Minería de Datos: Preprocesamiento y Clasificación
Memoria de la competición en Kaggle

Javier Poyatos Amador
Elena Romero Contreras
Juan Luis Suárez Díaz
Marta Verona Almeida

8 de marzo de 2019

1. Introducción

Este documento describe el trabajo realizado para la competición en Kaggle de la asignatura Minería de Datos: Preprocesamiento y Clasificación. En primer lugar, se presentará un análisis del conjunto de datos, con el que se han extraído algunas de las conclusiones más relevantes acerca del conjunto de datos, que será de gran utilidad en las técnicas de preprocesamiento que se utilizarán más adelante. En segundo lugar, se describirán las técnicas de preprocesamiento empleadas a nivel general a lo largo de la competición y, finalmente, se explicarán los procedimientos y soluciones elaboradas en cada una de las competiciones.

Cada componente del grupo se ha encargado de una competición, siendo el reparto de competiciones el siguiente:

- Árboles de clasificación: Javier.
- Ripper: Elena.
- K-NN: Juan Luis.
- SVM: Marta.

Junto con este documento se envía adjunto un conjunto de scripts en R que han sido utilizados para analizar los datos, preprocesarlos y predecir las soluciones enviadas a la plataforma Kaggle. Los scripts principales que contienen los algoritmos usados para cada competición pueden encontrarse en los ficheros `arboles.R`, `ripper.R`, `knn.R` y `svm.R`, respectivamente. Para su ejecución puede resultar necesario cargar primero el fichero `preprocesado.R`, que contiene algunas de las implementaciones de los métodos de preprocesado generales utilizados.

2. Análisis del conjunto de datos

En esta sección mostraremos algunas de las características de mayor interés observadas sobre nuestro conjunto de datos, que serán de utilidad más adelante a la hora de seleccionar las técnicas de preprocesamiento que aplicaremos. Suponemos que los datos de los conjuntos de entrenamiento y test están almacenados en variables `train` y `test`, respectivamente. En primer lugar, obtenemos un resumen de los datos mediante la función `summary`.

```
## ANÁLISIS

# Resumen de los datos (mostramos solo algunas variables por cuestiones de espacio).
summary(train)[,c(1:5,ncol(train))]
```

```
##        X1                  X2                   X3
##  Min.   :-68931.0   Min.   :-68931.94   Min.   :-68932
##  1st Qu.:   276.8   1st Qu.:    60.91   1st Qu.:  3213
##  Median :   331.0   Median :    96.48   Median :  7232
##  Mean   :   108.0   Mean   :  -129.94   Mean   :  8498
##  3rd Qu.:   388.7   3rd Qu.:   136.53   3rd Qu.:  9532
##  Max.   :  1034.8   Max.   :   498.51   Max.   :327589
##  NA's   :26         NA's   :17          NA's   :24
##        X4                   X5            C
##  Min.   :-68931.00   Min.   :-68932.09   0:5995
##  1st Qu.:    16.62   1st Qu.:     0.00   1:3149
##  Median :    18.78   Median :     0.37
##  Mean   :  -214.70   Mean   :  -233.51
##  3rd Qu.:    21.83   3rd Qu.:     0.95
##  Max.   :    43.93   Max.   :     7.96
##  NA's   :19          NA's   :18
```

```
summary(test)[,c(1:5)]
```

```
##        X1                   X2                   X3
##  Min.   :-68931.00   Min.   :-68931.00   Min.   :-68931
##  1st Qu.:   275.48   1st Qu.:    59.33   1st Qu.:  3315
##  Median :   330.40   Median :    95.90   Median :  7285
##  Mean   :    88.18   Mean   :  -143.67   Mean   :  8749
##  3rd Qu.:   387.74   3rd Qu.:   135.53   3rd Qu.:  9690
##  Max.   :   875.58   Max.   :   478.33   Max.   :304910
##        X4                   X5
##  Min.   :-68931.00   Min.   :-68931.00
##  1st Qu.:    16.73   1st Qu.:     0.00
##  Median :    18.79   Median :     0.34
##  Mean   :  -226.71   Mean   :  -245.59
##  3rd Qu.:    21.78   3rd Qu.:     0.90
##  Max.   :    42.53   Max.   :     6.13
```

El resumen de los datos nos muestra que el conjunto de datos está compuesto por 50 variables predictoras, todas ellas numéricas, tomando cada una valores en rangos muy distintos. La variable clase, `C`, toma los valores `0` y `1`. Además de todo esto, podemos hacer una primera observación interesante observando este resumen. Podemos ver que las medias de los atributos toman en general valores bastante negativos, y muy distintos a los de la mediana. Además, si observamos los mínimos de cada atributo, siempre toman un valor extraño (algo inferior a -68000), cuando en realidad los datos de cada atributo parecen moverse en rangos muy distintos. Analizamos con detalle los ejemplos en los que ocurre esto.

```
train[apply(train[,-ncol(train)], MARGIN=1, function(x) any(!is.na(x) & x < -68000)),
            c(1:5,ncol(train))]
```

```
##             X1        X2        X3        X4        X5 C
## 395  -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
```

```
## 545  -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 772  -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 1051 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 1242 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 1308 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 1
## 1365 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 1
## 1698 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 1919 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 1983 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 2042 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 2342 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 2674 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 2995 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 3078 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 3960 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 4197 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 4230 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 1
## 4990 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 5282 -68930.20 -68931.10 -68932.50 -68930.79 -68932.09 0
## 5629 -68929.06 -68931.94 -68929.92 -68929.59 -68930.83 1
## 5754 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 1
## 5942 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 6039        NA -68930.39 -68929.98 -68930.44 -68931.68 0
## 6443 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 1
## 6702 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 7108 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 7136 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 7429 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 7582 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 0
## 7654 -68931.00 -68931.00 -68931.00 -68931.00 -68931.00 1
```

Vemos que en realidad, los ejemplos en los que ocurre esto tienen este valor extraño en todos sus atributos. Por tanto, tenemos un conjunto de datos posiblemente erróneos en nuestro dataset, que toman un mismo valor muy negativo en todas las variables y que, en principio, pueden pertenecer a ambas clases. Estos datos apenas aportan información (al estar concentrados todos en un mismo punto), pero pueden estorbar mucho en los algoritmos que sean sensibles a outliers, o en aquellos que necesiten un preprocesamiento como la normalización (el cálculo de medidas como desviación típica para el z-score o mínimos para min-max se verán gravemente afectados). Por ello, la opción más razonable es eliminarlos del conjunto de entrenamiento. El problema es que también hay datos de este tipo en el conjunto test:

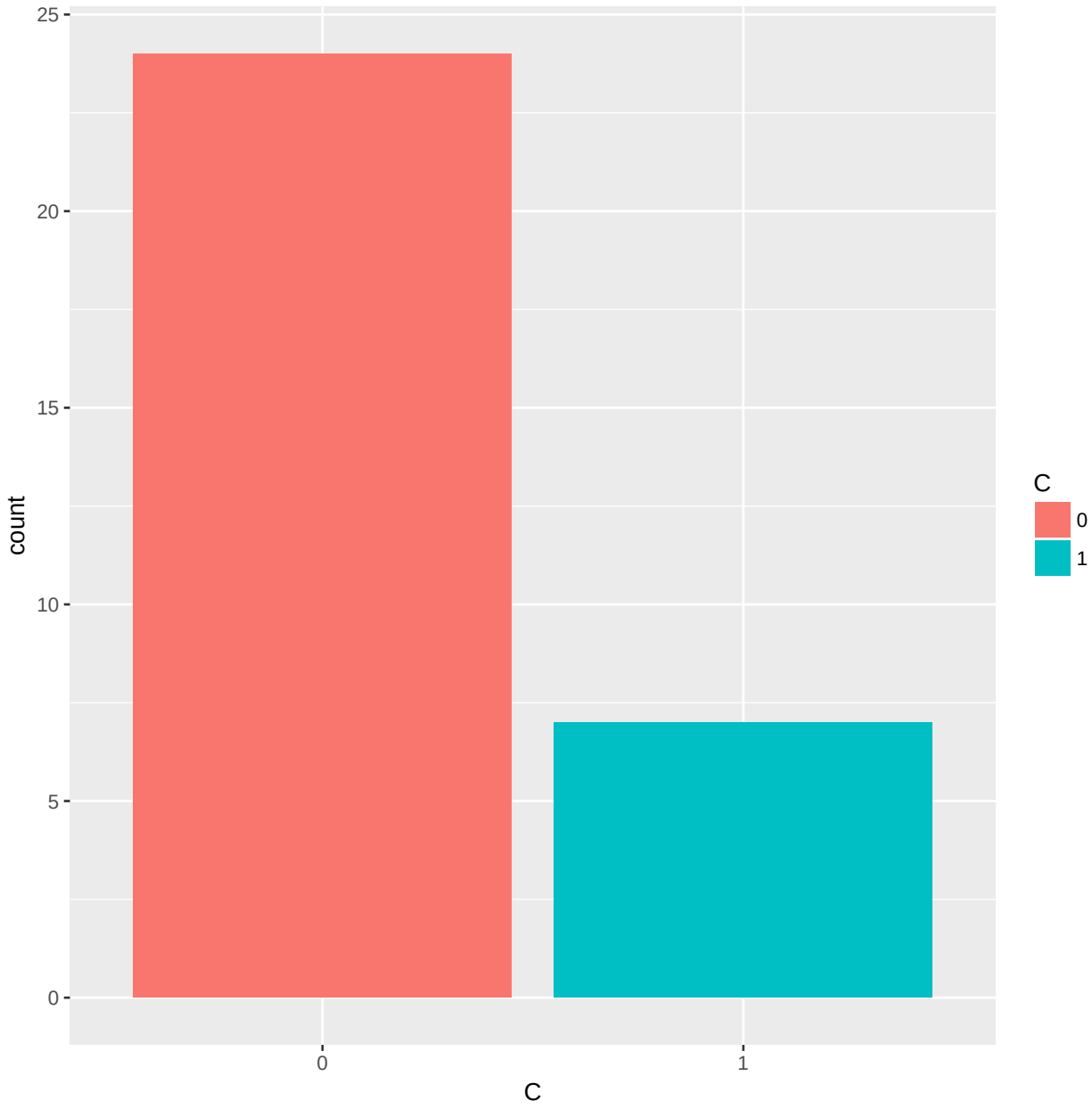
```
test[apply(test, MARGIN=1, function(x) any(!is.na(x) & x < -68000)),1:5]
```

```
##          X1     X2     X3     X4     X5
## 111  -68931 -68931 -68931 -68931 -68931
## 231  -68931 -68931 -68931 -68931 -68931
## 660  -68931 -68931 -68931 -68931 -68931
## 848  -68931 -68931 -68931 -68931 -68931
## 923  -68931 -68931 -68931 -68931 -68931
## 1213 -68931 -68931 -68931 -68931 -68931
## 1822 -68931 -68931 -68931 -68931 -68931
```

```
## 2107 -68931 -68931 -68931 -68931 -68931
## 2176 -68931 -68931 -68931 -68931 -68931
## 2783 -68931 -68931 -68931 -68931 -68931
## 2870 -68931 -68931 -68931 -68931 -68931
## 3460 -68931 -68931 -68931 -68931 -68931
## 3468 -68931 -68931 -68931 -68931 -68931
## 3810 -68931 -68931 -68931 -68931 -68931
```

A estos datos tendremos que asignarles también una etiqueta, pero si en los datos de entrenamiento hemos eliminado los ejemplos de este tipo, el clasificador aprendido no va a poder realizar ninguna asignación de etiquetas razonable. Al ser unos datos tan distintos del resto del conjunto la única información que podría ayudarnos a realizar una asignación más precisa es la de los propios outliers de este tipo que hay en el conjunto de entrenamiento. Vemos para ello cómo se distribuyen las clases sobre estos datos.

```
library(ggplot2)
ggplot(train[apply(train[,-ncol(train)], MARGIN=1, function(x) any(!is.na(x) & x < -68000)),],
aes(x=C, fill=C)) + geom_histogram(stat="count")
```



Vemos que hay muchos más outliers extremos que toman la clase 0 respecto a los que tienen clase 1 en el conjunto de entrenamiento. Usando esta información, una de las asignaciones más razonables para estos outliers en el conjunto test puede consistir en clasificarlos todos con la clase 0. Almacenaremos los índices de estos outliers en variables por si es necesario ignorarlos en los análisis que realizaremos a continuación.

```
# Índices de estos outliers por si hay que quitarlos
outliers.train.extremos <- which(apply(train[,-ncol(train)], MARGIN=1,
                                 function(x) any(!is.na(x) & x < -68000)))
outliers.test.extremos <- which(apply(test, MARGIN=1,
                                function(x) any(!is.na(x) & x < -68000)))
```

Pasamos ahora a observar los valores perdidos presentes en los datos. Podemos ver que el conjunto de entrenamiento presenta 1366 ejemplos con valores perdidos, mientras que el conjunto test está limpio.

```
# Valores perdidos
has.na <- function(x) apply(x,1,function(z)any(is.na(z)))
sum(has.na(train)) # 1366 ejemplos con na en train
```

```
## [1] 1366
```

```
sum(has.na(test))  # No hay NAs en test
```

```
## [1] 0
```

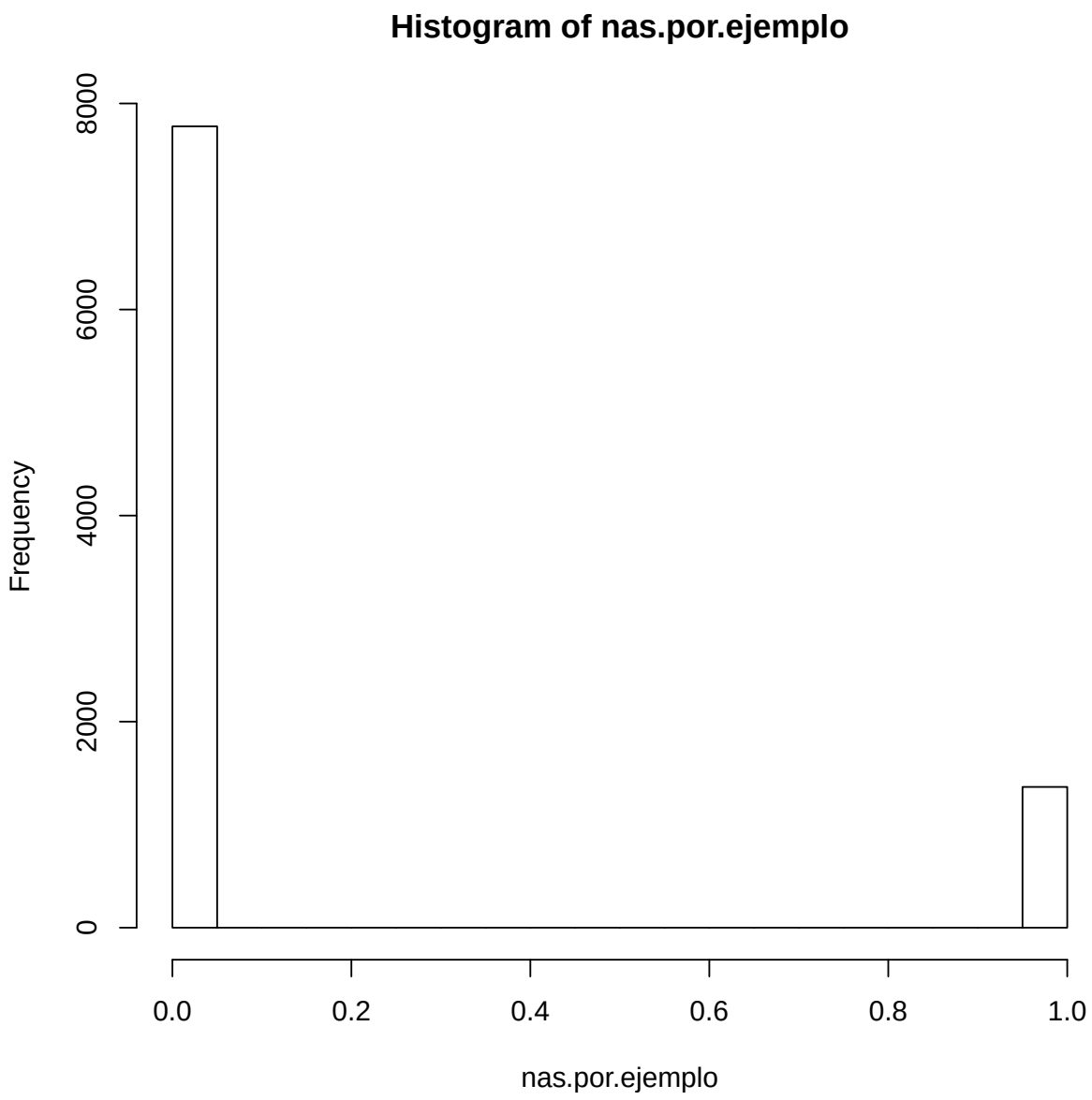
Si analizamos los valores perdidos por variable, vemos que están distribuidos de forma bastante equilibrada entre todas las variables. No hay ninguna variable que presente una cantidad excesiva de valores perdidos, y todas ellas presentan algunos.

```
# NAs por variable
apply(train, 2, function(x) sum(is.na(x)))
```

```
##  X1  X2  X3  X4  X5  X6  X7  X8  X9 X10 X11 X12 X13 X14 X15 X16 X17 X18
##  26  17  24  19  18  35  19  25  50  26  31  24  35  18  29  28  32  22
## X19 X20 X21 X22 X23 X24 X25 X26 X27 X28 X29 X30 X31 X32 X33 X34 X35 X36
##  28  28  22  27  31  27  15  32  23  22  28  29  32  41  26  28  26  20
## X37 X38 X39 X40 X41 X42 X43 X44 X45 X46 X47 X48 X49 X50   C
##  29  32  28  38  35  34  31  18  28  18  31  34  26  21   0
```

Si analizamos los valores perdidos por ejemplo, vemos que no hay ningún ejemplo con más de un valor perdido. Todos ellos tienen como mucho uno, tal y como observamos en el histograma de los valores perdidos por ejemplo. Las características de los valores perdidos facilitan el uso de distintos métodos de imputación que discutiremos más adelante.

```
# NAs por ejemplo
nas.por.ejemplo <- as.integer(apply(train, 1, function(x) sum(is.na(x))))
hist(nas.por.ejemplo)
```

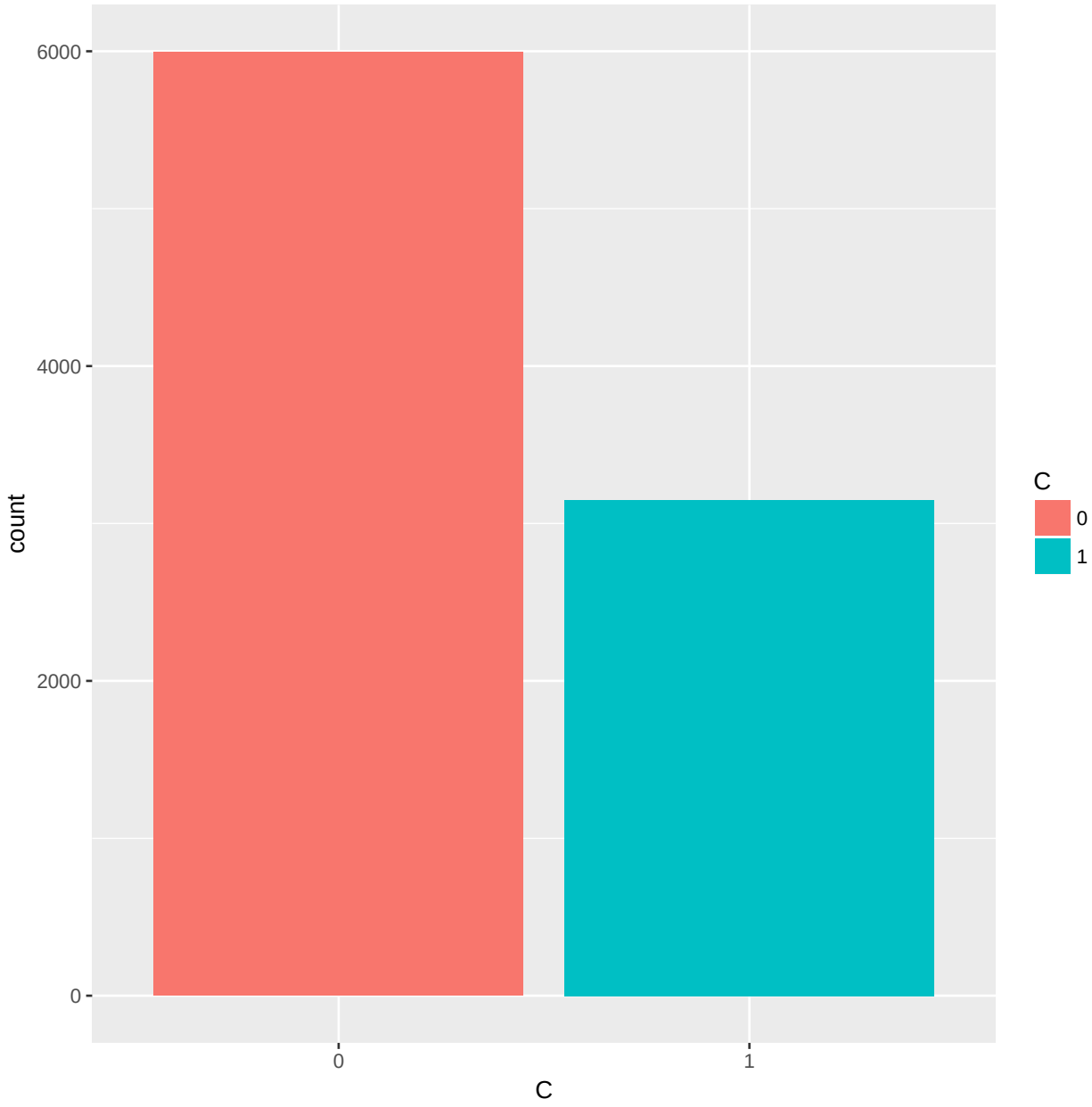


Almacenamos los índices de los valores perdidos, por si es necesario ignorarlos en las siguientes partes del análisis.

```
indices.nas.train <- which(has.na(train))
```

Si observamos cómo se distribuyen las clases sobre los datos de entrenamiento, vemos que presentan un ligero desbalanceo (en torno a dos datos de clase 0 por cada dato de clase 1). No es una tasa de desbalanceo muy fuerte, pero en algunos algoritmos podría resultar interesante corregir este desbalanceo.

```
library(dplyr)
# Histograma de distribución de clases
ggplot(train, aes(x=C, fill=C)) + geom_histogram(stat="count")
```

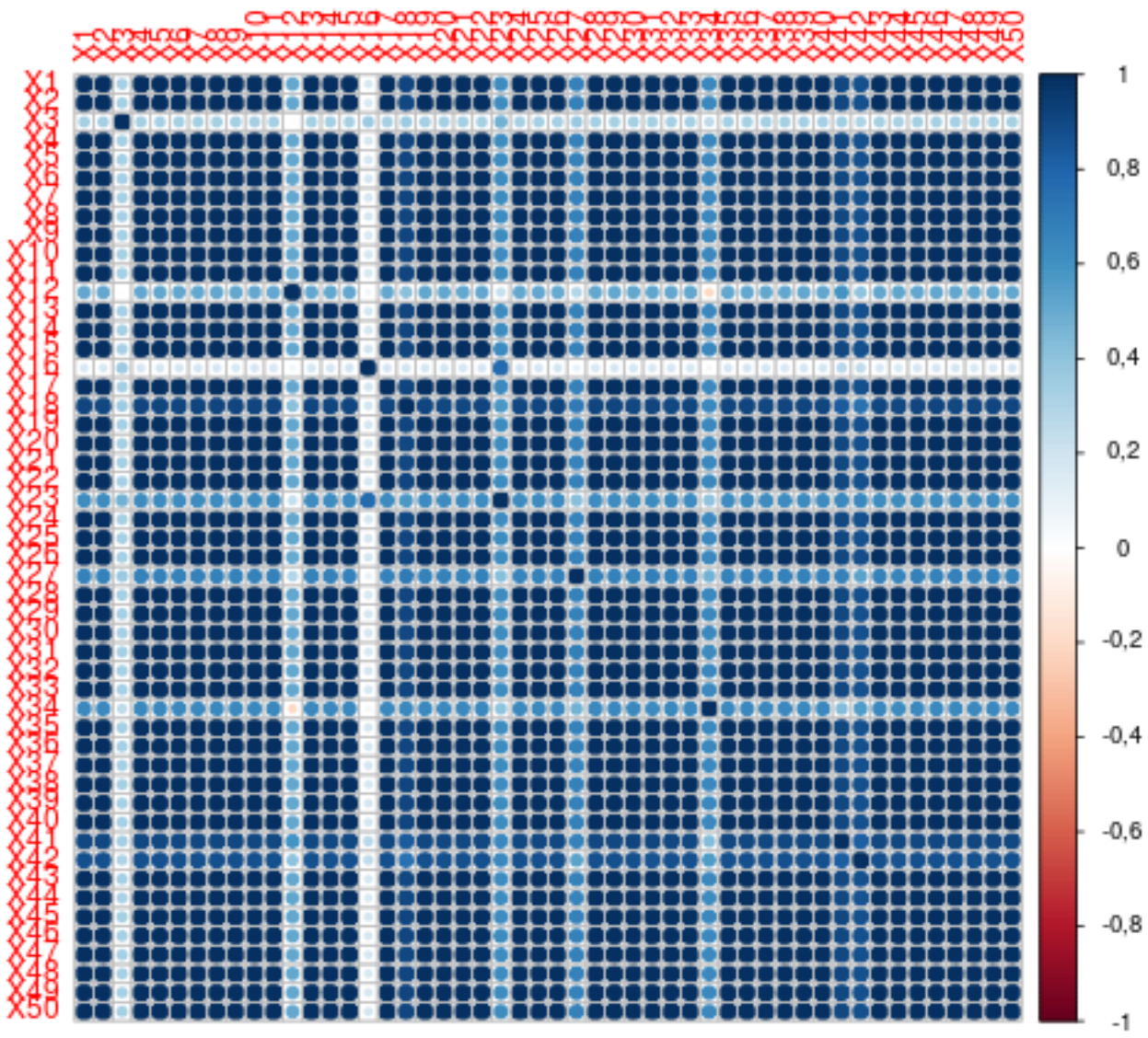



```
# Número de ejemplos por clase
train %>%
  group_by(C) %>%
  summarize(n = n())
```

```
## # A tibble: 2 x 2
##        C     n
##   <fctr> <int>
## 1      0  5995
## 2      1  3149
```

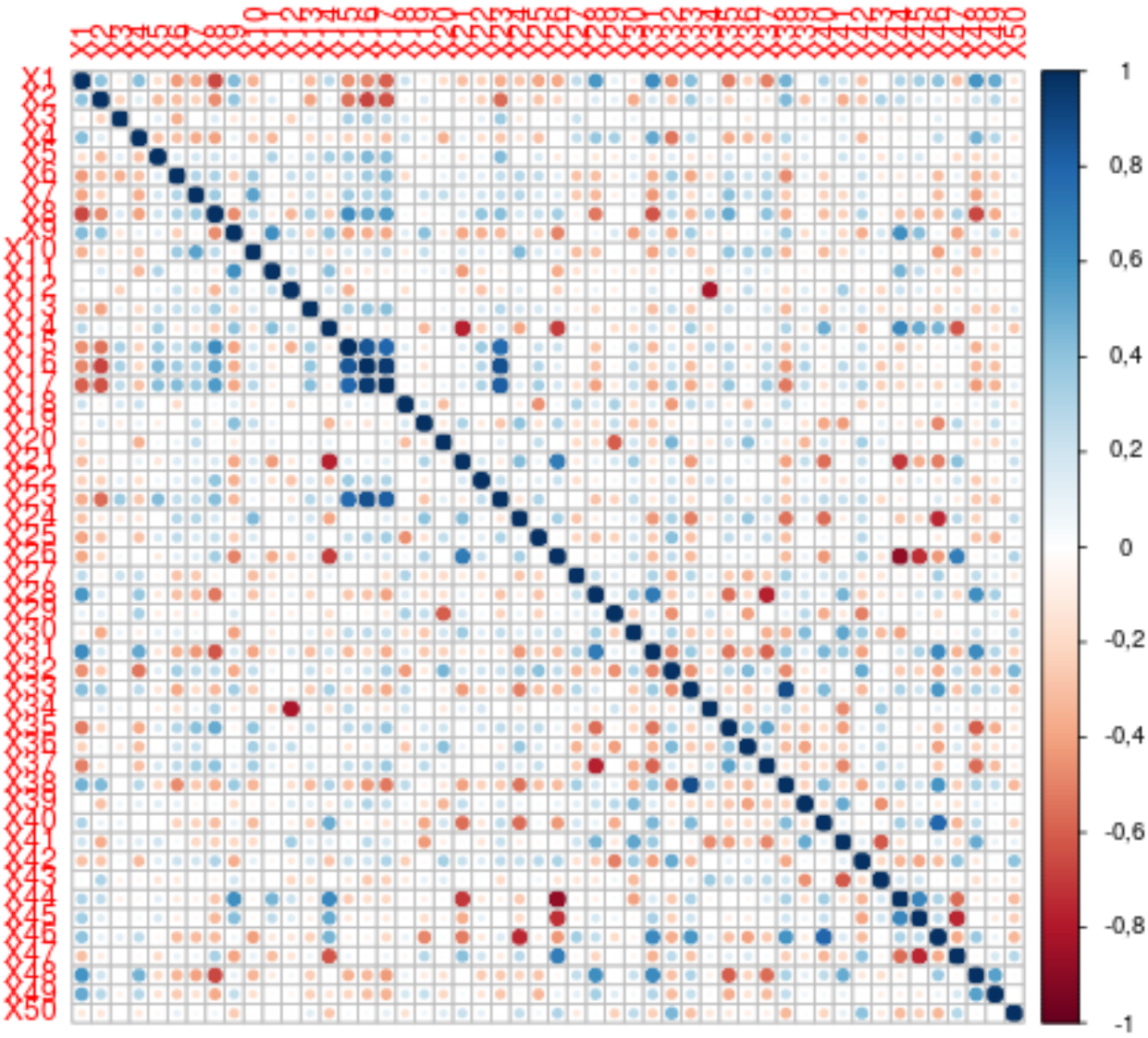
A continuación analizamos las correlaciones entre los distintos atributos del conjunto de datos. Si analizamos los datos originales tal cual vamos a ver que las correlaciones apenas aportan información. Los valores son altamente desviados por los outliers extremos.

```
library(corrplot)
# Correlaciones
corrplot(cor(train[-c(indices.nas.train),-ncol(train)]))
```



Por tanto, si queremos calcular correlaciones de forma adecuada, tenemos que eliminar los outliers extremos. De esta forma, obtenemos el siguiente mapa de calor:

```
corrplot(cor(train[-c(indices.nas.train, outliers.train.extremos),-ncol(train)]))
```



En el mapa anterior sí que podemos ver unas correlaciones más razonables. En general, podemos observar que la mayoría de entradas de la matriz no tienen valores elevados. Vamos a centrarnos en los pocos valores que presentan una correlación más elevada.

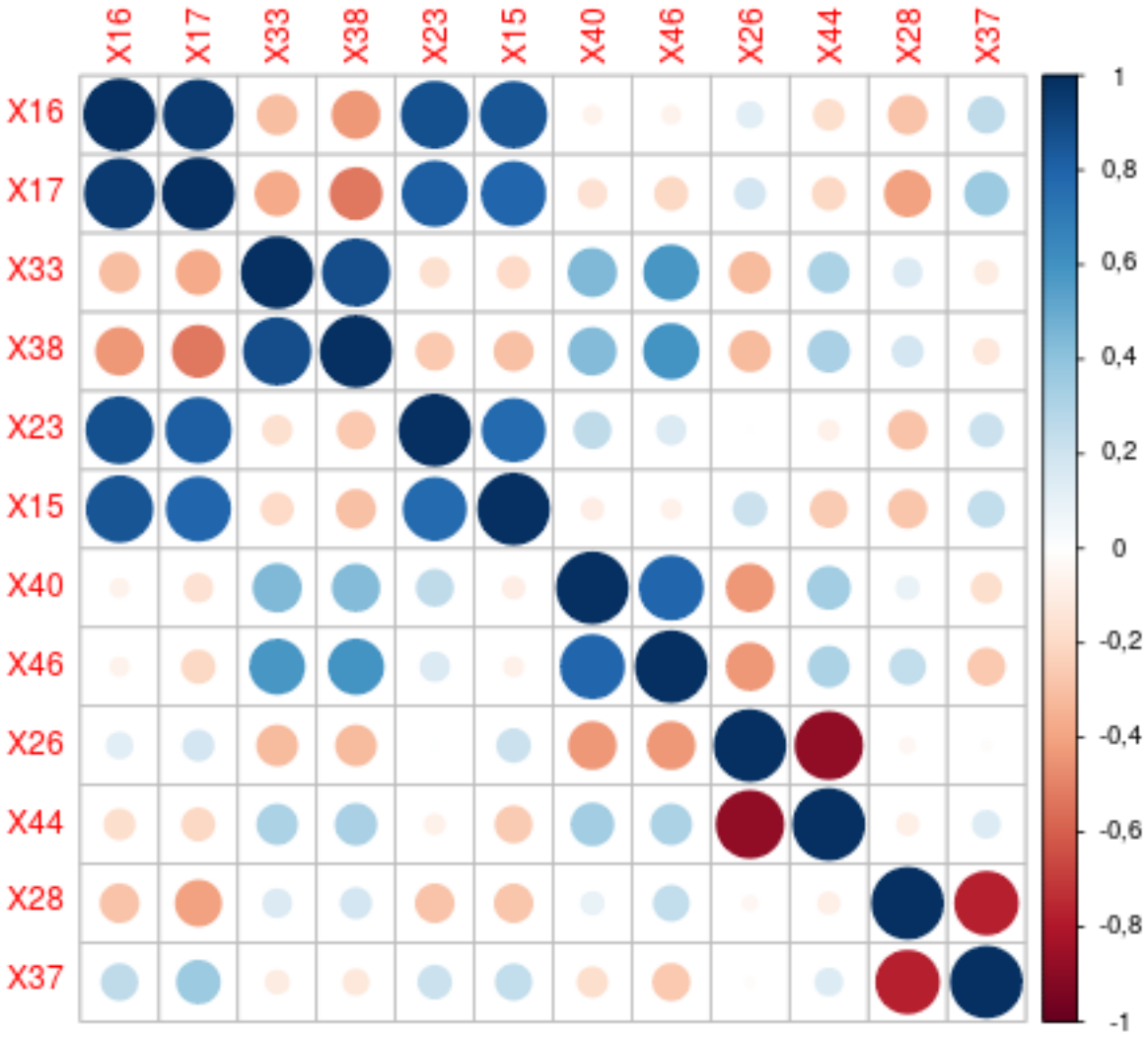
```
library(tidyr)
# Variables más correladas
cor(train[-c(indices.nas.train, outliers.train.extremos),-ncol(train)]) %>%
  as.data.frame() %>%
  mutate(var1 = rownames(.)) %>%
  gather(var2, value, -var1) %>% # Posibles combinaciones (par de variables, correlación)
  arrange(desc(abs(value))) %>% # Ordenamos según correlación (en valor absoluto)
  filter(var1 < var2) %>% # Eliminamos redundantes (y correlaciones de variables consigo misma)
```

```
  head(n=10)
```

var1	var2	Value
X16	X17	0.9526716
X26	X44	-0.8867555
X33	X38	0.8804476
X16	X23	0.8704657
X15	X16	0.8560844
X17	X23	0.8256626
X12	X34	-0.8143807
X15	X17	0.7999134
X40	X46	0.7946815
X28	X37	-0.7796407

Volvemos a observar la matriz de correlaciones, ahora ampliada sobre las variables anteriores.

```
corrplot(cor(train[-c(indices.nas.train, outliers.train.extremos),
             c(16,17,33,38,23,15,40,46,26,44,28,37)]))
```



Analizamos ahora las nubes de puntos entre las variables más correladas. Comenzamos con X16 y X17.

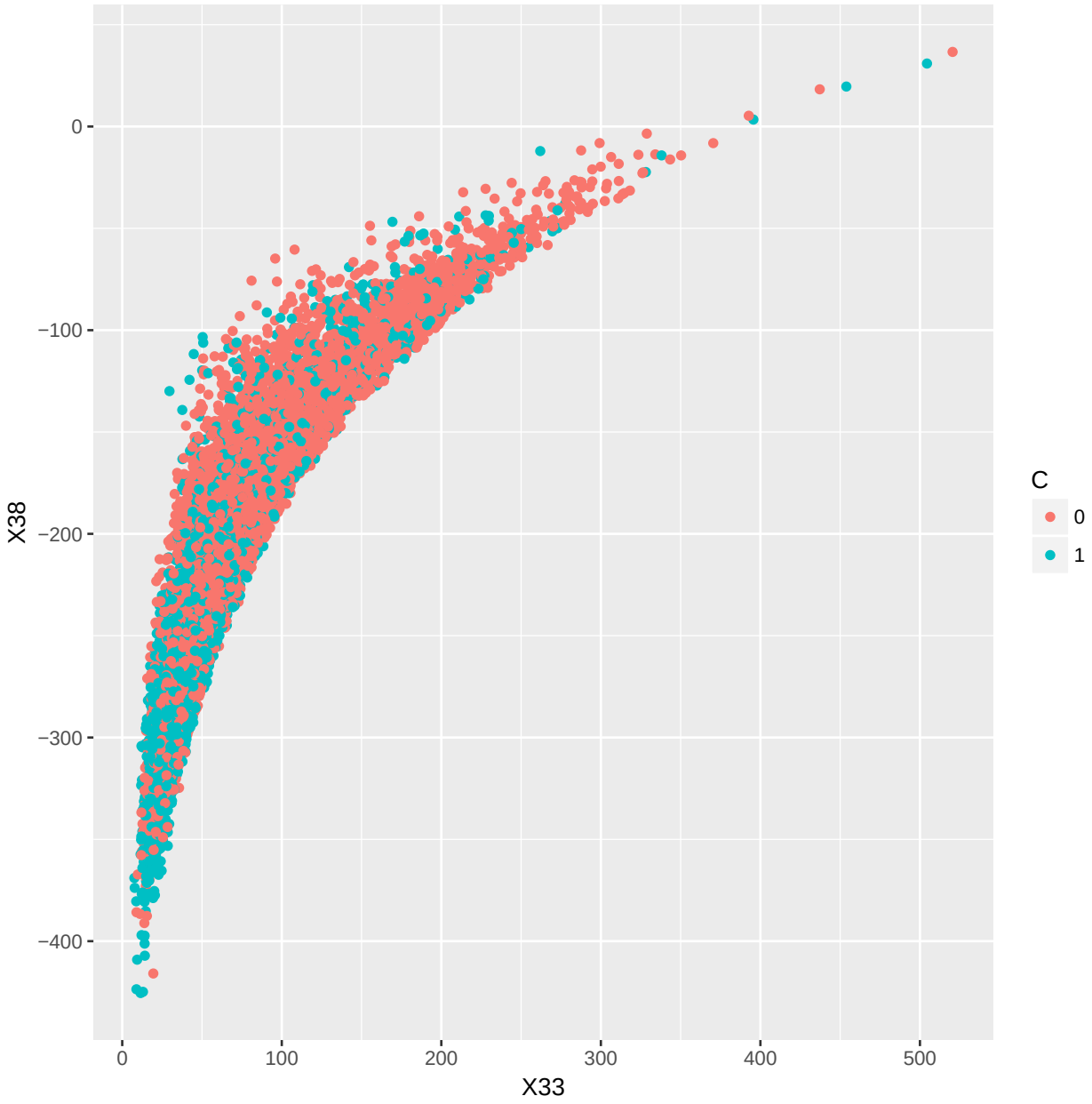
```
ggplot(train[-outliers.train.extremos,], aes(x=X16, y=X17, col=C)) + geom_point()
```

Entre X16 y X17 se aprecia una dependencia lineal bastante clara. Puede ser de utilidad quedarnos con una sola de estas variables, y así reducir el posible ruido que puedan introducir, a la hora de seleccionar características para los clasificadores. Observamos ahora la relación entre X26 y X44.

```
ggplot(train[-outliers.train.extremos,], aes(x=X26, y=X44, col=C)) + geom_point()
```

Entre X26 y X44 se aprecia también una dependencia lineal fuerte. Ambas distribuciones parecen ser bastante asimétricas, pues los datos se concentran en la zona inferior derecha de la nube de puntos. Analizamos ahora X33 frente a X38.

```
ggplot(train[-outliers.train.extremos,], aes(x=X33, y=X38, col=C)) + geom_point()
```



Entre estas variables se aprecia una relación bastante peculiar. La nube de puntos determinada por ambas variables parece estar encerrada entre dos curvas suaves. No podemos decir que haya una correlación lineal pura entre ambas variables, pero sí parece que claro que ambas variables se relacionan de alguna forma. Veamos qué ocurre entre X16 y X23.

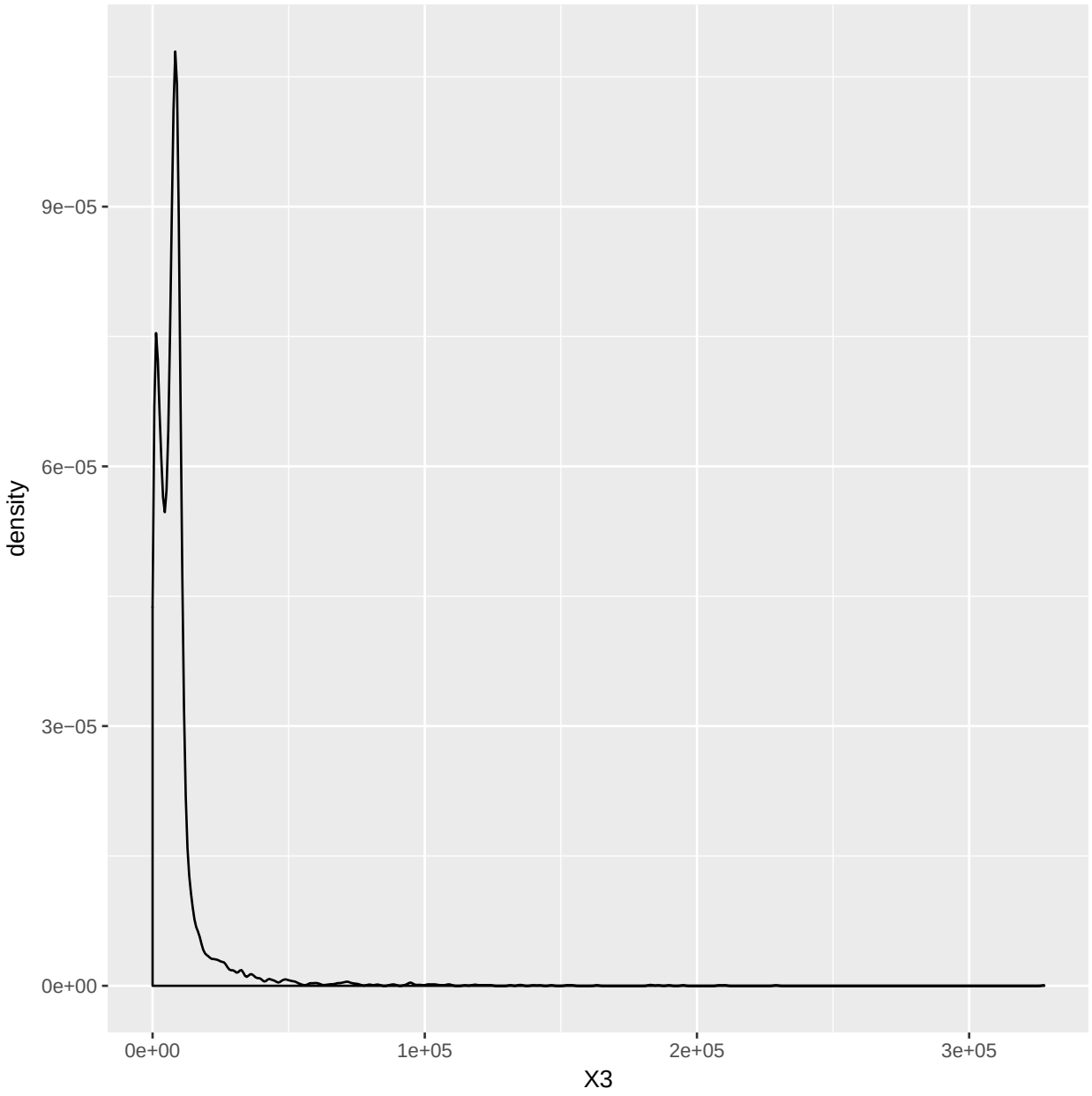
```
ggplot(train[-outliers.train.extremos,], aes(x=X16, y=X23, col=C)) + geom_point()
```

De nuevo vemos una relación clara entre ambas variables, aunque las nubes de puntos empiezan a presentar una dispersión algo mayor. Lo mismo ocurre en la siguiente gráfica, que compara X15 y X16.

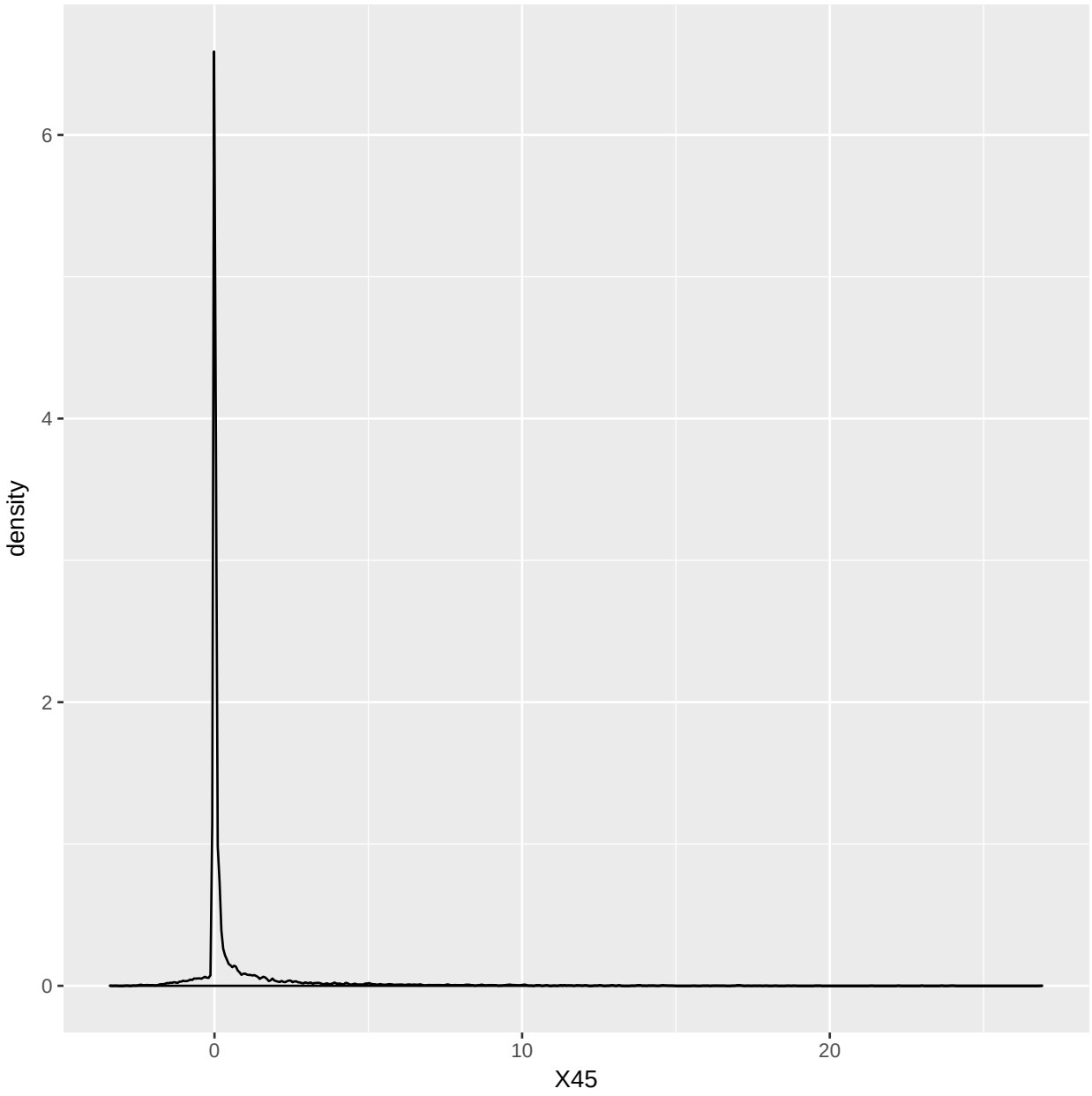
```
ggplot(train[-outliers.train.extremos,], aes(x=X15, y=X16, col=C)) + geom_point()
```

Por último, vamos a ver cómo se distribuyen los datos. Podemos ver que, en general, muchas de las variables presentan distribuciones muy asimétricas. A continuación mostramos a modo de ejemplo las distribuciones de las variables X3 y X45.

```
ggplot(train[-outliers.train.extremos,], aes(x=X3)) + geom_density()
```

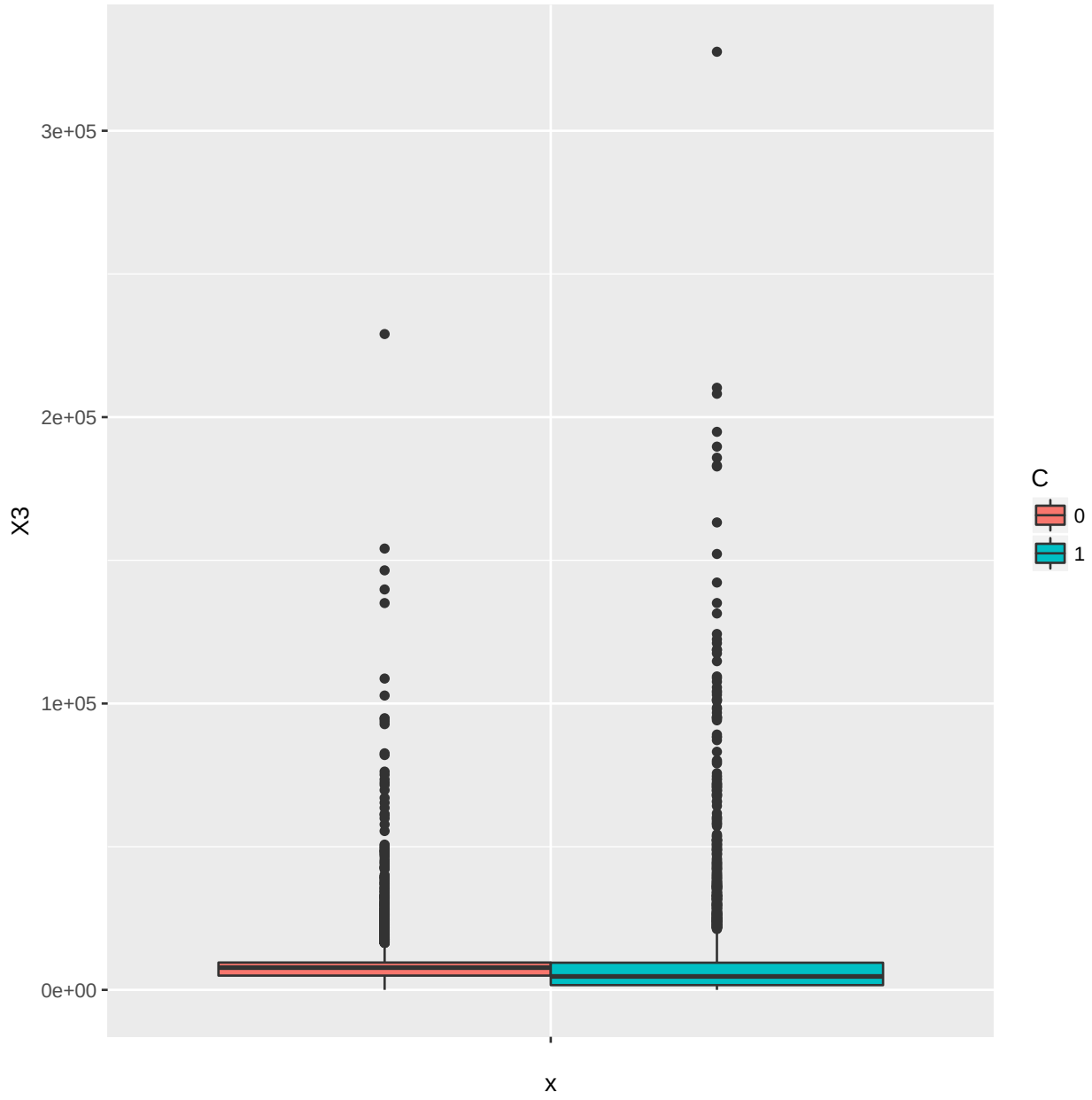



```
ggplot(train[-outliers.train.extremos,], aes(x=X45)) + geom_density()
```

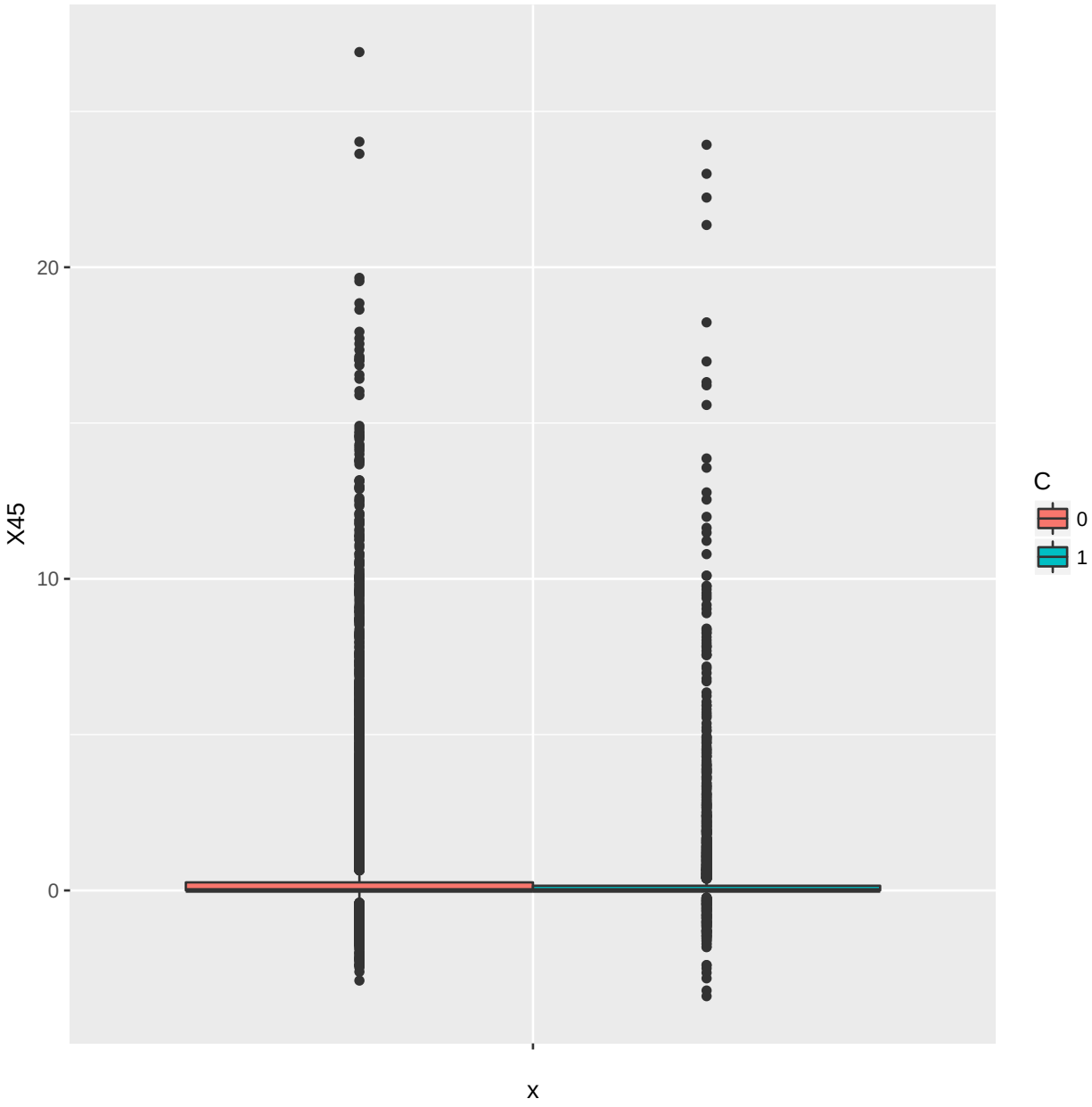


También podemos observar la distribución por clases.

```
ggplot(train[-outliers.train.extremos,], aes(x="", y=X3, fill=C)) + geom_boxplot()
```




```
ggplot(train[-outliers.train.extremos,], aes(x="", y=X45, fill=C)) + geom_boxplot()
```



La asimetría puede ser perjudicial en algunos algoritmos o técnicas de preprocesamiento. Por ello, puede resultar útil aplicar transformaciones a las variables que reduzcan esta asimetría. Las principales transformaciones que contribuyen a dicha reducción son logaritmos, raíces cuadradas, raíces cúbicas y cuadrados. En la siguiente sección se explican las transformaciones que se han barajado. Otro detalle de interés sobre la distribución de las variables, es que algunas de ellas parecen concentrarse en regiones positivas cerca del cero, tomando algunos valores negativos muy cercanos a cero. Es posible que, en realidad, dichas variables tomen siempre valores no negativos y los valores observados en el conjunto de entrenamiento se deban a ruido. Esto ocurre, por ejemplo, con X6, X23, X35 y X45. Convertir los valores negativos a 0 puede ser otra posible opción para limpiar los datos.

3. Preprocesamiento general

Uno de los apartados más importantes a la hora de saber tratar los datos a los que nos enfrentamos es el preprocesamiento que se les aplica ya que podemos tener datos incompletos, con ruido o inconsistentes. Con el preprocesamiento queremos generar datos de calidad que nos permitan obtener buenos modelos usando los algoritmos especificados.

Una parte del preprocesamiento consiste en la limpieza de los datos, esto es, la resolución de inconsistencias, imputar valores perdidos, suavizar el ruido de datos, detección y tratamiento de valores anómalos, etc. En los siguientes apartados vamos a explicar los diferentes procesos que se han llevado a cabo en los datos para generar datos con mayor calidad respecto a los iniciales

3.1. Transformaciones de los datos

Dentro de las posibles transformaciones de los datos destaca la normalización. Hay varias formas de normalizar los datos, pero hemos optado por usar como método de normalización la normalización zero-mean o z-score, y para ello usamos la función `preProcess` del paquete `caret`. Dado que del conjunto test no disponemos de ninguna información, la normalización de cada atributo del test se realiza en función a la media y desviación estándar del correspondiente atributo del conjunto train.

También se ha llevado a cabo el balanceo de las clases ya que una de ellas tiene menos datos que la otra. Para ello, se han empleado técnicas de oversampling, undersampling y Smote, llevándose a cabo a través de la función `ubBalance` del paquete `unbalanced`.

Por último, para intentar corregir la asimetría o posible ruido en los datos, además de permitir introducir más información sobre el conjunto de datos, se han utilizado las siguientes transformaciones sobre las variables.

- Logarítmica. Se aplica la función logaritmo directamente sobre las variables X2, X3, X4, X7, X9, X15, X16, X17, X18, X24, X28, X29, X31, X33 y X44. Además, esta función se aplica sobre la traslación de otras variables: X23+100, X35+0.1 y X48+25.
- Cuadrática. Se aplica una transformación cuadrática sobre las variables X8, X13 y X25.
- Raíz cuadrada. Se aplica la raíz cuadrada sobre la variable X45.
- Raíz cúbica. Se aplica la raíz cúbica sobre las variables X26, X27, X40, X47 y X49.
- Los valores negativos de las variables X6, X23, X35 y X45 son sustituidos por el valor 0.

3.2. Valores perdidos

Para imputar los valores perdidos que hay en los datos hemos usado las siguientes técnicas:

- Eliminar las tuplas que tengan valores perdidos.
- Rellenar los valores perdidos usando la media de los datos de la columna.
- Rellenar los valores perdidos usando la mediana de los datos de la columna.
- Rellenar los valores perdidos usando el algoritmo K-NN, de forma que se tienen en cuenta los K vecinos más cercano a dicha tupla. Para ello usamos la función `knnImputation` del paquete `DMwR`.

- Rellenar los valores perdidos usando el algoritmo Random Forest. Se usa la función `rfImpute` del paquete `randomForest`.
- Rellenar los valores perdidos usando `mice`: está basado en la unión de modelos y en la especificación de condiciones totales. Lo primero permite crear una distribución multivariante para los datos perdidos y permite ver cómo se haría la imputación de éstos basándose en sus distribuciones condicionales. Por otra parte, la especificación de las condiciones totales especifica el modelo de imputación multivariante variable a variable, basándose en un conjunto de densidad condicionales, una por cada variable incompleta.
- Rellenar los valores perdidos usando `Amelia`: permite realizar imputaciones múltiples de manera que reduce el sesgo en los datos. Para ello, toma m muestras meidante bootstrap y aplica el algoritmo EMB (Expectation-Maximization Boostrapping) de forma que se calculan m medias y varianzas y cada una de éstas se usa para generar los m conjuntos sin valores perdidos que se quieran obtener del inicial usando regresión.
- Rellenar los valores perdidos por clases: en este caso nos quedamos con las instancias cuya clase es 0 y 1 por separado y entonces aplicamos cualquier método de imputación de valores perdidos, como puede ser el `knnImputation` comentado anteriormente.

3.3. Filtros de Ruido

La detección de ruido y su filtrado es esencial para algunos algoritmos. Por eso uno de los pasos fundamentales en nuestro preprocesado ha sido el filtrado de ruido. Hemos probado tres filtros de ruido de clase para eliminar ejemplos mal clasificados. Los filtros que se han probado son: Ensemble Filter, Cross-Validated Committes Filter y Iterative-Partitioning Filter disponibles en el paquete `NoiseFiltersR` como `EF`, `CVCF` y `IPF`, respectivamente.

Las técnicas mencionadas se caracterizan por:

- EF: Se usan diferentes algoritmos de aprendizaje como clasificadores en varios subconjuntos del conjunto training y, para cada algoritmo, se aplica validación cruzada k-fold para etiquetar cada ejemplo como correcto o incorrecto (si la predicción coincide con la etiqueta del ejemplo o no).
- CVCF: Se diferencia del anterior en que se utiliza el mismo algoritmo de aprendizaje para crear clasificadores en varios subconjuntos del conjunto training y cada clasificador construido con la validación cruzada k-fold se usa para etiquetar todos los ejemplos del conjunto training como correctos o incorrectos.
- IPF: Elimina ruido en múltiples iteraciones usando CVCF hasta que se alcanza un criterio de parada.

Para determinar qué casos eliminar del conjunto de entrenamiento tras haber sido clasificados utilizan un esquema de votación, que puede ser de dos tipos: mayoría (elimina una instancia si está mal clasificada para la mayoría de los clasificadores) y consenso (elimina una instancia si está mal clasificada para todos los clasificadores). Las funciones usadas disponen de un parámetro `consensus` que será `TRUE` si el esquema por votación es por consenso y `FALSE` si es por mayoría.

3.4. Detección y Tratamiento de Outliers

Los outliers son datos con características considerablemente distintas a las del resto de datos del conjunto. Esto puede ocurrir tanto si una característica presenta un valor extremo, como si presenta una combinación de valores *extraña* en varias características.

Además de los outliers extremos encontrados durante el análisis de la base de datos, pueden existir otros. Se han intentado detectar outliers univariantes, y para ello se ha utilizado la técnica IQR para detección de outliers. El IQR se define como rango intercuartílico y viene dado por la distancia entre el tercer cuartil y el primero. Utilizando esta medida, un dato con valor **q** será clasificado como outlier normal si se da alguno de los siguientes casos:

- $q < Q_1 - 1{,}5$ IQR
- $q > Q_3 - 1{,}5$ IQR

Del mismo modo, se considerará que un dato es outlier es extremo si verifica uno de los dos siguientes casos:

- $q < Q_1 - 3$ IQR
- $q > Q_3 - 3$ IQR

Una vez que los outliers han sido identificados, es posible eliminarlos o imputar sus valores. En los casos en que se ha llevado a cabo su imputación, se ha hecho reutilizando las técnicas de imputación de valores perdidos.

3.5. Selección de características

La selección de características es un proceso que nos permite reducir el tamaño de nuestro conjunto de datos, lo cual permite que el proceso de aprendizaje sea más rápido y también permite aumentar la generalidad del modelo que se obtiene tras el aprendizaje. Además, es importante saber qué variables son interesantes para nuestro modelo y podemos eliminar las que sean irrelevantes. Además, hay algunos algoritmos que son sensibles a la presencia de variables no significativas, por lo que seleccionar las características importantes es posible que haga que mejore su rendimiento.

Nos hemos centrado en dos aproximaciones principalmente: filter, que permite obtener un orden de importancia de los atributos de acuerdo a alguna medida estadística, y embedded, que permite determinar la importancia de los atributos durante el proceso de aprendizaje de un modelo.

Para ello, hemos usado el paquete `FSelector`. Dentro de él se especifican varias funciones para realizar la selección de características. De entre todas ellas, hemos usado la función `chi.squared`, `linear.correlation` y `rank.correlation` como ejemplos de aproximación filter, y `random.forest.importance` como aproximación embedded. Comentar que también se usó el esquema de valoración con aprendizaje de random forest que proporciona el paquete `caret`.

También, como alternativa a la selección de características se han utilizado otras técnicas de reducción de dimensionalidad, como por ejemplo, el análisis de componentes principales (`PCA`), que permite obtener transformaciones lineales de los datos sobre espacios de menor dimensión.

3.6. Discretización

Con esta técnica tratamos de dividir el intervalo de posibles valores en un conjunto de fragmentos y asignar una etiqueta a cada uno de ellos, de forma que se sustituyen en el conjunto de datos los valores originales por las etiquetas asignadas a los intervalos. Este proceso se puede llevar a cabo de diferentes formas. Para

ello, se han usado varios algoritmos que son los que se comentan a continuación.

Los tres primeros se usaron con la función `disc.Topdown` del paquete `discretization` mientras que el último algoritmo que se aplicó de discretización se uso la función `discretizeDF.supervised` del paquete `arulesCBA`.

- Ameva: usa una medida basada en el test chi-cuadrado para la discretización óptima con el mínimo numero de intervalos y con la mínima pérdida de dependencia con respecto a la variable clase.
- CAIM (Class-Attribute Interdependence Maximization): mide la dependencia entre la variable clase y la discretización a emplear para el atributo.
- CACC (Class-Attribute Contingency Coefficient): calcula la tabla de contingencias entre la variable clase y la variable discretizada.
- MDLP (Minimum Description Length Principle): es un algoritmo que nos permite realizar una discretización para atributos continuos. Este método intenta maximizar la ganancia de información y considera todas las clases de los datos como clases distintas. Este algoritmo mide la ganancia de información de un punto de ruptura comparando los valores de entropía, comparando la entropía del intervalo inicial con la suma ponderada de los dos intervalos resultantes, donde la entropía se calcula en base a la proporción de ejemplos que hay en ese subconjunto de cada una de las clases presentes en los datos.

4. K-NN

En esta sección se muestran los resultados obtenidos en la competición de clasificación con el clasificador de vecinos cercanos (K-NN).

El clasificador de vecinos cercanos es una técnica de clasificación que, para clasificar nuevos datos, utiliza las clases de los datos más cercanos en el conjunto entrenamiento, fijada una distancia sobre dichos datos. Es un clasificador denominado *perezoso*, pues los cómputos se aplazan hasta el momento de la predicción, y no realiza ninguna hipótesis sobre las fronteras de decisión, pudiendo adaptarse a conjuntos con fronteras muy poco lineales.

Una ventaja del K-NN sobre nuestro conjunto de datos es que todos los atributos que presenta son reales. De esta forma podemos utilizar la distancia euclídea para obtener los vecinos. Uno de los preprocesamientos fundamentales para el K-NN consiste en normalizar los datos, pues diferentes rangos en cada atributo pueden condicionar en gran medida la distancia. A lo largo de la competición se ha utilizado el Z-Score para normalizar los datos. Los outliers extremos detectados durante el análisis han sido eliminados, ya que, aunque no afectan directamente al K-NN (esos outliers no van a ser nunca vecinos de datos normales), sí afecta mucho al Z-Score, condicionando mucho a la media y desviación típica. Simplemente estas medidas han permitido superar el 90 % de acierto en la mayoría de los intentos realizados.

Para la imputación de valores perdidos se ha utilizado de nuevo un K-NN, seleccionando como número de vecinos 10. La imputación de vecinos ha conseguido mejorar en torno a un 0.6 % el resultado frente a la solución más simple de tratamiento de valores perdidos consistente en eliminar los ejemplos con este tipo de valores. También ha sido de gran utilidad el uso de filtros de ruido. Se han probado los filtros EF, CVCF e IPF, utilizando tanto los criterios de consenso como de mayoría, siendo el filtro CVCF por mayoría el que ha permitido obtener mejores resultados en clasificación. Puesto que CVCF puede trabajar con valores perdidos, al utilizar internamente el clasificador C4.5, también se ha probado a intercambiar el orden de aplicación entre imputación y filtro de ruido. La aplicación del filtro de ruido en primer lugar consiguió nuevas mejoras en los resultados.

Otros tratamientos realizados sobre los datos se han basado en selección de instancias, reducción de dimensionalidad y selección y transformación de características. Las técnicas de selección de instancias han consistido principalmente en eliminar valores anómalos (además de los outliers extremos) que puedan perturbar la normalización. En general, esta selección de instancias no ha producido mejoras. Para reducción de dimensionalidad se ha utilizado el análisis de componentes principales (PCA), para distintos umbrales de varianza. En general tampoco se han conseguido mejoras con esta técnica.

Para selección de características se ha utilizado, por una parte, el algoritmo RELIEF (disponible en `FSelector::relief`). Este algoritmo permite aprender pesos para cada característica. Dichos pesos se calculan de forma iterativa según las distancias (atributo a atributo) de cada ejemplo a sus vecinos más cercanos de igual y distinta clase. Estos pesos, además de para seleccionar atributos, pueden utilizarse para ponderar cada componente de la distancia, de forma que los atributos con más peso influyan más en la distancia. A pesar de esto, no se han conseguido mejoras con esta técnica.

Por otra parte, también se ha utilizado la selección de características mediante el test chi cuadrado. Esta técnica, junto con la adición de las transformaciones en las características (logaritmos, raíces y cuadrados) explicadas en la sección de análisis, han conseguido mejorar de forma significativa los resultados.

Por último, también se han evaluado diferentes parámetros del clasificador K-NN para intentar aprender un modelo óptimo. En general, estos parámetros se han estimado mediante validación cruzada o bootstrap, usando el `trainControl` de `caret`. Se han considerado tres de los parámetros que mayor influencia pueden tener en la clasificación por vecinos cercanos, los cuales son:

- **Número de vecinos.** Este parámetro, normalmente llamado `K`, dando así nombre al K-NN, es fundamental a la hora de clasificar un nuevo ejemplo. Si el número de vecinos es bajo, el modelo puede ajustarse demasiado al conjunto de entrenamiento. Si es muy alto, el modelo puede perder capacidad de aprendizaje. Los mejores valores de `K` obtenidos han estado normalmente entre `15` y `25`, siendo `K=17` el que mejores resultados ha proporcionado en las soluciones finales.
- **Kernels.** Los kernels son funciones que proporcionan pesos a la distancia de un dato a sus vecinos. El kernel más común es el kernel rectangular, que asigna a todas las distancias el mismo peso, independientemente de su valor. Sin embargo, al asignar mayor peso a los datos más cercanos, podemos mejorar los resultados de la clasificación. En la clasificación con kernels, la clase de un dato es aquella que tenga mayor suma de pesos. Los kernels que mejores resultados han proporcionado han sido el triangular, dado por la función $k(d) = 1 - |d|$, y el kernel de *Epanechnikov*, dado por $k(d) = 3(1 - d^2)/4$.
- **Distancias.** Por defecto, los modelos de K-NN utilizan la distancia euclídea para medir la cercanía a los vecinos. Sin embargo, en ocasiones, la distancia euclídea puede no adaptarse bien a los datos, por lo que la búsqueda de una distancia apropiada puede mejorar los resultados de la clasificación. Toda una familia de distancias, denominadas *distancias de Mahalanobis*, pueden parametrizarse mediante matrices semidefinidas positivas M, de forma que $d(x, y) = (x - y)^T M(x - y)$. Esta distancia es equivalente a transformar los datos mediante una aplicación lineal L que verifique que $M = L^T L$, y medir la distancia euclidea sobre los datos transformados. Existen muchos métodos de *aprendizaje de distancias* disponibles para estimar una matriz M o L que se adapte bien a los datos. Dos de ellos son DMLMJ (*distance metric learning through the maximization of the Jeffrey divergence)* y NCA (*neighborhood component analysis*). El primero busca aprender una distancia que maximice la separación entre distribuciones asociadas a datos similares y no similares. El segundo busca, mediante gradiente ascendente, una distancia que maximice el valor esperado de acierto de un clasificador 1-NN, definiendo para ello una probabilidad que decrece con las distancias entre los datos. Se han probado ambos algoritmos, disponibles en la biblioteca `rDML`, (no se encuentra en CRAN, pero puede instalarse desde GitHub[1]), sin conseguir mejoras. Sin embargo, también se ha modificado el código de NCA para

[1]Para instalar `rDML` se puede utilizar la orden `devtools::install_github("jlsuarezdiaz/rDML")`

que la esperanza que se maximice dependa solo de un subconjunto de las instancias que sea capaz de explicar la mayor parte de los datos. Este conjunto de instancias se selecciona siguiendo la misma idea que en el algoritmo *Condensed Nearest Neighbors*, pero usando como criterio de selección umbrales de probabilidad en lugar del acierto o fallo del 1-NN. Este algoritmo también se ha añadido a la biblioteca `rDML`, con el nombre CNCA (*condensed neighborhood component analysis*) y ha conseguido mejorar a la anterior mejor solución de forma considerable.

Las Tablas 1, 2 y 3 muestran un esquema de las distintas soluciones subidas a Kaggle, junto a sus resultados. En todas ellas, aunque no se especifica, se aplica la eliminación de outliers extremos y la normalización. También, si no se especifica nada, el valor de `K` se ha obtenido mediante la selección de modelos de la función `train` de `caret`, el kernel por defecto utilizado es el rectangular, y la distancia por defecto la euclídea. Se han omitido algunas subidas debidas a soluciones erróneas o repeticiones.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
1	Eliminación de valores perdidos.	**0.90051**	**0.90045**
2	Imputación de valores perdidos con la media.	**0.90459**	**0.90199**
3	Imputación con K-NN.	**0.90612**	**0.90403**
4	Imputación con K-NN separando por clases.	0.90561	0.90352
5	Preprocesado de 3 con PCA al 95 %.	**0.90765**	0.89943
7	Imputación encadenada con K-NN: para cada variable, se aprende un modelo K-NN para imputar los valores perdidos. Los valores imputados se utilizan con la siguiente variable, para la que se aprende un nuevo K-NN.	0.90612	0.90352.
8	Imputación con K-NN y filtro EF por consenso.	**0.91071**	**0.90607**
9	Imputación con K-NN y filtro EF por mayoría.	0.90204	0.90148
10	Imputación con K-NN y filtro CVCF por consenso.	0.90816	0.90454
11	Imputación con K-NN y filtro CVCF por mayoría.	0.90918	**0.90862**
12	Imputación con K-NN y filtro IPF por consenso.	0.90714	0.90556
13	Imputación con K-NN y filtro IPF por mayoría	0.90714	0.90045
14	Preprocesado de 11 añadiendo la transformación de `YeoJohnson` para simetrizar las variables (disponible en la función `preProcess`).	0.90918	0.90862.
15	Preprocesado de 11, con `K=17` y el kernel de Epanechnikov.	**0.91479**	**0.91168**
16	Preprocesado de 11, con `K=17` y kernel triangular.	**0.91530**	**0.91220**
17	Preprocesado de 11, con `K=17` y voto mayoritario de los kernels rectangular, triangular, epanechnikov, inverso y gaussiano.	0.91020	0.90709
18	Preprocesado de 11, añadiendo pesos con RELIEF. `K=17` y kernel gaussiano (obtenido por validación cruzada).	0.91173	0.90760
19	Preprocesado de 11, añadiendo pesos con RELIEF. `K=17` y kernel coseno.	0.91428	0.90913

Cuadro 1: K-NN (I)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
20	Preprocesado de 11, añadiendo pesos con RELIEF. `K=17` y kernel triangular.	0.91377	0.90811
22	Preprocesado de 11, añadiendo PCA al 99 %. `K=17` y kernel gaussiano.	0.90510	0.89688
23	Preprocesado de 11, añadiendo PCA a 20 dimensiones (obtenidas por validación cruzada). `K=17` y kernel gaussiano.	0.91122	0.90658
24	Preprocesado de 11, eliminando los outliers más alejados (aparte de los extremos) en X1, X7, X15, X16, X17, X20, X21, X24, X26, X29, X33, X39, X43 y X45. `K=23` y kernel gaussiano.	0.91326	0.91066
25	Preprocesado de 24. `K=23` y kernel coseno.	0.91224	0.90760
26	Preprocesado de 11, eliminando los outliers más alejados (aparte de los extremos) en X1, X3, X7, X38 y X43.	0.91428	0.91015
27	Preprocesado de 11, cambiando los datos por las transformaciones indicadas en el análisis. `K=23` y kernel gaussiano.	0.91224	0.90658
28	Preprocesado de 11, cambiando los datos por las transformaciones indicadas en el análisis. `K=17` y kernel triangular.	0.91224	0.90862
29	Preprocesado de 11, añadiendo como nuevas variables las transformaciones indicadas en el análisis. `K=25` y kernel óptimo.	**0.91887**	0.91066
30	Preprocesado de 11, añadiendo como nuevas variables las transformaciones indicadas en el análisis. `K=17` y kernel triangular.	0.91887	0.91117
31	Preprocesado de 11, eliminando las variables cuyo peso obtenido con el test chi cuadrado es menor que 0.1, y añadiendo como nuevas variables las transformaciones indicadas en el análisis. `K=17` y kernel triangular.	**0.91989**	**0.91424**
32	Preprocesado de 31. `K=25` y kernel triangular.	0.91938	0.91373
33	Preprocesado de 31. `K=19` y kernel de Epanechnikov.	0.91989	**0.91475**
34	Preprocesado de 31, aprendiendo distancias con DMLMJ. `K=19` y kernel de Epanechnikov.	0.90153	0.89688
35	Preprocesado de 31, añadiendo la variable X34 y su transformación. `K=19` y kernel de Epanechnikov.	0.91887	0.91066

Cuadro 2: K-NN (II)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
36	Preprocesado de 31, sin modificar valores en X33. `K=19` y kernel de Epanechnikov.	**0.92142**	0.90964
38	Preprocesado de 31, eliminando las variables cuyo peso obtenido con el test chi cuadrado es menor que 0.2. `K=19` y kernel de Epanechnikov.	0.90816	0.90403
39	Preprocesado de 31, aplicando el filtro CVCF y añadiendo las transformaciones antes de la imputación K-NN. `K = 19` y kernel de Epanechnikov.	0.91785	**0.91781**
41	Preprocesado de 39. `K=17` y kernel triangular.	0.91785	0.91730
42	Preprocesado de 39. `K=23` y kernel gaussiano.	0.91224	0.91066
43	Preprocesado de 39. `K=17` y kernel de Epanechnikov.	0.91836	**0.91832**
44	Preprocesado de 39. Aprendizaje de distancias con NCA, `K=17` y kernel de Epanechnikov.	0.91224	0.90964
46	Preprocesado de 39. `K=15` y kernel de Epanechnikov.	0.91887	0.91628
47	Preprocesado de 39. `K=21` y kernel de Epanechnikov.	0.91938	0.91475
50	Preprocesado de 39, con la modificación implementada NCA (CNCA) para aprender distancias. `K=17` y kernel de Epanechnikov.	**0.92448**	**0.92036**

Cuadro 3: K-NN (III)

5. SVM

En esta sección se busca el mejor preprocesado posible para aplicar las Máquinas de Soporte Vectorial (SVM) sobre este problema de clasificación. Además, se mostrarán los resultados obtenidos en la competición de Kaggle para cada preprocesado.

Las SVM son algoritmos de aprendizaje supervisado que tratan de encontrar un hiperplano que separe los puntos de ambas clases. En caso de que el problema no sea lineal, es posible utilizar kernels para *trasladar* los datos a un espacio donde el hiperplano solución sea lineal. Una vez obtenida la solución, se vuelve a transformar al estado original.

Aunque al clasificador SVM no le afecta la normalización (escalado) de los datos, este factor sí puede ser relevante dependiendo del kernel utilizado. Por ello, en todos las secuencias de preprocesado llevadas a cabo se escalan los datos, tanto del conjunto de entrenamiento como del de prueba.

En las tablas 4,5, 6, 7 y 8 pueden verse las distintas subidas hechas a la competición de Kaggle. En ella, además del número de subida y la puntuación obtenida, tanto en privado como en público, se puede observar

una breve descripción del preprocesado llevado a cabo.

En primer lugar, en la tabla 4, se observan las primeras subidas llevadas a cabo. En ellas, trato de encontrar los métodos de imputación de valores perdidos y outliers que mejor funcionan para este problema. Puede observarse que el mejor resultado se obtiene en la subida 7, donde se utiliza k-NN para imputar los valores perdidos y se rellenan los valores de los outliers utilizando la media. El número de vecinos utilizado para aplicar el algoritmo k-NN ha sido 2, ya que no se obtenían mejoras significativas en el conjunto de entrenamiento aumentando este número.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
1	Escalar y eliminar los datos con valores perdidos	**0.85612**	**0.84992**
2	Escalar y rellenar los valores perdidos utilizando la media	**0.85612**	0.84992
3	Escalar, rellenar los valores perdidos con la media y eliminar los outliers	0.71926	0.71056
4	Escalar, rellenar los valores perdidos con la media y rellenar outliers utilizando knn	0.71926	0.71056
5	Escalar y rellenar valores perdidos con knn	0.85153	0.84839
6	Escalar, rellenar valores perdidos con knn y rellenar outliers con knn	0.81020	0.81929
7	Escalar, rellenar valores perdidos con knn y rellenar outliers con la media	**0.88928**	**0.87953**
8	Escalar, rellenar valores perdidos con knn, rellenar outliers con la media y eleminar datos pocos correlados con la variable de salida	0.87448	0.8749

Cuadro 4: SVM (I)

El siguiente paso es, partiendo del mejor preprocesado hasta el momento, evaluar distintas técnicas de balanceo para el conjunto de entrenamiento. En la tabla 5 pueden verse los resultados obtenidos en este proceso y, como se puede observar, no son buenos, aunque se mejora muy ligeramente utilizando la técnica de oversampling. Además, en las subidas 12 y 13 se ha utilizado random forest para rellenar los valores de los outliers, pero el resultado obtenido no mejora.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
9	Escalar, rellenar valores perdidos con knn, rellenar outliers con la media y balanceo conjunto de entrenamiento utilizando oversampling	**0.88979**	**0.88514**
10	Escalar, rellenar valores perdidos con knn, rellenar outliers con la media y balanceo conjunto de entrenamiento utilizando undersampling	0.87857	0.88463
11	Escalar, rellenar valores perdidos con knn, rellenar outliers con la media y balanceo conjunto de entrenamiento utilizando SMOTE	0.87908	0.88208
12	Escalar, rellenar valores perdidos con knn y rellenar outliers utlizando random forest	0.88826	0.88055
13	Escalar, rellenar valores perdidos con knn, rellenar outliers utlizando random forest y oversampling	0.87959	0.87544

Cuadro 5: SVM (II)

Llegados a este punto, se decide eliminar los outliers que han sido destacados en la sección de análisis del conjunto de datos, aquellos datos que contienen valores menores que -68.000. Como puede verse en la tabla 6 esto, junto con la imputación mediante k-nn de los valores perdidos, supone la primera mejora significativa hasta el momento.

Las siguientes subidas (15,16 y 17) consisten en intentar completar este preprocesado utilizando técnicas que ya han sido probadas sin la eliminación de estos outliers: imputación de outliers, técnicas de balanceo y eliminar variables poco relacionadas con la variable de salida.

En las subidas 19 y 20 se aplica la técnica de filtrado de ruido CVCF en primer lugar después de la imputación de valores perdidos con k-nn y, en segundo lugar, antes de la misma. Además, se aplican las transformaciones expuestas anteriormente y se decide etiquetar como clase negativa a los outliers con valores menores que -68.000 del conjunto test. Esta decisión se ha tomado porque, como se ha visto en la sección de análisis del conjunto de datos, la mayoría de estos datos en el conjunto de entrenamiento pertenecen a la clase negativa.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
14	Escalar, eliminar outliers (-68.000) e imputar valores perdidos con k-NN	**0.92704**	**0.92547**
15	Preprocesado 14 y utilizar k-NN para imputar el resto de outliers	0.89336	0.88208
16	Preprocesado 14 y balanceo del conjunto train con oversampling	0.90918	0.90964
17	Preprocesado 14 y eliminar variables poco relacionadas con la salida	0.92193	0.92496.
18	Preprocesado 14 y aplicar PCA tomando 30 componententes principales	0.92397	0.92036.
19	Preprocesado 14, filtro CVCF, transformaciones y los ejemplos del test con valores extremos (menores que -68000) se estiquetan con su clase mayoritaria, la negativa	0.92602	**0.92649**
20	Similar a la anterior, pero utilizando primero el filtro CVCF y después la imputación de valores perdidos con k-NN	0.92397	**0.92751**

Cuadro 6: SVM (III)

Hasta ahora, se ha utilizado el kernel radial en todas las ejecuciones. Llegados a este punto se decide probar diferentes kernels por si esta decisión no hubiera sido la mejor. Los kernels probados, además del radial, han sido el lineal, el polinomial y el sigmoidal. Los resultados obtenidos pueden verse en la tabla 7, donde se observa que el mejor resultado se obtiene utilizando el kernel radial.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
21	Preprocesado 20 y kernel polinomial	0.91428	0.91220
22	Preprocesado 20y kernel radial	0.92397	0.92751
23	Preprocesado 20 y kernel sigmoidal	0.81428	0.81827.
24	Preprocesado 20 y kernel lineal	0.91785	0.91066

Cuadro 7: SVM (IV)

Por último, se vuelve a aplicar técnicas de balanceo sobre el conjunto de entrenamiento tras el mejor preprocesado encontrado hasta el momento, obtenido en la subida 20.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
25	Preprocesado 20 con kernel radial y balanceo con SMOTE	**0.92857**	0.92547
26	Preprocesado 20 con kernel radial y balanceo con undersampling	0.91071	0.90352
27	Preprocesado 20 con kernel radial y balanceo con oversampling	0.91836	0.91883

Cuadro 8: SVM (V)

6. RIPPER

En esta sección se muestran los resultados obtenidos para la competición de clasificación usando el algoritmo basado en reglas *Repeated Incremental Pruning to Produce Error Reduction* (RIPPER). Se ha usado la función `JRip` de el paquete `RWeka`.

El objetivo de este algoritmo es cubrir el mayor número de ejemplos de la clase considerada positiva. Además, se aplica una poda para reducir el número de reglas.

Sus principales ventajas son su sencillez tanto en interpretación como en aplicación y su flexibilidad. A diferencia de otros algoritmos, no es necesario que los datos estén normalizados o sean de un tipo en concreto y puede trabajar con valores perdidos.

Principalmente podemos dividir las pruebas realizadas en tres bloques: tratamiento de outliers, valores perdidos y ruido. Por último, se han realizado algunas pruebas que no se encuentran dentro de ninguno de los bloques anteriores y se describen en el último punto:

- Tras una primera prueba de la aplicación del algoritmo sobre los datos en bruto se han eliminado los outliers extremos detectados durante el análisis y se obtiene cierta mejora en la puntuación pública pasando de un 88.5 % a un 89.1 %. Si bien, la puntuación privada no mejora.
- Se ha realizado una subida eliminando los valores perdidos simplemente y, como cabía esperar, no supone ningún cambio ya que por defecto se omiten al aplicar el algoritmo. En posteriores subidas se han probado diversas estrategias para imputarlos: usando la media, k-NN o k-NN por clases. La técnica que mejores resultados nos proporciona, tanto en la puntuación pública como la privada, es la imputación de valores perdidos con k-NN por clases que consigue superar la mejor puntuación hasta el momento pasando de un 89.1 % a un 89.7 % en la puntuación pública y de un 89.1 % a un 89.5 % en la privada.
- Se han aplicado los filtros de ruido descritos en la sección 3.3: EF, CVCF e IPF probando los esquemas de votación de mayoría y consenso. La aplicación de EF por mayoría supuso una mejora significativa de la puntuación tanto pública como privada: pasando de un 89.7 % a un 90.1 % en la pública y de un 89.5 % a un 90.2 % en la privada.
- También se han realizado las transformaciones de características (logaritmos, raíces y cuadrados) descritas en la sección de análisis pero no se han conseguido mejoras con ellas. Por último, se probó a alterar el orden de la imputación de valores perdidos y de filtros de ruido. Para ello se utilizó CVCF ya que también permite trabajar con valores perdidos y posteriormente se imputaron los valores perdidos usando k-NN por clases pero no se obtuvo mejora.

La combinación de eliminación de outliers, imputación de valores perdidos con k-NN por clases y filtro de ruido EF por mayoría es la que nos proporciona mejores resultados superando el 90 % de tasa de acierto tanto en la competición pública como en la privada como anticipábamos antes.

La tabla 9 muestra los resultados obtenidos en las distintas subidas a Kaggle así como una descripción esquemática del preprocesado realizado. En todas las subidas, excepto en la primera, se han eliminado los outliers extremos. Se han omitido algunas subidas por ser erróneas o repeticiones.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
1	Datos en crudo	0.89081	0.88514
2	Eliminación de outliers extremos	0.88469	**0.89076**
3	Eliminación de valores perdidos	0.88469	0.89076
4	Imputación de valores perdidos con la media	0.88979	0.88514
5	Imputación de valores perdidos con k-NN	0.87908	0.87953
8	Balanceo `ubUnder`	0.86224	0.85349
9	Imputación de valores perdidos con k-NN por clases	**0.89540**	**0.89688**
10	Imputación k-NN por clases y filtro EF por consenso	0.89336	0.88718
12	Imputación k-NN por clases y filtro CVCF por consenso	0.89030	0.88973
13	Imputación k-NN por clases y filtro CVCF por mayoría	**0.90051**	0.88922
14	Imputación k-NN por clases y filtro IPF por consenso	0.88520	0.88769
15	Imputación k-NN por clases y filtro IPF por mayoría	0.89489	0.88820
16	Imputación k-NN por clases y filtro EF por mayoría	**0.90255**	**0.90148**
17	Preprocesado 16 y transformaciones indicadas en el análisis (sin escalado)	0.89132	0.89076
18	Preprocesado 16 y transformaciones indicadas en el análisis (con escalado)	0.89795	0.88667
19	En primer lugar, filtro CVCF por mayoría en primer lugar y imputación k-NN por clases	0.89132	0.89025

Cuadro 9: RIPPER (I)

7. Árboles de clasificación

En esta sección vamos a mostrar los resultados obtenidos en la competición de árboles de clasificación.

Los árboles de clasificación intentan dividir el espacio de búsqueda en regiones más simples. Esto les otorga más simpleza que otros métodos de aprendizaje y son buenos debido a la interpretación que se puede obtener a partir de ellos. En el caso de los árboles de clasificación, se predice que una observación va a pertenecer a la clase de las observaciones de entrenamiento más común dentro de la región que le pertenezca.

Los problemas que pueden tener los árboles de clasificación están relacionados con el underfitting y overfitting y los valores perdidos. Para solucionar el segundo problema vamos a emplear técnicas de imputación de valores perdidos, mientras que para el primer problema vamos a realizar la poda del árbol.

A continuación, vamos a describir las diferentes técnicas que se han utilizado en la competición. Lo primero

comentar que se ha realizado un preprocesamiento basado en el Z-Score para normalizar los datos, aunque para el caso de árboles de decisión no se ven afectados por transformaciones monotónicas. Este preprocesamiento de normalización de datos es el que hemos usado en la competición.

Otro aspecto importante son los valores anómalos, que afectan a la media y varianza de cada uno de los atributos del problema y, por tanto, a la normalización. A pesar de que es escalado no influye en los árboles de decisión, los outliers sí que pueden tener un factor importante cuando se crea el árbol de decisión, lo que puede llevar a que el árbol se cree en base a éstos y eso puede llevar a un peor modelo aprendido y pueda, como consecuencia, haber problemas de sobreaprendizaje.

Otro aspecto importante del preprocesado es la imputación de valores perdidos y, sobre todo, para el caso de los árboles. Por ello, hemos probado varias formas de imputarlos: usando un algoritmo K-NN, a partir de `knnImputation`, usando la media y la mediana, la función `mice` del paquete `mice`, el paquete `amelia` y random forest a través de `rfImpute`.

El uso de filtros de ruido también ha sido importante en la competición ya que los mejores resultados obtenidos han sido usando alguno de los filtros de ruido presentados anteriormente junto con otras técnicas de preprocesamiento. En particular, se ha probado los filtros de ruido EF, IPF y CVCF, tanto con la opción de consenso como por mayoría. Dado que CVCF también permite trabajar con valores perdidos, en las últimas subidas de esta competición se probó a usar este filtro como primer paso del preprocesamiento, aunque no se obtuvo mejoras significativas.

También se probó a realizar selección de características a través de los métodos comentados anteriormente, como son `random.forest.importance`, `chi.squared`, entre otros, pero en general, no mejoraban el modelo obtenido. Sin embargo, una de las mejores subidas se obtuvo imputando valores perdidos usando K-NN, después usar el filtro de ruido IPF por mayoría, centrado y escalado y tomando los 10 atributos más importantes según `random forest importance`. La selección de instancias consistió en eliminar valores raros de algunas variables (incluyendo también los outliers extremos) y también se probó como reducción de dimensionalidad el análisis de componentes principales (PCA) para un conjunto de umbrales, pero se optó por subir los dos más prometedores y no se obtuvo mejora.

Como último experimento en el preprocesamiento también se añadieron las transformaciones en las características (logaritmos, raíces y cuadrados) que se han explicado en la sección de análisis y ayudaron a obtener el mejor resultado en la clasificación privada.

En esta competición se podía usar árboles de clasificación simples. Por ello, hemos decidido probar tres de ellos.

- **Ctree.** Permite crear un árbol de decisión usando condiciones de inferencia entre la variable de salida y los predictores, es decir, mide el grado de asociación entre Y y X_j, $j = 1, \ldots, N$, siendo N el número total de predictores. Tras ello, selecciona la variable con mejor asociación y se realiza el particionamiento. Uno de los parámetros que hemos usado para controlar la poda en estos árboles es `mincriterion`, que representa el valor del test estadístico (o bien 1- p-valor) que se debe superar para que se realice el particionamiento.
- **Rpart.** Permite crear un árbol usando un proceso de partición recursiva de forma que se toma una variable y se separa en dos grupos de forma que esta variable sea la mejora (según un criterio) y este proceso se realiza de forma recursiva hasta llegar a un tamaño mínimo. La segunda parte de este

algoritmo consiste en podar el árbol usando validación cruzada. Un parámetro importante de este tipo de árbol es `cp`, el parámetro de complejidad, que nos permite no desarrollar nodos del árbol que no sean de calidad.

- **J48.** Este tipo de árbol divide en rangos en base a los valores encontrados en el conjunto de entrenamiento. Además, permite realizar la clasificación mediante el árbol que genera o bien a través de las reglas generadas a partir de él. El criterio de selección para la división es el ratio de ganancia de información, que tiene en cuenta la cardinalidad de la división. Este tipo de árbol no permite combatir el sobreaprendizaje usando la pre-poda y post-poda. En la competición hemos usado dos parámetros para estudiar el comportamiento del modelo que se genera y son `M`, que nos indica el número mínimo de instancias por hoja y `C`, que fija el umbral de confianza para la poda.

Las Tablas 10, 11,12,13,14,15,16 y 17 muestran un esquema de las distintas soluciones subidas a Kaggle, junto a los resultados obtenidos en ellas.

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
1	Datos en crudo y se usó Rpart como árbol de clasificación	**0.86020**	**0.85502**
2	Datos en crudo con validación cruzada (CV) usando Rpart	**0.86479**	**0.86727**
3	Imputación de valores perdidos usando un 2-NN y centrado y escalado de los datos usando Rpart con CV como algoritmo de árboles	**0.86989**	**0.86932**
4	Imputación de valores perdidos usando la mediana y centrado y escalado de los datos usando Rpart con CV como algoritmo de árboles	0.86989	0.86932
5	Imputación de valores perdidos usando mice, centrado y escalado de los datos y usando Rpart con CV y se realiza poda usando el parámetro `cp` como algoritmo de árboles	0.86989	0.86932
6	Imputación de valores perdidos usando la mediana, centrado y escalado y usando Ctree con CV como algoritmo de árboles	0.86989	0.86932
7	Imputación de valores perdidos usando la mediana, centrado y escalado y usando J48 con CV como algoritmo de árboles	0.86989	0.86932
8	Imputación de valores perdidos usando Random Forest, centrado y escalado y uso de Rpart con CV como algoritmo de aprendizaje	0.86479	0.85962

Cuadro 10: Árboles de clasificación (I)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
9	Imputación de valores perdidos usando la mediana, centrado y escalado, selección de características usando `linear.correlation` tomando los 30 atributos más importantes y usando Rpart como algoritmo de aprendizaje	0.71326	0.71056
10	Imputación de valores perdidos usando la mediana, centrado y escalado, selección de características usando `linear.correlation` tomando los 3 atributos más importantes, usando Rpart como algoritmo de aprendizaje y poda usando `cp`	0.71326	0.71056
11	Imputación de valores perdidos usando la mediana, centrado y escalado, selección de características usando `rank.correlation` tomando los 7 atributos más importantes, usando Rpart como algoritmo de aprendizaje y poda usando `cp`	0.71326	0.71056
12	Imputación de valores perdidos usando 1-NN, filtro de ruido IPF por mayoría, centrado y escalado y `random.forest.importance` con 10 atributos usando Rpart con CV	0.85204	0.85145
13	Imputación de valores perdidos usando 1-NN, filtro de ruido IPF por mayoría, selección de características usando `random.forest.importance` con 10 variables y Ctree con CV	**0.88367**	**0.88361**

Cuadro 11: Árboles de clasificación (II)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
14	Imputación de valores perdidos usando 1-NN, filtro de ruido IPF por mayoría, selección de características usando `random.forest.importance` con 20 variables y Ctree con CV	**0.88571**	0.87697
15	Imputación de valores perdidos usando Random Forest, quitamos los outliers, discretizamos usando el método CAIM, Ranking Learning Forest, centrado y escalado y Ctree con CV	0.49081	0.48647
16	Imputación de valores perdidos usando Random Forest, quitamos los outliers, usamos `chi.squared` como selección de características con 10 variables, centrado y escalado y Ctree con CV	0.88112	0.87697
17	Imputación de valores perdidos usando Random Forest, quitamos los outliers, usamos `chi.squared` como selección de características con 20 variables, centrado y escalado y Ctree con CV	0.87244	0.86013
18	Imputación de valores perdidos usando Random Forest, quitamos los outliers, usamos `random.forest.importance` como selección de características con 10 variables, centrado y escalado y Ctree con CV	0.87959	0.87646

Cuadro 12: Árboles de clasificación (III)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
19	Imputación de valores perdidos usando Random Forest, quitamos los outliers, usamos `random.forest.importance` como selección de características con 20 variables, centrado y escalado y Ctree con CV	0.86581	0.86727
20	Imputación de valores perdidos usando Random Forest, quitamos los outliers, discretizamos usando el método CACC, Ranking Learning Forest, centrado y escalado y Ctree con CV	0.82142	0.84992
21	Imputación de valores perdidos usando Random Forest, quitamos los outliers, discretizamos usando el método AMEVA, Ranking Learning Forest, centrado y escalado y Ctree con CV	0.74081	0.74476
22	Imputación de valores perdidos usando Random Forest, centrado y escalado y usamos `random.forest.importance` como selección de características con 10 variables junto con Ctree	0.87244	0.86013
23	Imputación de valores perdidos usando 1-NN, filtro de ruido IPF por mayoría, selección de características usando `random.forest.importance` con 10 variables y Ctree cambiando el parámetro `mincriterion` para la poda	0.86938	0.86625

Cuadro 13: Árboles de clasificación (IV)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
24	Imputación de valores perdidos usando 1-NN, filtro de ruido IPF por mayoría, selección de características usando `random.forest.importance` con 10 variables y Ctree cambiando el parámetro `mincriterion` para la poda	0.86581	0.86727
25	Imputamos los valores perdidos con 1-NN, usamos PCA para seleccionar las variables principales, balanceamos el conjunto de datos para igualar la proporción de etiquetas y usamos Ctree con CV	0.84591	0.85860
26	Imputamos los valores perdidos con 1-NN, usamos PCA para seleccionar las variables principales (menos que en el caso anterior), balanceamos el conjunto de datos para igualar la proporción de etiquetas y usamos Ctree con CV	0.84591	0.85860
27	Quitamos los valores perdidos, usamos el filtro de ruido IPF por mayoría, selección de características con `random.forest.importance` con 10 variables principales y centrado y escalado y usando Rpart	0.71173	0.70495
28	Similar al anterior, pero rellenamos los valores perdidos usando la mediana	0.86530	0.86472

Cuadro 14: Árboles de clasificación (V)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
29	Usamos Amelia para rellenar los valores perdidos, IPF por mayoría para el ruido, `random.forest.importance` para seleccionar características con 10 variables, centrado y escalado y Rpart con CV	0.87959	0.87544
30	Imputamos los valores perdidos usando 1-NN, usamos un filtro de ruido IPF por mayoría, `random.forest.importance` para seleccionar características con 10 variables, centrado y escalado, balanceamos el conjunto de etiquetas y quitamos las variables altamente correladas y usamos Rpart con CV	0.86428	0.86370
31	Quitamos únicamente los valores extremos y usamos Ctree	0.86836	0.86983
32	Quitamos los outliers extremos, imputamos los valores perdidos con 10-NN, filtro de ruido CVCF por mayoría, centrado y escalado y Ctree con grid para poda con mincriterion	0.86683	0.86983
33	Similar a la anterior, pero se usa filtro de ruido EF por consenso	0.87091	0.88259
34	Similar a la anterior, pero además del filtro de ruido EF por consenso, añadimos selección de características con `random.forest.importance` con 10 variables importantes	0.87551	0.86574

Cuadro 15: Árboles de clasificación (VI)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
35	Similar a la subida 34, pero usamos ahora el filtro de ruido EF por mayoría	0.87346	0.88157
36	Similar a la subida 33, pero con filtro de ruido CVCF por consenso	0.87959	0.87544
37	Similar a la subida 34, pero hemos añadido las transformaciones a las variables respecto a la variable de salida	0.87653	0.86319
38	Similar a la subida 37, pero con Rpart y CV y realizando poda del árbol	0.86326	0.86421
39	Similar a la subida 34, pero usando EF por consenso	0.87653	0.87034
40	Quitamos los valores anómalos extremos, imputación de valores perdidos con 10-NN, usamos filtro de ruido IPF por mayoría y Rpart con grid para obtener modelo con menor cp usando CV	0.87244	0.87953
41	Quitamos los outliers extremos, imputación de valores perdidos por clases y usamos Ctree	0.87755	0.86983
42	Quitamos los outliers extremos, imputación de valores perdidos con Amelia, EF por consenso, balanceado de clases con Smote y Ctree con CV	0.86326	0.85451
43	Quitamos los valores extremos raros, 10-NN para valores perdidos, EF por consenso, centrado y escalado y usamos J48	0.88367	0.87442

Cuadro 16: Árboles de clasificación (VII)

Número	Preprocesado	Puntuación (privado)	Puntuación (público)
44	Similar a la anterior, pero con CVCF por mayoría y con `random.forest.importance` usando las 25 variables más importantes	0.86428	0.86881
45	Primero realizamos CVCF por mayoría, después quitamos outliers extremos, realizamos las transformaciones en los atributos respecto a la variable de salida, centrado y escalado y usamos J48	0.86377	0.86830
46	Quitamos todos los outliers extremos, 10-NN para valores perdidos, EF por consenso y Ctree	0.87500	0.88259
47	Similar a la anterior, pero probamos ajustar el mejor valor para una malla de mincriterion en Ctree para la poda	0.87908	0.87646
48	Similar a la subida 45, pero también discretizamos por columnas usando `mdlp`	0.86734	0.86370
49	Similar a la anterior, pero eliminas variables altamente correladas, tanto positiva como negativamente	0.86734	0.86370
50	Similar a la subida 48, pero usando Ctree en vez de J48	0.88520	0.87748
51	A la subida 49, le añadimos selección de características con `chi.squared` de 35 variables	0.85561	0.86932
52	Similar a la anterior, pero usando Ctree en vez de J48	0.87704	0.87289

Cuadro 17: Árboles de clasificación (VIII)