ the data come from
new.proportion = table(dataset.tra.preprocessed$class)
old.proportion = table(dataset$class)

print("Class␣frequency␣prior␣outlier␣removal:")
print(sprintf("negative:␣%.2f␣%%", 100*prop.table(old.proportion)
   ↪ [1]))
print(sprintf("positive:␣%.2f␣%%", 100*prop.table(old.proportion)
   ↪ [2]))

print("Class␣frequency␣after␣outlier␣removal:")
print(sprintf("negative:␣%.2f␣%%", 100*prop.table(new.proportion)
   ↪ [1]))
print(sprintf("positive:␣%.2f␣%%", 100*prop.table(new.proportion)
   ↪ [2]))

## Proportions are very similar, i.e. the removal of registers is
   ↪ not conditioned by the class
## (it could be thought that the minoritary class would be selected
   ↪ for removals)


# Analyze the percentage of removed registers
```

```r
print(sprintf(
  "Proportion of removed outliers: %.2f %%",
  100*(1 - nrow(dataset.tra.preprocessed) / nrow(my.dataset))
))
## Around 18% of the registers have been removed

# Remove temporal variables
if (rm.variables) {
  rm(
    new.proportion,
    old.proportion,
    MCD.outliers,
    UNI.outliers,
    removed.rows,
    the.outliers,
    is.MCD.outlier,
    alpha.value,
    my.selection,
    my.dataset.10,
    my.dataset.10.cl,
    my.dataset.10.dt,
    mvoutlier.plot,
    weights.chi.squared
  )
}
```

End Of File 'outlier-removal.R'

## C.2. Código fuente usado en la discretización de atributos

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: discretization.R                                            #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(discretization)
library(reshape)
library(ggplot2)

# Visualization
if (!exists("plot.enable")) {
  plot.enable = F
}

# Remove variables
if (!exists("rm.variables")) {
  rm.variables = F
}

# Set the random seed in order to be able to reproduce the results
set.seed(1)

# Discetize the data
dataset.discretized = disc.Topdown(
  subset.numeric(dataset.tra.preprocessed)
)

# Give a proper name to output from this step
dataset.tra.preprocessed.discretized = dataset.discretized$Disc.data

# Plot the results
if (plot.enable) {
  ## Select 10 most meaningful attributes for plot (according to chi
     ↪ .squared test)
  weights.chi.squared = chi.squared(class~., subset.numeric(dataset.
     ↪ tra))
  weights.chi.squared =
    weights.chi.squared[order(weights.chi.squared, decreasing = T),
       ↪ , drop = F]

  my.selection = names(subset.numeric.dt(dataset.tra.preprocessed))
     ↪ %in% rownames(weights.chi.squared)[1:10]
  my.dataset.10.dt = subset.numeric.dt(dataset.tra.preprocessed)[,my
     ↪ .selection]
  my.dataset.10.cl = subset.numeric.cl(dataset.tra.preprocessed)
  my.dataset.10    = cbind(my.dataset.10.dt, class = my.dataset.10.
     ↪ cl)
  my.dataset.10.discretized.dt = subset.numeric.dt(dataset.
     ↪ discretized$Disc.data)[,my.selection]
  my.dataset.10.discretized.cl = subset.numeric.cl(dataset.
     ↪ discretized$Disc.data)
```

```r
my.dataset.10.discretized     = cbind(my.dataset.10.discretized.dt, class = my.dataset.10.cl)
cuts.10 = dataset.discretized$cutp[my.selection]


## Plot the discretized data (only 10 most meaningful variables)
myData = melt.data.frame(
  my.dataset.10.discretized,
  id.vars = "class"
)

ggplot(myData, aes(value, fill = class)) +
  geom_bar(position = "dodge") +
  facet_wrap(~variable, scales = "free", ncol = 5)


## Plot the cut points, along with preprocessed data distributions
## (only 10 most meaningful variables)
myData = melt.data.frame(
  my.dataset.10,
  id.vars = "class"
)
myCuts = data.frame(
  variable = rep(
    names(my.dataset.10)[1:10],
    each = 3
  ),
  value = unlist(
    cuts.10
  )
)
ggplot(myData) +
  geom_freqpoly(aes(value), bins = 50) +
  geom_freqpoly(aes(value, color = class), bins = 50) +
  geom_vline(
    data = myCuts,
    aes(xintercept = value),
    color = "#BB1199",
    alpha = 0.5
  ) +
  facet_wrap(~variable, scales = "free", ncol = 2)


# Discretization evaluation based on integer variables
## Scatterplots before discretization
int.data          =
  subset.int.dt(dataset.tra.preprocessed)
int.data.discret =
  subset.int.dt(dataset.tra.preprocessed.discretized)

weights.correlation.rank = rank.correlation(int.data)
weights.correlation.rank =
  weights.correlation.rank[order(weights.correlation.rank, decreasing = T), , drop = F]

most.important.4 = rownames(weights.correlation.rank)[c(2:4, 8)]

important.data.n = which(names(int.data) %in% most.important.4)
int.data.important = cbind(
  int.data[,important.data.n],
  class = subset.int.cl(dataset.tra.preprocessed)
```

```
)

myData1 =
  melt.data.frame(int.data.important, id.vars=c("X18", "class"))
myData2 =
  melt.data.frame(int.data.important, id.vars=c("X19", "class"))
myData3 =
  melt.data.frame(int.data.important, id.vars=c("X20", "class"))
myData4 =
  melt.data.frame(int.data.important, id.vars=c("X25", "class"))
myData1$variableX = "X18"
myData2$variableX = "X19"
myData3$variableX = "X20"
myData4$variableX = "X25"
names(myData1)[1] = "x"
names(myData2)[1] = "x"
names(myData3)[1] = "x"
names(myData4)[1] = "x"
myData = rbind(myData4, myData1, myData2, myData3)

ggplot(myData, aes(x, value)) +
  geom_jitter(aes(colour = class), alpha = 0.1) +
  facet_grid(variableX~variable) +
  xlab("") + ylab("")


## Scatterplots after discretization
important.data.n = which(
  names(int.data.discret)
  %in%
  most.important.4
)
int.data.important = cbind(
  int.data.discret[,important.data.n],
  class = subset.int.cl(dataset.tra.preprocessed)
)

myData1 =
  melt.data.frame(int.data.important, id.vars=c("X18", "class"))
myData2 =
  melt.data.frame(int.data.important, id.vars=c("X19", "class"))
myData3 =
  melt.data.frame(int.data.important, id.vars=c("X20", "class"))
myData4 =
  melt.data.frame(int.data.important, id.vars=c("X25", "class"))
myData1$variableX = "X18"
myData2$variableX = "X19"
myData3$variableX = "X20"
myData4$variableX = "X25"
names(myData1)[1] = "x"
names(myData2)[1] = "x"
names(myData3)[1] = "x"
names(myData4)[1] = "x"
myData = rbind(myData4, myData1, myData2, myData3)

ggplot(myData, aes(x, value)) +
  geom_jitter(aes(colour = class), alpha = 0.1) +
  facet_grid(variableX~variable) +
  xlab("") + ylab("")


```

```
  # Remove temporal variables
  if (rm.variables) {
    rm(
      myData,
      myData1,
      myData2,
      myData3,
      myData4,
      int.data.important,
      int.data.discret,
      int.data,
      important.data.n,
      most.important.4,
      myCuts,
      cuts.10,
      my.selection,
      my.dataset.10.discretized,
      my.dataset.10.discretized.cl,
      my.dataset.10.discretized.dt,
      my.dataset.10,
      my.dataset.10.cl,
      my.dataset.10.dt,
      weights.chi.squared
    )
  }
}
## As it can be seen in the plots, the cuts are made where the
## possitive/negative proportion changes the most


# Remove temporal variables
if (rm.variables) {
  rm(
    dataset.discretized
  )
}
```

End Of File 'discretization.R'

## C.3. Código fuente usado la selección de características

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: attribute-selection.R                                       #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(utils)
library(ggplot2)
library(reshape)
library(FSelector)
library(caret)

# Visualization
if (!exists("plot.enable")) {
  plot.enable = F
}

# Remove variables
if (!exists("rm.variables")) {
  rm.variables = F
}

# Set the random seed in order to be able to reproduce the results
set.seed(1)


# Get a score for the importance of attributes according to the
    ↪ information they provide

numeric.data.dt = subset.numeric.dt(dataset.tra.preprocessed)
numeric.data = subset.numeric(dataset.tra.preprocessed)
weights = data.frame(attribute = names(dataset.tra.preprocessed)
    ↪ [1:50])
weights.num = data.frame(attribute = names(numeric.data.dt))

## According to their independence to the class variable
weights$chi.squared =
  chi.squared(class~., dataset.tra.preprocessed)

weights.chi.squared =
  weights$chi.squared[
    order(weights$chi.squared, decreasing = T), , drop = F
  ]

## These results can be verified with a KS test, by checking
## for similar distributions between the positive and
## negative class
ks.test.results.pn = matrix(
  nrow = ncol(numeric.data.dt),
  ncol = 1
)
rownames(ks.test.results.pn) = names(numeric.data.dt)
myData.p = subset(
```

```
  numeric.data,
  subset = class == "positive"
)
myData.n = subset(
  numeric.data,
  subset = class == "negative"
)
for (i in 1:(ncol(numeric.data.dt))) {
  ks.test.results.pn[i,1] = ks.test(
    myData.p[,i],
    myData.n[,i]
  )$p.value
}

## Really non-useful classes
rownames(ks.test.results.pn)[ks.test.results.pn>0.1]

## Verification of the results of chi.squared test
rownames(ks.test.results.pn)[ks.test.results.pn>0.00001]
rownames(weights.chi.squared)[weights.chi.squared == 0]

## In conclusion: having the same distribution for positive and
## negative is not very informative... We will drop every
## variable getting 0 score in chi.squared test.


## Entropy measures
weights$information.gain = information.gain(class~., dataset.tra.
    ↪ preprocessed, "log2")
weights.num$information.gain = information.gain(class~., numeric.
    ↪ data, "log2")
weights.information.gain = weights$information.gain[order(weights$
    ↪ information.gain, decreasing = T), , drop = F]

weights$gain.ratio = gain.ratio(class~., dataset.tra.preprocessed, "
    ↪ log2")
weights.num$gain.ratio = gain.ratio(class~., numeric.data, "log2")
weights.gain.ratio = weights$gain.ratio[order(weights$gain.ratio,
    ↪ decreasing = T), , drop = F]
### Results differ greatly!!

## OneR measure
weights$oneR = 1-oneR(class~., dataset.tra.preprocessed)
weights.oneR = weights$oneR[order(weights$oneR, decreasing = T), ,
    ↪ drop = F]

# Get a summary of the different measurements
my.scale = function(x) { return ((x-min(x))/(max(x)-min(x))) }

weights.scaled = weights
weights.scaled[,2:ncol(weights)] = data.frame(lapply(weights[,2:ncol
    ↪ (weights)], my.scale ))
weights.scaled$sum = rowSums(weights.scaled[,2:ncol(weights)])
weights.scaled$sum.scaled = 1.5*my.scale(weights.scaled$sum)
weights.scaled$type = sapply(dataset, class)[1:50]
weights.scaled = weights.scaled[order(weights.scaled$sum, decreasing
    ↪  = T), , drop = F]
weights.scaled = weights.scaled[1:40, ]
rownames(weights.scaled) = 1:40

# Select attributes according to this summary
```

```r
my.selection = which(names(dataset.tra) %in% weights.scaled[1:30,1])


# Plot results
if(plot.enable) {
  myData = melt.data.frame(
    weights.scaled,
    measure.vars = c("sum.scaled", "chi.squared", "oneR", "gain.ratio", "information.gain")
  )
  myData$attribute = factor(myData$attribute, levels = weights.scaled$attribute)

  ggplot(myData) +
    geom_rect(
      aes(color = type),
      xmin = -Inf,
      xmax = Inf,
      ymin = -Inf,
      ymax = Inf,
      alpha = 0.02
    ) +
    geom_col(aes(variable, value, fill = variable)) +
    facet_wrap(~attribute, ncol = 5) + coord_flip() +
    xlab("") + ylab("")

  if(rm.variables) {
    rm(
      myData
    )
  }
}

# Check for redundancies
## Check correlation amongst variables
correlations = cor(numeric.data.dt)
drop.attributes.n = c(
  findCorrelation(correlations, cutoff = 0.8)
)

## Remove redundant attributs from my.selection of attributes
my.new.selection = setdiff(my.selection, drop.attributes.n)

# Simplify the original datasets
select.attributes = function(df, cor.cutoff = 0.8) {
  drop.attributes.n = c(
    findCorrelation(correlations, cutoff = cor.cutoff)
  )

  my.new.selection = setdiff(my.selection, drop.attributes.n)
  return(df[,my.new.selection])
}

# Get a dataframe with the selected attributes
dataset.tra.preprocessed.selected.dt = select.attributes(dataset.tra.preprocessed)
dataset.tra.preprocessed.selected.cl = dataset.tra.preprocessed.cl
dataset.tra.preprocessed.selected = cbind(
  dataset.tra.preprocessed.selected.dt,
  class = dataset.tra.preprocessed.selected.cl
)
```

```
166
167
168 # Build a summary table for the document
169 mt = matrix(nrow = ncol(dataset.tra.preprocessed.selected.dt), ncol
    ↪ = 3)
170 mt[,1] = names(dataset.tra.preprocessed.selected.dt)
171 mt[,2] = sapply(dataset.tra.preprocessed.selected.dt, class)
172 for (i in 1:25) {
173   if (mt[i,2] == "factor") {
174     mt[i,3] = paste("{", paste(levels(dataset.tra.preprocessed.
        ↪ selected.dt[,i]), collapse = ",␣"), "}", sep = "")
175   } else if (mt[i,2] == "numeric") {
176     mt[i,3] = sprintf("[%.2f,␣%.2f]", min(dataset.tra.preprocessed.
        ↪ selected.dt[,i]), max(dataset.tra.preprocessed.selected.dt
        ↪ [,i]))
177   } else {
178     mt[i,3] = sprintf("[%d,␣%d]", min(dataset.tra.preprocessed.
        ↪ selected.dt[,i]), max(dataset.tra.preprocessed.selected.dt
        ↪ [,i]))
179   }
180 }
181
182 # Remove temporal variables
183 if(rm.variables) {
184   rm(
185     mt,
186     numeric.data,
187     numeric.data.dt,
188     numeric.data.simplified,
189     weights,
190     weights.num,
191     weights.scaled,
192     weights.oneR,
193     weights.gain.ratio,
194     weights.information.gain,
195     weights.chi.squared,
196     myData.n,
197     myData.p,
198     ks.test.results.pn
199   )
200 }
```

End Of File 'attribute-selection.R'

## C.4. Código fuente usado en la detección de ruido

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: noise-filter.R                                              #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(NoiseFiltersR)
library(unbalanced)
library(ggplot2)
library(reshape)

# Visualization
if (!exists("plot.enable")) {
  plot.enable = F
}

# Remove variables
if (!exists("rm.variables")) {
  rm.variables = F
}

# Set the random seed in order to be able to reproduce the results
set.seed(1)

# Remove noisy instances using IPF
IPF.out = IPF(
  formula = class~.,
  data = dataset.tra.preprocessed.selected,
  nfolds = 10,
  consensus = T
)

## Get a summary of the noise elimination
summary(IPF.out, explicit = F)

prop.table(table(IPF.out$cleanData$class))
### The noise elimination has affected the positive class, which is
    ↪ the minoritary class.
### We need another approach...



# Remove noise using ENN

# Remove some points with ENN
ENN.out = ubENN(
  subset.numeric.dt(dataset.tra.preprocessed.selected),
  ifelse(
    dataset.tra.preprocessed.selected.cl == "negative",
    0,
    1
  ),
  verbose = T
```

```
)

## Get a summary of the results
print(sprintf(
  "Removed␣%d␣noisy␣instances␣(%.2f␣%%␣of␣the␣total)",
  nrow(dataset.tra.preprocessed.selected) - nrow(ENN.out$X),
  100 * ( 1 - nrow(ENN.out$X) / nrow(dataset.tra.preprocessed.
    ↪ selected))
))

## Retrieve filtered data
removed.instances = ENN.out$id.rm
my.clean.data     = dataset.tra.preprocessed.selected[-removed.
  ↪ instances, ]
my.clean.data.dt = dataset.tra.preprocessed.selected.dt[-removed.
  ↪ instances, ]
my.clean.data.cl = dataset.tra.preprocessed.selected.cl[-removed.
  ↪ instances]

## Check the relative frequencies of each class
print(
  prop.table(table(my.clean.data.cl))
)

### The proportion has improved (as expected)


# Some plots demonstrating that the removed points were indeed
  ↪ strange
# (only points from the negative class have been removed)
if (plot.enable) {
  ggplot(dataset.tra.preprocessed.selected, aes(X2, X3, color =
    ↪ class)) +
    geom_point(alpha = 0.1) +
    geom_point(data = dataset.tra.preprocessed.selected[removed.
      ↪ instances,], alpha = 0.5, color = "#FFCC00")

  ggplot(dataset.tra.preprocessed.selected, aes(X1, X5, color =
    ↪ class)) +
    geom_jitter(alpha = 0.1) +
    geom_jitter(data = dataset.tra.preprocessed.selected[removed.
      ↪ instances,], alpha = 0.5, color = "#FFCC00")

  ggplot(dataset.tra.preprocessed.selected, aes(X1, X2, color =
    ↪ class)) +
    geom_point(alpha = 0.1) +
    geom_point(data = dataset.tra.preprocessed.selected[removed.
      ↪ instances,], alpha = 0.5, color = "#FFCC00")

  ggplot(dataset.tra.preprocessed.selected, aes(X2, X8, color =
    ↪ class)) +
    geom_jitter(alpha = 0.1) +
    geom_jitter(data = dataset.tra.preprocessed.selected[removed.
      ↪ instances,], alpha = 0.5, color = "#FFCC00")
}


if (rm.variables) {
  rm(
    ENN.out,
    IPF.out,
```

```
104       removed.instances
105     )
106 }
```

End Of File 'noise-filter.R'

# D. Código fuente: Clasificación

## D.1. Código fuente usado en la clasificación de los datos originales

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: classification-orig.R                                       #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(dplyr)
library(caret)
library(DMwR)
library(purrr)
library(pROC)

# Beautiful tree graphics
if (class.opts$method == "rpart") {
  library(rattle)
}

# Paralelization
if (!is.na(class.opts$cores)) {
  library(doMC)
  registerDoMC(cores = class.opts$cores)
}


start.time = Sys.time()


# FUNCTIONS
test_roc = function(model, data) {
  # Calculate the ROC for the minoritary class with test data
  roc(
    data$class,
    predict(model, data, type = "prob")[, "positive"]
  )
}

test_accuracy = function(model, data) {
  return(confusionMatrix(predict(model, data), data$class)$overall["
    ↪ Accuracy"])
}

test_kappa = function(model, data) {
  return(confusionMatrix(predict(model, data), data$class)$overall["
    ↪ Kappa"])
}



# Setup control function for training
ctrl = trainControl(
  method = "repeatedcv",
```

```
  number = class.opts$cv$number,
  repeats = class.opts$cv$repeats,
  summaryFunction = twoClassSummary,
  verboseIter = TRUE,
  classProbs = TRUE
)


# Build a list to store the results
fit = list()

# Set the random seed
set.seed(1234)

# Learn a model with the original data
fit$original = caret::train(
  class ~ .,
  data = training.set,
  method = class.opts$method,
  metric = "ROC",
  trControl = ctrl
)

print(
  fit$original %>%
    test_roc(data = test.set) %>%
    auc()
)


# Use the same seed to ensure same cross-validation splits
ctrl$seeds = fit$original$control$seeds

end.time = Sys.time()
save.image(paste0(class.opts$method, "-orig", ".RData"))
```

End Of File 'classification-orig.R'

## D.2. Código fuente usado en la clasificación de los datos con submuestreo

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: classification-down.R                                       #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################


# Use the same seed to ensure same cross-validation splits
ctrl$seeds = fit$original$control$seeds


# Build down-sampled model
ctrl$sampling = "down"

fit$down = caret::train(
  class ~ .,
  data = training.set,
  method = class.opts$method,
  # verbose = FALSE,
  metric = "ROC",
  trControl = ctrl
)

print(
  fit$down %>%
    test_roc(data = test.set) %>%
    auc()
)


end.time = Sys.time()
save.image(paste0(class.opts$method, "-down",  ".RData"))
```

End Of File 'classification-down.R'

## D.3. Código fuente usado en la clasificación de los datos con sobremuestreo

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: classification-up.R                                         #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################


# Use the same seed to ensure same cross-validation splits
ctrl$seeds = fit$original$control$seeds


# Build up-sampled model
ctrl$sampling = "up"

fit$up = caret::train(
  class ~ .,
  data = training.set,
  method = class.opts$method,
  # verbose = FALSE,
  metric = "ROC",
  trControl = ctrl
)

print(
  fit$up %>%
    test_roc(data = test.set) %>%
    auc()
)


end.time = Sys.time()
save.image(paste0(class.opts$method, "-up",  ".RData"))
```

End Of File 'classification-up.R'

## D.4. Código fuente usado en la clasificación de los datos con sobremuestreo (SMOTE)

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: classification-smote.R                                      #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################


# Use the same seed to ensure same cross-validation splits
ctrl$seeds = fit$original$control$seeds


# Build up-sampled model using SMOTE
ctrl$sampling = "smote"

fit$smote = caret::train(
  class ~ .,
  data = training.set,
  method = class.opts$method,
  # verbose = FALSE,
  metric = "ROC",
  trControl = ctrl
)

print(
  fit$smote %>%
    test_roc(data = test.set) %>%
    auc()
)


end.time = Sys.time()
save.image(paste0(class.opts$method, "-smote",  ".RData"))
```

End Of File 'classification-smote.R'

## D.5. Código fuente usado en la visualización de los resultados producidos por los diferentes modelos de clasificación

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: classification-visualization.R                              #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Load required libraries
library(dplyr)
library(caret)
library(DMwR)
library(purrr)
library(pROC)

# Beautiful tree graphics
if (class.opts$method == "rpart") {
  library(rattle)
}

# Visualize the results
## Calculate ROC for every element in the 'fit' list
model_list_roc = fit %>%
  map(test_roc, data = test.set)

## Calculate the AUC
print(
  model_list_roc %>%
    map(function(x) return(auc(x)))
)

# Get a table to put in the document
test_measures = data.frame(AUC = as.numeric(lapply(model_list_roc,
    ↪ function(x) return(auc(x)))))
test_measures$Accuracy = as.numeric(lapply(fit, test_accuracy, data
    ↪ = test.set))
test_measures$Kappa = as.numeric(lapply(fit, test_kappa, data = test
    ↪ .set))
rownames(test_measures) = names(fit)

# PLOT
# Build a dataframe to plot
results_list_roc = list()
i = 1

for(the_roc in model_list_roc){
  results_list_roc[[i]] =
    data_frame(
      tpr = the_roc$sensitivities,
      fpr = 1 - the_roc$specificities,
      model = names(fit)[i]
    )

  if (class.opts$method == "rpart") {
    print(fancyRpartPlot(fit[[i]]$finalModel, sub = names(fit)[i]))
```

```
  }

  i = i + 1
}

results_df_roc = bind_rows(results_list_roc)


# Plot ROC curve for all tested models
print(
  ggplot(aes(x = fpr,  y = tpr, group = model), data = results_df_roc) +
    geom_abline(intercept = 0, slope = 1, color = "gray", size = 1) +
    geom_line(aes(color = model), size = 1)
)
```

End Of File 'classification-visualization.R'

# E. Código fuente: Ejecución

## E.1. Código fuente del script para la ejecución de todo el proceso

```
#####################################################################
#       Mineria de Datos: Preprocesamiento y Clasificacion          #
#                                                                   #
# FILE: mdpc-script.R                                               #
#                                                                   #
# (C) Cristian González Guerrero                                    #
#####################################################################

# Clear workspace
closeAllConnections()
rm(list=ls())

# Visualization option
if (!exists("plot.enable")) {
  plot.enable = F
}

# Remove variables option
if (!exists("rm.variables")) {
  rm.variables = F
}

# Load options
source("mdpc-options.R")

# Load data
source("load-data.R")

# Change the default training and test sets
k = mdpc.options$data.load$k

dataset.tra = dataset.train(k)
dataset.tra.cl = dataset.tra$class
dataset.tra.dt = subset(dataset.tra, select = -class)
dataset.tst = dataset.test(k)
dataset.tst.cl = dataset.tst$class
dataset.tst.dt = subset(dataset.tst, select = -class)

# Visualization
if (plot.enable) {
  source("initial-visualization.R")
}

# Preprocessing
if (!is.na(mdpc.options$preprocessing$outliersUNI)) {
  source("outlier-removal-UNI.R")
}

if (!is.na(mdpc.options$preprocessing$outliersMCD)) {
  source("outlier-removal-MCD.R")
}

source("outlier-removal.R")

```

```
55
56 if (!is.na(mdpc.options$preprocessing$discretization)) {
57   source("discretization.R")
58 }
59
60 if (!exists("dataset.tra.preprocessed")) {
61   dataset.tra.preprocessed = dataset.tra
62   dataset.tra.preprocessed.dt = dataset.tra.dt
63   dataset.tra.preprocessed.cl = dataset.tra.cl
64 }
65
66 if (!is.na(mdpc.options$preprocessing$attribute.selection)) {
67   source("attribute-selection.R")
68 }
69
70
71 if (!is.na(mdpc.options$preprocessing$noise.filter)) {
72   source("noise-filter.R")
73 }
74
75 # Classification
76 if (!is.na(mdpc.options$preprocessing$noise.filter)) {
77   if (mdpc.options$preprocessing$noise.filter == "IPF") {
78     training.set = IPF.out$cleanData
79   } else if (mdpc.options$preprocessing$noise.filter == "ENN") {
80     training.set = my.clean.data
81   }
82 } else if (!is.na(mdpc.options$preprocessing$discretization)) {
83   training.set = dataset.tra.preprocessed.discretized
84 } else if (!is.na(mdpc.options$preprocessing$attribute.selection)) {
85   cor.cutoff = mdpc.options$preprocessing$attribute.selection
86   training.set = select.attributes(dataset.tra.preprocessed, cor.
       ↪ cutoff)
87 } else {
88   training.set = dataset.tra.preprocessed
89 }
90
91 test.set = dataset.tst
92
93 class.opts = mdpc.options$classification
94
95 source("classification-orig.R")
96
97 if (!is.na(class.opts$down)) {
98   source("classification-down.R")
99 }
100
101 if (!is.na(class.opts$up)) {
102   source("classification-up.R")
103 }
104
105 if (!is.na(class.opts$smote)) {
106   source("classification-smote.R")
107 }
108
109 # Visualize the results
110 if (plot.enable) {
111   source("classification-visualization.R")
112 }
```

End Of File 'mdpc-script.R'

## E.2. Código fuente de las opciones globales

```
mdpc.options = list()


# Options for DATA LOAD
mdpc.options$data.load = list(
  # Fold for test set (the other ones will be used as training set)
  k = 5
)


# Options for PREPROCESSING
mdpc.options$preprocessing = list(
  # Ouliers removal (univariate)
  outliersUNI = T,

  # Ouliers removal (multivariate)
  outliersMCD = T,

  # Discretization
  discretization = NA,

  # Attribute selection (correlation cutoff for redundant attributes
    ↪ )
  attribute.selection = 1.0,

  # Noise filtering (select either "ENN" or "IPF")
  noise.filter = "IPF"
)


# Options for CLASSIFICATION
mdpc.options$classification = list(
  # Method (select a method, such as rpart, rf of svmLinear)
  method = "rpart",

  # Number of cores to use in paralelization
  cores = 3,

  # Cross validation options
  cv = list(
    number = 10,
    repeats = 2
  ),

  # Input downsampled data
  down = T,

  # Input upsampled data
  up = NA,

  # Input upsampled data (using SMOTE)
  smote = NA
)
```

End Of File 'mdpc-options.R'