Práctica 1: Clasificación

Grupo 5

Curso 2025/2026

Esta práctica debe entregarse antes del viernes 28 de noviembre de 2025 en formato pdf, incluyendo el código R utilizado, las correspondientes salidas y los comentarios (o interpretaciones de los resultados) pertinentes (para ello se recomienda emplear RMarkdown, a partir de un fichero *.Rmd* o un fichero *.R* mediante spin).

En esta práctica se empleará una modificación del conjunto de datos `spotify`, que contiene una lista exhaustiva de las canciones más famosas reproducidas en 2023 que aparecen en Spotify. Lista completa de características:

Variable	Descripción
track_name	Nombre de la canción
artist(s)_name	Nombre del artista o artistas de la canción
artist_count	Número de artistas que participan en la canción
streams	Número total de reproducciones en Spotify
released_year	Año en que se lanzó la canción
released_month	Mes en que se lanzó la canción
released_day	Día del mes en que se lanzó la canción
in_spotify_charts	Posición de la canción en las listas de Spotify
in_apple_charts	Posición de la canción en las listas de Apple Music
in_deezer_charts	Posición de la canción en las listas de Deezer
bpm	Pulsaciones por minuto, medida del tempo de la canción
key	Tono o tonalidad de la canción
mode	Modo de la canción (mayor o menor)
danceability_	Porcentaje que indica cuán adecuada es la canción para bailar
valence_	Nivel de positividad del contenido musical de la canción
energy_	Nivel percibido de energía de la canción
acousticness_	Cantidad de sonido acústico presente en la canción
instrumentalness_	Cantidad de contenido instrumental en la canción
liveness_	Presencia de elementos de interpretación en vivo
speechiness_	Cantidad de palabras habladas presentes en la canción
hit	Factor que indica si la canción ha sido un éxito (`"Yes"`) o no (`"No"`)

Se considerará como respuesta la variable `hit`, factor que indica si una canción es un *éxito*, `hit=Yes` o no `hit=No`. Como predictores se pueden utilizar el resto de variables del conjunto de datos. Más información relacionada con el conjunto de datos, visitar https://www.kaggle.com/datasets/nelgiriyewithana/top-spotify-songs-2023.

Cada grupo de práctica dispone de una modificación del conjunto de datos personalizada, que contieneuna lista reducida con de variables y observaciones, en el fichero `spotify2023_XX.RData` donde `XX` indica el número del grupo. Se debe establecer la semilla igual al número de grupo multiplicado por 10 mediante la función `set.seed()`:

```
grupo <- 5
load(paste0("spotify2023_", grupo, ".RData"))
semilla <- as.numeric(grupo) * 10
set.seed(semilla)
```

También se recomienda establecer la semilla antes de ajustar cada modelo. Se considerarán el 80% de las observaciones como muestra de aprendizaje y el 20% restante como muestra de test.

Ejercicio 1

Obtener un árbol de decisión que permita clasificar si una canción (observación) se considera un éxito (`hit=Yes`) o no (`hit=No`).

a. Seleccionar el parámetro de complejidad de forma automática siguiendo el criterio de un error estándar de Breiman et al. (1984).

b. Representar, interpretar el árbol resultante y comentar la importancia de las variables.

c. Evaluar la precisión, de las predicciones usando la métrica (o métricas) que consideréis más oportuna. Proporcionar las estimaciones de la probabilidad de `"Yes"`, en la muestra de test.

Análisis Exploratorio Inicial

Separamos el data frame completo en un conjunto de train y test, y a continuación examinamos los datos:

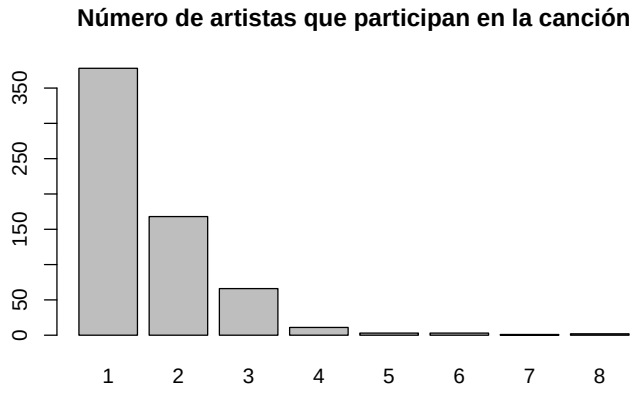
```
set.seed(semilla)
nobs <- nrow(df)
itrain <- sample(nobs, 0.8 * nobs)
train <- df[itrain, ]
test <- df[-itrain, ]

str(train)
```

```
## 'data.frame':    632 obs. of  16 variables:
##  $ artist_count      : Factor w/ 8 levels "1","2","3","4",..: 1 1 1 2 1 1 1 3 1 2 ...
##  $ released_year     : int  2023 2022 2022 2023 2019 2023 2021 2022 2023 2022 ...
##  $ released_month    : Ord.factor w/ 12 levels "Jan"<"Feb"<"Mar"<..: 5 3 10 7 10 1 6 11 2 8 ...
##  $ released_day      : int  11 19 21 14 4 27 11 4 17 25 ...
##  $ in_spotify_charts : int  29 0 0 147 0 0 24 0 2 13 ...
##  $ in_deezer_charts  : int  11 0 0 10 0 0 3 0 0 3 ...
##  $ bpm               : int  144 170 108 125 80 107 92 122 92 93 ...
##  $ mode              : Factor w/ 2 levels "Major","Minor": 1 1 1 1 2 2 1 2 1 2 ...
##  $ danceability_.    : int  75 56 64 80 56 66 81 75 57 63 ...
##  $ valence_.         : int  35 52 4 89 19 47 39 45 68 34 ...
##  $ energy_.          : int  48 64 40 83 46 40 60 63 76 86 ...
##  $ acousticness_.    : int  84 11 6 31 92 72 31 6 7 26 ...
```

```
##  $ instrumentalness_.: int  0 0 0 0 72 0 0 0 0 0 ...
##  $ liveness_.        : int  10 45 10 8 11 11 7 35 33 21 ...
##  $ speechiness_.     : int  12 7 6 4 3 3 3 12 3 39 ...
##  $ hit               : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 1 1 1 1 ...
```

artist_count

```
plot(train$artist_count, main="Número de artistas que participan en la canción")
```




```
table(train$hit , train$artist_count)
```

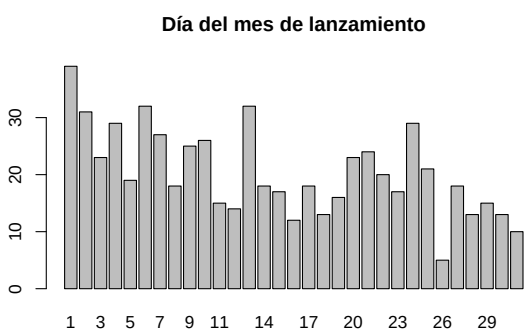
```
##
##         1   2   3   4   5   6   7   8
##   No  280 114  52   9   3   3   1   2
##   Yes  98  54  14   2   0   0   0   0
```

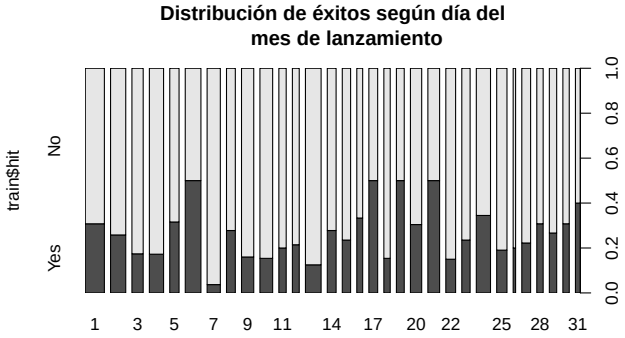
La variable `artist_count` tiene clases muy poco representadas. Además, en estas clases poco representadas no hay éxitos.

released_day

Convertimos los días a tipo factor:

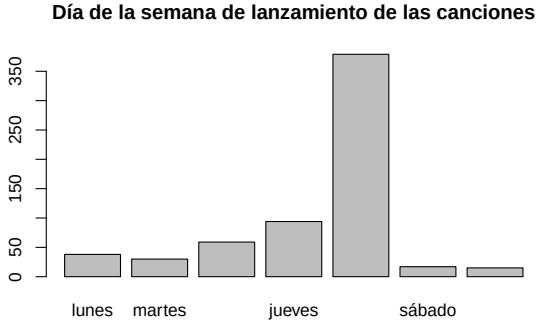
```
train$released_day <- factor(train$released_day, levels = 1:31)

par(mfrow=c(1,2))
plot(train$released_day, main="Día del mes de lanzamiento")
plot(train$hit ~ train$released_day, main="Distribución de éxitos según día del
     mes de lanzamiento", xlab=" ")
```

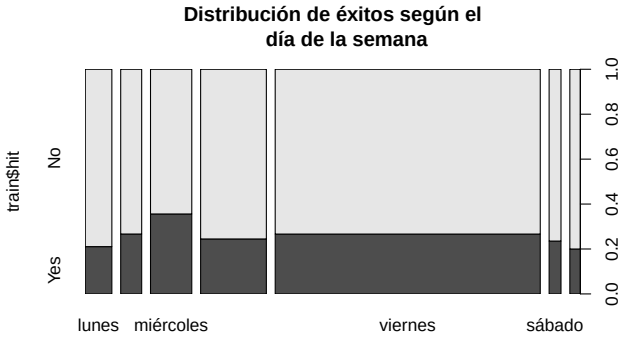




Utilizando la librería `lubridate` podemos obtener el día de la semana, lo cual resulta más informativo para el modelo, ya que utilizar `released_day` como categoría para cada día del mes carece de sentido práctico. Como alternativa, si se quisiera capturar efectos intramensuales, podrían agruparse los días en inicios, mediados o finales de mes.

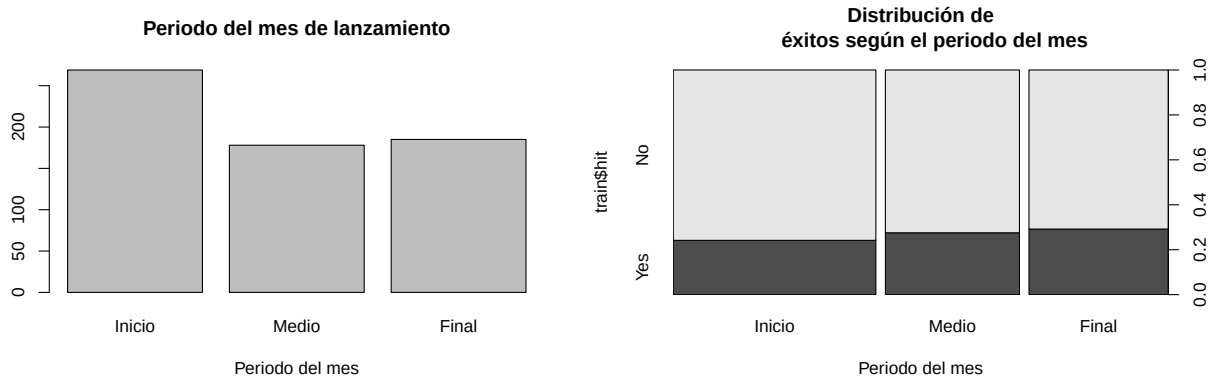
```
library(lubridate)

fecha_train <- make_date(train$released_year, train$released_month, train$released_day)
train$released_week_day <- factor(wday(fecha_train,
                            label = TRUE,
                            abbr = FALSE,
                            week_start = getOption("lubridate.week.start", 1)),
                            ordered = FALSE)

fecha_test <- make_date(test$released_year, test$released_month, test$released_day)
test$released_week_day <- factor(wday(fecha_test,
                            label = TRUE,
                            abbr = FALSE,
                            week_start = getOption("lubridate.week.start", 1)),
                            ordered = FALSE)

par(mfrow=c(1,2))
plot(train$released_week_day, main="Día de la semana de lanzamiento de las canciones")
plot(train$hit ~ train$released_week_day, main="Distribución de éxitos según el
     día de la semana",
     xlab= " ")
```





La mayoría de las canciones son lanzadas los viernes, pero no vemos la mayor tasa de hits en dicho día sino en el miércoles.

```
train$released_day <- as.integer(train$released_day)
train$released_month_period <- cut(
  train$released_day,
  breaks = c(0, 10, 20, 31),
  labels = c("Inicio", "Medio", "Final"),
  right = TRUE
)

test$released_day <- as.integer(test$released_day)
test$released_month_period <- cut(
  test$released_day,
  breaks = c(0, 10, 20, 31),
  labels = c("Inicio", "Medio", "Final"),
  right = TRUE
)

par(mfrow=c(1,2))
plot(train$released_month_period, main="Periodo del mes de lanzamiento",
     xlab="Periodo del mes")
plot(train$hit ~ train$released_month_period, main="Distribución de
     éxitos según el periodo del mes",
     xlab="Periodo del mes")
```




```
# eliminamos la variable released_day
train$released_day <- NULL
test$released_day <- NULL
```

Vemos mayor número de lanzamientos a inicios de mes, pero una tasa de hits ligeramente superior a finales.

released_month

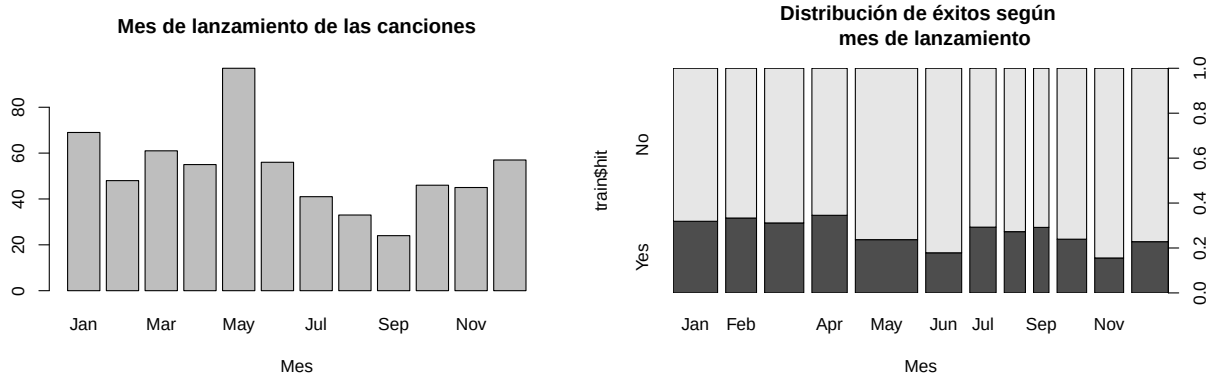
```
head(train$released_month)
```

```
## [1] May Mar Oct Jul Oct Jan
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

La variable `released_monthes` de tipo factor ordenado

Vamos a eliminar el orden, ya que no tiene sentido que un mes sea mayor o menor que otro.

```
train$released_month <- factor(train$released_month, ordered = FALSE)
test$released_month <- factor(test$released_month, ordered = FALSE)

par(mfrow=c(1,2))
plot(train$released_month, main="Mes de lanzamiento de las canciones",
                xlab="Mes")
plot(train$hit~train$released_month, main="Distribución de éxitos según
     mes de lanzamiento", xlab="Mes")
```



Parece haber una tasa mayor de hits en los primeros meses del año. Vemos que mayo, el último mes antes del verano, es el que más lanzamientos tiene.

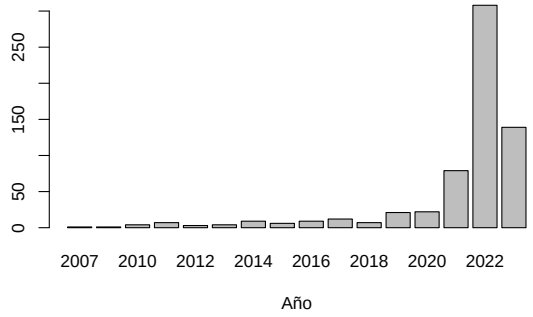
released_year

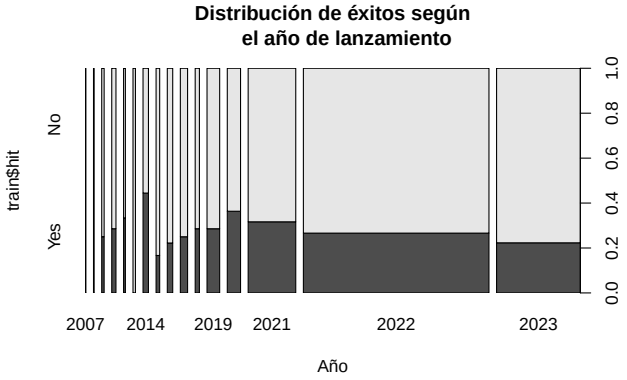
```
str(train$released_year)
```

```
##  int [1:632] 2023 2022 2022 2023 2019 2023 2021 2022 2023 2022 ...
```

La variable `released_year` está guardada como variable de tipo entero, pero en realidad es una variable categórica:

```
train$released_year <- as.factor(train$released_year)
test$released_year <- as.factor(test$released_year)

par(mfrow=c(1, 2))
plot(train$released_year, main="Año de lanzamiento de las canciones",
        xlab="Año")
plot(train$hit ~ train$released_year, main="Distribución de éxitos según
     el año de lanzamiento",
     xlab="Año")
```





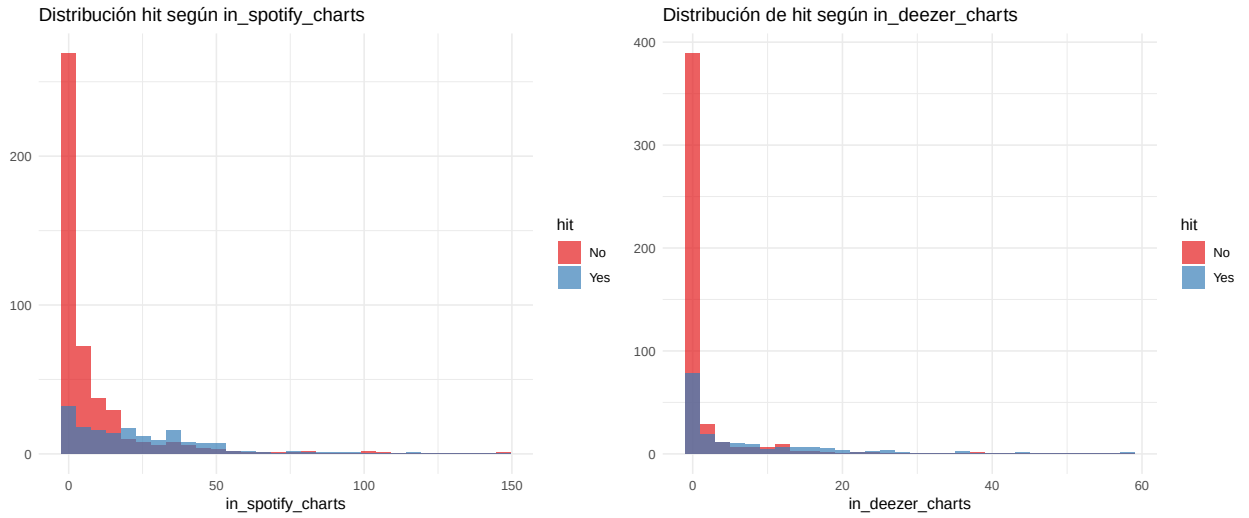

```
table(train$hit[train$released_year=="2013"])
```

```
##
##  No Yes
##   4   0
```

De nuevo, nos encontramos en una situación parecida al número de artistas en las canciones, años muy poco representados. Podemos observar también que hay años donde no hay hits, como el 2013.

in_spotify_charts e in_deezer_charts

```
library(ggplot2)
library(gridExtra)


p1 <- ggplot(train, aes(x = in_spotify_charts, fill = hit)) +
  geom_histogram(alpha = 0.7, position = "identity", bins = 30) +
  scale_fill_brewer(palette = "Set1") +
  labs(title = "Distribución hit según in_spotify_charts",
       x = "in_spotify_charts",
       y = NULL) +
  theme_minimal()

p2 <- ggplot(train, aes(x = in_deezer_charts, fill = hit)) +
  geom_histogram(alpha = 0.7, position = "identity", bins = 30) +
  scale_fill_brewer(palette = "Set1") +
  labs(title = "Distribución de hit según in_deezer_charts",
       x = "in_deezer_charts",
       y = NULL) +
  theme_minimal()


grid.arrange(p1, p2, ncol = 2)
```



Las variables `in_spotify_charts` e `in_deezer_charts` cuantifican la presencia y el ranking de una canción en las listas de éxitos de Spotify y Deezer, respectivamente. Ambas agregan en una única métrica si la canción entró en listas, cuánto tiempo permaneció y qué posiciones alcanzó.

En `in_spotify_charts` los valores oscilan entre 1 y 150, mientras que en `in_deezer_charts` lo hacen entre 1 y 60. En ambos casos, valores bajos corresponden a mejores posiciones, y valores altos a posiciones más bajas.

Los histogramas muestran que:

- La mayoría de canciones con `hit = No` se concentran en valores bajos o directamente en los primeros bins, lo que indica mucha presencia y rankings generalmente altos. En valores altos de `in_deezer_charts` es difícil encontrar canciones de esta clase.
- Las canciones con `hit = Yes` aparecen a lo largo de todo el rango de ambas variables, pero se concentran más en valores bajos, es decir, en posiciones altas de las listas. A diferencia de las canciones que no son hits, esta clase se distribuye de manera más uniforme en los primeros bins, mientras que los no hits tienen una distribución más parecida a una exponencial (esto se ve más claro en la variable `in_spotify_charts`).
- En resumen, aunque cualquier canción puede llegar a ser un hit independientemente de su rango exacto, los no éxitos tienden a no aparecer en posiciones intermedias y bajas de estas dos variables.

hit

```
table(df$hit)
```

```
##
##  No Yes
## 591 200
```

Cambiamos el nivel de referencia de la variable respuesta, de forma que el nivel de referencia sea `"Yes"` en lugar de `"No"`.

```r
train$hit <- relevel(as.factor(train$hit), ref = "Yes")
test$hit <- relevel(as.factor(test$hit), ref = "Yes")

table(train$hit)
```

```
##
## Yes  No
## 168 464
```

Podemos ver que las clases no están balanceadas. Aproximadamente una de cuatro canciones son éxitos.

```r
str(train)
```

df final

```
## 'data.frame':    632 obs. of  17 variables:
##  $ artist_count         : Factor w/ 8 levels "1","2","3","4",..: 1 1 1 2 1 1 1 3 1 2 ...
##  $ released_year        : Factor w/ 16 levels "2007","2008",..: 16 15 15 16 12 16 14 15 16 15 ...
##  $ released_month       : Factor w/ 12 levels "Jan","Feb","Mar",..: 5 3 10 7 10 1 6 11 2 8 ...
##  $ in_spotify_charts    : int  29 0 0 147 0 0 24 0 2 13 ...
##  $ in_deezer_charts     : int  11 0 0 10 0 0 3 0 0 3 ...
##  $ bpm                  : int  144 170 108 125 80 107 92 122 92 93 ...
##  $ mode                 : Factor w/ 2 levels "Major","Minor": 1 1 1 1 2 2 1 2 1 2 ...
##  $ danceability_.       : int  75 56 64 80 56 66 81 75 57 63 ...
##  $ valence_.            : int  35 52 4 89 19 47 39 45 68 34 ...
##  $ energy_.             : int  48 64 40 83 46 40 60 63 76 86 ...
##  $ acousticness_.       : int  84 11 6 31 92 72 31 6 7 26 ...
##  $ instrumentalness_.   : int  0 0 0 0 72 0 0 0 0 0 ...
##  $ liveness_.           : int  10 45 10 8 11 11 7 35 33 21 ...
##  $ speechiness_.        : int  12 7 6 4 3 3 3 12 3 39 ...
##  $ hit                  : Factor w/ 2 levels "Yes","No": 2 1 2 2 2 2 2 2 2 2 ...
##  $ released_week_day    : Factor w/ 7 levels "lunes","martes",..: 4 6 5 5 5 5 5 5 5 4 ...
##  $ released_month_period: Factor w/ 3 levels "Inicio","Medio",..: 2 2 3 2 1 3 2 1 2 3 ...
```

Primero ajustamos un árbol de decisión completo:

```r
library(rpart)

set.seed(semilla)

tree <- rpart(hit ~ ., data = train, cp = 0, method = "class")
tree
```

```
## n= 632
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 632 168 No (0.26582278 0.73417722)
```

```
##     2) in_spotify_charts>=16.5 152  61 Yes (0.59868421 0.40131579)
##       4) released_year=2014,2018,2021,2022 66  10 Yes (0.84848485 0.15151515)
##         8) released_month=Jan,Feb,Mar,Apr,May,Jul,Aug,Sep,Oct 40   0 Yes (1.00000000 0.00000000) *
##         9) released_month=Jun,Nov,Dec 26  10 Yes (0.61538462 0.38461538)
##          18) liveness_.< 10.5 10   1 Yes (0.90000000 0.10000000) *
##          19) liveness_.>=10.5 16   7 No (0.43750000 0.56250000) *
##       5) released_year=2007,2008,2010,2011,2012,2015,2016,2017,2019,2020,2023 86  35 No (0.40697674 0
##        10) released_month=Jan,Feb,Mar,Apr,Aug,Oct,Nov 50  19 Yes (0.62000000 0.38000000)
##          20) released_year=2011,2015,2016,2017,2023 43  13 Yes (0.69767442 0.30232558)
##            40) valence_.>=42.5 29   5 Yes (0.82758621 0.17241379) *
##            41) valence_.< 42.5 14   6 No (0.42857143 0.57142857) *
##          21) released_year=2007,2008,2012,2019,2020 7   1 No (0.14285714 0.85714286) *
##        11) released_month=May,Jun,Jul 36   4 No (0.11111111 0.88888889) *
##     3) in_spotify_charts< 16.5 480  77 No (0.16041667 0.83958333)
##       6) in_deezer_charts>=13.5 10   0 Yes (1.00000000 0.00000000) *
##       7) in_deezer_charts< 13.5 470  67 No (0.14255319 0.85744681)
##        14) released_year=2010,2012,2014,2019,2020,2021 116  33 No (0.28448276 0.71551724)
##          28) released_month=Jan,Mar,Apr,May 36  13 Yes (0.63888889 0.36111111)
##            56) acousticness_.>=22.5 13   1 Yes (0.92307692 0.07692308) *
##            57) acousticness_.< 22.5 23  11 No (0.47826087 0.52173913)
##             114) valence_.>=57.5 12   3 Yes (0.75000000 0.25000000) *
##             115) valence_.< 57.5 11   2 No (0.18181818 0.81818182) *
##          29) released_month=Feb,Jun,Jul,Aug,Sep,Oct,Nov,Dec 80  10 No (0.12500000 0.87500000)
##            58) in_deezer_charts>=1.5 7   3 Yes (0.57142857 0.42857143) *
##            59) in_deezer_charts< 1.5 73   6 No (0.08219178 0.91780822) *
##        15) released_year=2011,2013,2015,2016,2017,2018,2022,2023 354  34 No (0.09604520 0.90395480)
##          30) in_spotify_charts>=5.5 85  20 No (0.23529412 0.76470588)
##            60) released_year=2018,2022 47  17 No (0.36170213 0.63829787)
##             120) released_month=Jan,Apr,May,Jul,Aug,Oct 23  10 Yes (0.56521739 0.43478261)
##               240) released_week_day=miércoles,viernes,domingo 16   4 Yes (0.75000000 0.25000000) *
##               241) released_week_day=lunes,martes,jueves 7   1 No (0.14285714 0.85714286) *
##             121) released_month=Feb,Mar,Jun,Sep,Nov,Dec 24   4 No (0.16666667 0.83333333) *
##            61) released_year=2013,2015,2016,2017,2023 38   3 No (0.07894737 0.92105263) *
##          31) in_spotify_charts< 5.5 269  14 No (0.05204461 0.94795539) *
```

```
tree$cptable
```

```
##            CP nsplit rel error    xerror       xstd
## 1 0.178571429      0 1.0000000 1.0000000 0.06610675
## 2 0.095238095      1 0.8214286 0.8333333 0.06214107
## 3 0.071428571      2 0.7261905 0.8095238 0.06149538
## 4 0.059523810      3 0.6547619 0.7440476 0.05960616
## 5 0.029761905      4 0.5952381 0.6845238 0.05773330
## 6 0.020833333      7 0.5059524 0.7202381 0.05887560
## 7 0.011904762      9 0.4642857 0.6964286 0.05812042
## 8 0.005952381     14 0.4047619 0.7202381 0.05887560
## 9 0.000000000     17 0.3869048 0.7559524 0.05996250
```

El árbol con un único nodo, CP=0.178571429, predice siempre la categoría mayoritaria para minimizar el error. En nuestro caso, dado que la proporción de hit="No" (0.7341772) es mayor que la de hit="Yes" (0.2658228), el árbol clasifica todas las observaciones como "No".

El árbol completo, CP=0, utiliza 18 divisiones y consigue un error relativo de 0.375 en los datos de entrenamiento, es decir, disminuye el error más de un 60% con respecto al árbol con solo el nodo raíz. Sin embargo

el error de validación cruzada es de 0.8452381, lo que indica que el árbol completo se está sobreajustando a los datos de entrenamiento.

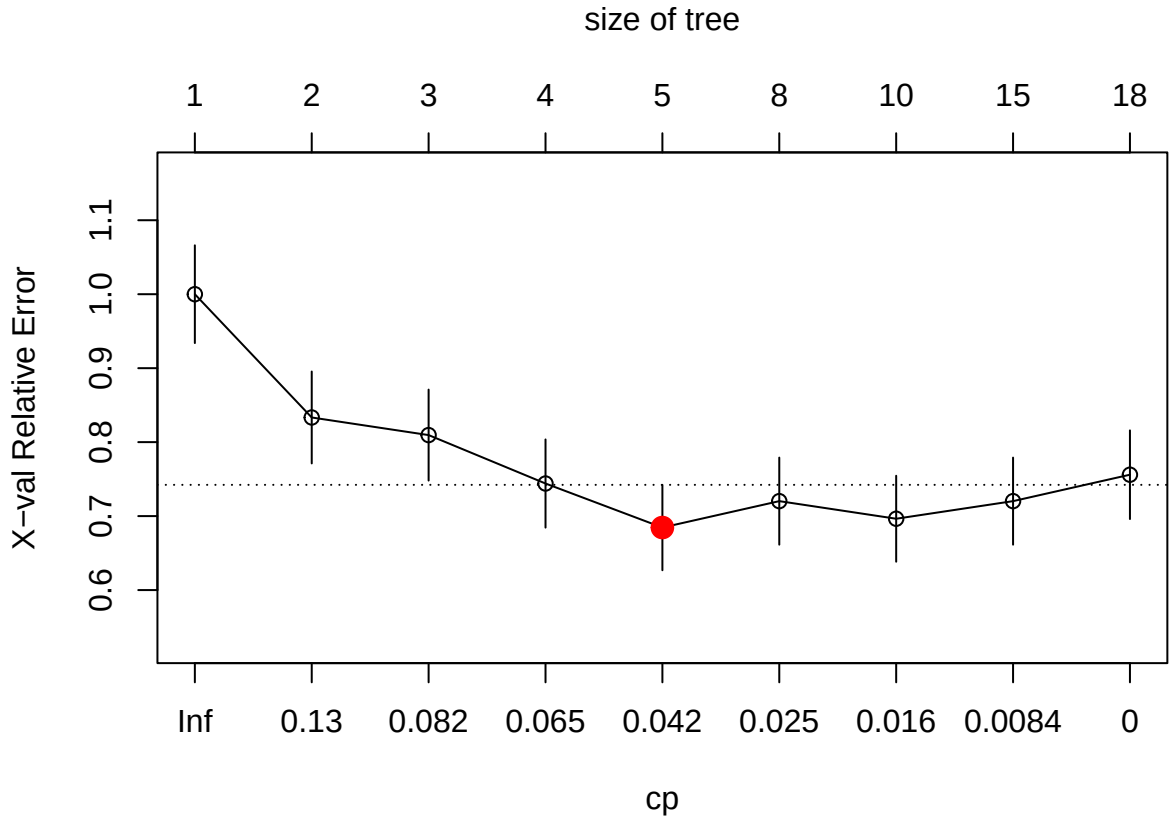
Para obtener el parámetro de complejidad óptimo seguimos el criterio de un error estándar de Breiman et al. (1984):

```
xerror <- tree$cptable[,"xerror"]
imin.xerror <- which.min(xerror)
upper.xerror <- xerror[imin.xerror] + tree$cptable[imin.xerror, "xstd"]
icp <- min(which(xerror <= upper.xerror))

# Parámetro de complejidad óptimo
cp <- tree$cptable[icp, "CP"]; cp
```

```
## [1] 0.0297619
```

Podemos representar el error de validación cruzada y el parámetro de complejidad óptimo seleccionado:

```
plotcp(tree)
points(x = icp, y = tree$cptable[icp, "xerror"], col = "red", pch = 19, cex = 1.5)
```



Utilizando el CP óptimo podamos el árbol y lo representamos:

```
library(rpart.plot)

tree <- prune(tree, cp = cp)
tree
```

```
## n= 632
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 632 168 No (0.2658228 0.7341772)
##    2) in_spotify_charts>=16.5 152  61 Yes (0.5986842 0.4013158)
##      4) released_year=2014,2018,2021,2022 66  10 Yes (0.8484848 0.1515152) *
##      5) released_year=2007,2008,2010,2011,2012,2015,2016,2017,2019,2020,2023 86  35 No (0.4069767 0.5
##       10) released_month=Jan,Feb,Mar,Apr,Aug,Oct,Nov 50  19 Yes (0.6200000 0.3800000) *
##       11) released_month=May,Jun,Jul 36   4 No (0.1111111 0.8888889) *
##    3) in_spotify_charts< 16.5 480  77 No (0.1604167 0.8395833)
##      6) in_deezer_charts>=13.5 10   0 Yes (1.0000000 0.0000000) *
##      7) in_deezer_charts< 13.5 470  67 No (0.1425532 0.8574468) *
```

Interpretamos el árbol

---

1. Primera división: `in_spotify_charts`

La primera variable que se utiliza para dividir el árbol es ***in_spotify_charts***.

Cuando ***in_spotify_charts >= 16.5***, es decir cuando la canción aparece en posiciones medias-altas en las listas de Spotify, la canción se clasifica como éxito.

Errores tras el split:

- El nodo raíz comete **168 errores** de clasificación.
- Tras esta primera división, el número total de errores se reduce a **138** (61 en el nodo izquierdo y 77 en el derecho).
- El error baja de **0.2658228** a **0.2183544** (138/632).

En la tabla `tree$cptable` podemos ver que con `split = 1`, **rel.error = 0.8214286**, lo cual coincide con (0.2183544 / 0.2658228).

---

2. Segunda división: `released_year`

La siguiente variable que se utiliza es ***released_year***.

Si ***in_spotify_charts >= 16.5*** y ***released_year = 2014, 2018, 2021, 2022***, la canción se clasifica como éxito.

Canciones con posiciones medias-altas en Spotify y cuyo año de lanzamiento fue uno de esos valores se consideran éxitos.

Errores tras esta división:

- En el nodo izquierdo se cometen **10 errores**.
- En el nodo derecho, **35 errores**.
- El nodo padre tenía un error de **0.4013158 = 61/152**.

- Con esta nueva regla se baja a **0.2960526 = (10 + 35) / 152**.

El árbol pasa de un error de **0.2183544** a **((10 + 35 + 77) / 632) = 0.193038**. El error relativo es **(0.193038 / 0.2658228) = 0.7261905**, coincidiendo con `tree$cptable` en `split = 3`.

---

3. Tercera división: `released_month`

La siguiente división se realiza sobre ***released_month***.

Para el nodo donde:

- ***in_spotify_charts >= 16.5***, y
- ***released_year = 2007, 2008, 2010, 2011, 2012, 2015, 2016, 2017, 2019, 2020 o 2023***,

el árbol distingue según el mes de lanzamiento.

Si ***released_month = Jan, Feb, Mar, Apr, Aug, Oct o Nov***, la canción se clasifica como éxito.

Errores tras esta división:

- Nodo izquierdo: **19 errores**.
- Nodo derecho: **4 errores**.
- Nodo padre: **35/86 = 0.4069767**.
- Error tras la regla: **(19 + 4)/86 = 0.2674419**.

El árbol ahora tiene un error de **((10 + (19 + 4) + 77) / 632) = 0.1740506**, y un error relativo de **(0.1740506 / 0.2658228) = 0.6547617**, coincidiendo con `tree$cptable` en `split = 4`.

---

4. Cuarta división (última): `in_deezer_charts`

La cuarta y última variable utilizada es ***in_deezer_charts***.

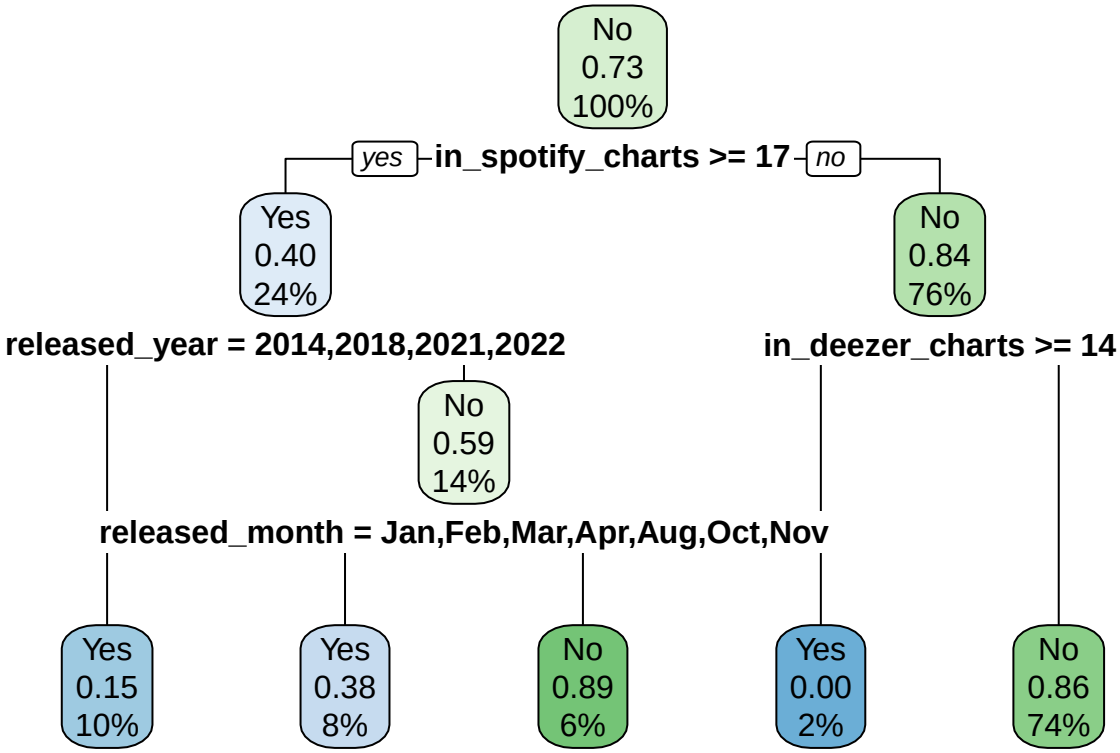
Si **in_spotify_charts < 16.5** y ***in_deezer_charts >= 13.5***, la canción se clasifica como éxito.

Errores en esta división:

- Nodo izquierdo: **0 errores** (las 10 canciones que cumplen la regla son éxitos).
- Nodo derecho: **67 errores**.
- Nodo padre: **77/480 = 0.1604167**.
- Error tras la regla: **67/480 = 0.1395833**.

El error del árbol podado es **((10 + (19 + 4) + (0 + 67)) / 632) = 0.1582278**. El error relativo es **0.1582278 / 0.2658228 = 0.5952379**, coincidiendo con `tree$cptable` en `split = 5`.

```
rpart.plot(tree)
```



Importancia de las variables

```
importance <- tree$variable.importance
importance <- round(100*importance/sum(importance), 1)
importance
```

```
##     in_spotify_charts      in_deezer_charts        released_month
##                  39.1                  24.2                  13.4
##         released_year released_month_period     released_week_day
##                  13.0                   2.0                   2.0
##              energy_.                   bpm        danceability_.
##                   2.0                   1.9                   1.5
##            liveness_.
##                   1.1
```

De esta tabla se concluye que las posiciones en las listas de *Spotify* es la variables más decisiva a la hora de predecir si una canción será un éxito.

- *in_spotify_charts* es la variable más importante.
- Le sigue *released_month*, *in_deezer_charts* y *released_year*.
- Las demás variables no tienen gran importancia en nuestro árbol.

En resumen, a la hora de decidir si una canción es un éxito o no lo más importante es aparecer en las listas superiores de Spotify.

A continuación evaluamos la precisión de las predicciones en la muestra de test:

```r
obs <- test$hit
pred <- predict(tree, newdata = test, type = "class")
caret::confusionMatrix(pred, obs, mode = "everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  17  10
##        No   15 117
##
##                Accuracy : 0.8428
##                  95% CI : (0.7767, 0.8956)
##     No Information Rate : 0.7987
##     P-Value [Acc > NIR] : 0.09668
##
##                   Kappa : 0.4806
##
##  Mcnemar's Test P-Value : 0.42371
##
##             Sensitivity : 0.5312
##             Specificity : 0.9213
##          Pos Pred Value : 0.6296
##          Neg Pred Value : 0.8864
##               Precision : 0.6296
##                  Recall : 0.5312
##                      F1 : 0.5763
##              Prevalence : 0.2013
##          Detection Rate : 0.1069
##    Detection Prevalence : 0.1698
##       Balanced Accuracy : 0.7263
##
##        'Positive' Class : Yes
##
```

Aunque el árbol tiene una precisión global del 84.28%, este valor está influido por el fuerte desbalance entre clases, tal como indica el alto valor de `No Information Rate` (0.7987 = `Accuracy` si clasificamos todos los datos a la clase mayoritaria). La `balanced accuracy` (72.63%) confirma que, al corregir el efecto del desbalance entre clases, el rendimiento del árbol es moderado. El coeficiente Kappa, que es adecuado para evaluar problemas con clases desbalanceadas, es de 0.4806, indicando una concordancia moderada entre las predicciones y las observaciones reales.

El clasificador destaca por una especificidad muy alta (0.9213), lo que significa que clasifica correctamente la gran mayoría de canciones que no son éxitos. Sin embargo, su sensibilidad es baja (0.5312), es decir, identifica aproximadamente la mitad de las canciones que realmente son éxitos. Esto revela que el árbol tiende a ser conservador y a predecir "No" en caso de duda.

En conjunto, el árbol de decisión es útil para descartar canciones que probablemente no serán un éxito, pero tiene dificultades para detectar los verdaderos éxitos.

Las estimaciones de la probabilidad de que una canción sea un éxito en la muestra de test son:

```r
p.est <- predict(tree, newdata = test, type="prob")[,"Yes"]
head(data.frame(Prob_est = p.est,
                hit_obs = test$hit), 10) # mostramos solo las 10 primeras
```

```
##      Prob_est hit_obs
## 819 0.1425532      No
## 272 0.8484848     Yes
## 828 0.1425532      No
## 854 0.1425532      No
## 414 0.1425532      No
## 415 0.1425532      No
## 865 0.1425532      No
## 229 0.1425532      No
## 58  0.8484848     Yes
## 702 0.1425532      No
```

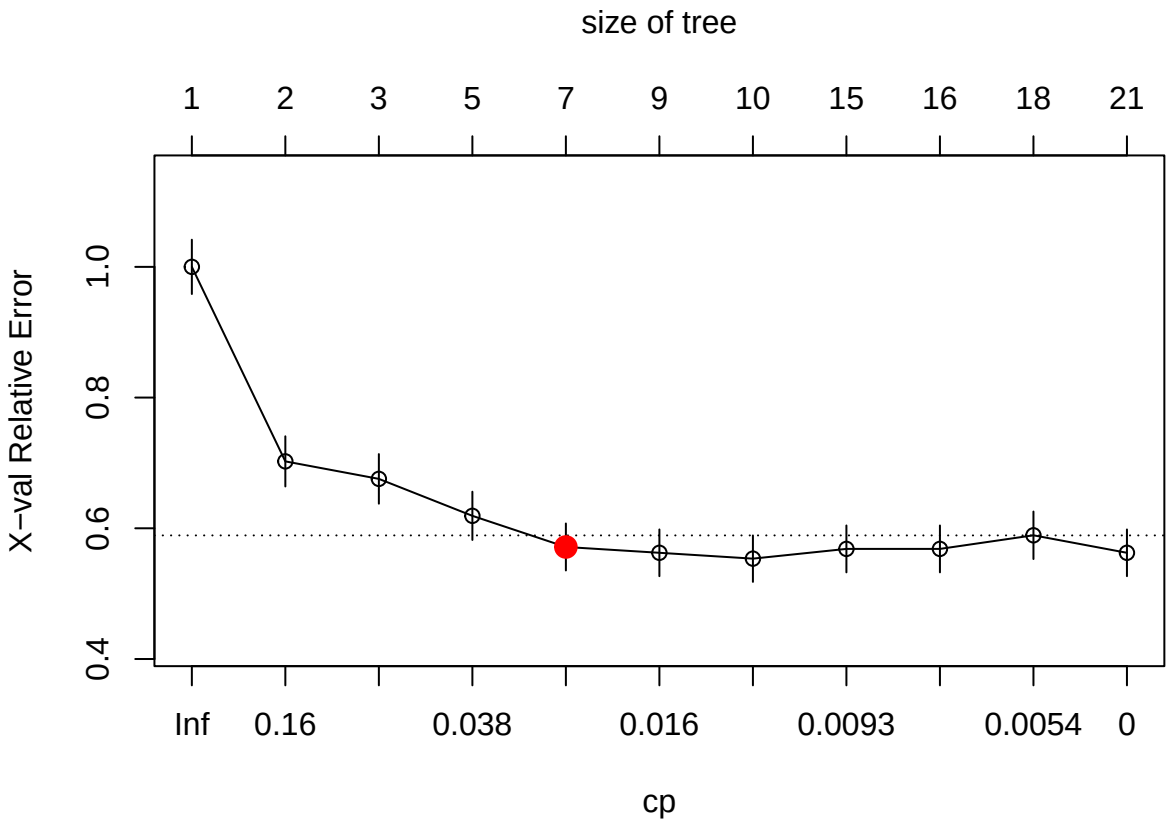
Daremos mayores pesos a los éxitos con el fin de mitigar el efecto del desbalanceo. Se asignará el doble de peso a los éxitos, de modo que clasificar erróneamente un éxito equivale a clasificar erróneamente dos canciones que no lo son.

```
freq <- table(train$hit)
weights <- ifelse(train$hit == "Yes",
                  2,
                  1)

set.seed(semilla)
tree <- rpart(hit ~ ., data = train, method = "class",
              weights = weights,
              cp = 0)
```

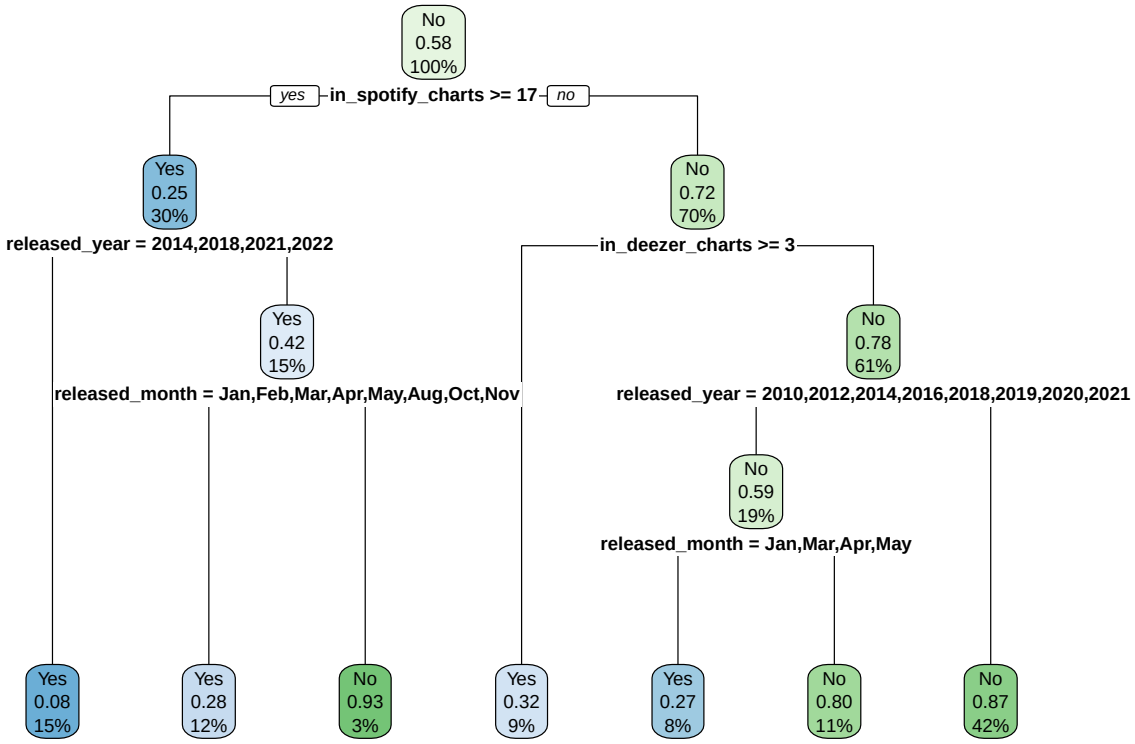
```
xerror <- tree$cptable[,"xerror"]
imin.xerror <- which.min(xerror)
upper.xerror <- xerror[imin.xerror] + tree$cptable[imin.xerror, "xstd"]
icp <- min(which(xerror <= upper.xerror))

# Parámetro de complejidad óptimo
cp <- tree$cptable[icp, "CP"]; cp
```

```
## [1] 0.01636905
```

```
plotcp(tree)
points(x = icp, y = tree$cptable[icp, "xerror"], col = "red", pch = 19, cex = 1.5)
```




```
tree <- prune(tree, cp = cp)
rpart.plot(tree)
```




```
obs <- test$hit
pred <- predict(tree, newdata = test, type = "class")
caret::confusionMatrix(pred, obs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  21  22
##        No   11 105
##
##                Accuracy : 0.7925
##                  95% CI : (0.7211, 0.8526)
##     No Information Rate : 0.7987
##     P-Value [Acc > NIR] : 0.62343
##
##                   Kappa : 0.428
##
##  Mcnemar's Test P-Value : 0.08172
##
##             Sensitivity : 0.6562
##             Specificity : 0.8268
##          Pos Pred Value : 0.4884
##          Neg Pred Value : 0.9052
##              Prevalence : 0.2013
```

```
##          Detection Rate : 0.1321
##    Detection Prevalence : 0.2704
##       Balanced Accuracy : 0.7415
##
##        'Positive' Class : Yes
##
```

Al asignarle mayor peso a las canciones que son éxitos, el árbol aumenta notablemente su capacidad para detectarlas.

La sensibilidad sube de 0.5312 a 0.6562. Este cambio implica sacrificar parte de la especificidad (de 0.9213 a 0.8268), ya que el modelo ahora predice más "Yes" y comete más falsos positivos. Aun así, la `balanced accuracy` (de 0.7263 a 0.7415) mejora ligeramente respecto al árbol original, lo que indica un rendimiento más equilibrado entre ambas clases.

En resumen, este modelo es menos preciso en términos globales, pero resulta más adecuado cuando el objetivo es no dejar escapar canciones que podrían convertirse en un éxito. No obstante, una sensibilidad del 65% sigue siendo relativamente baja.

Ejercicio 2

Realizar la clasificación anterior empleando Bosques Aleatorios mediante el método `"rf"` del paquete `caret` considerando 300 árboles.

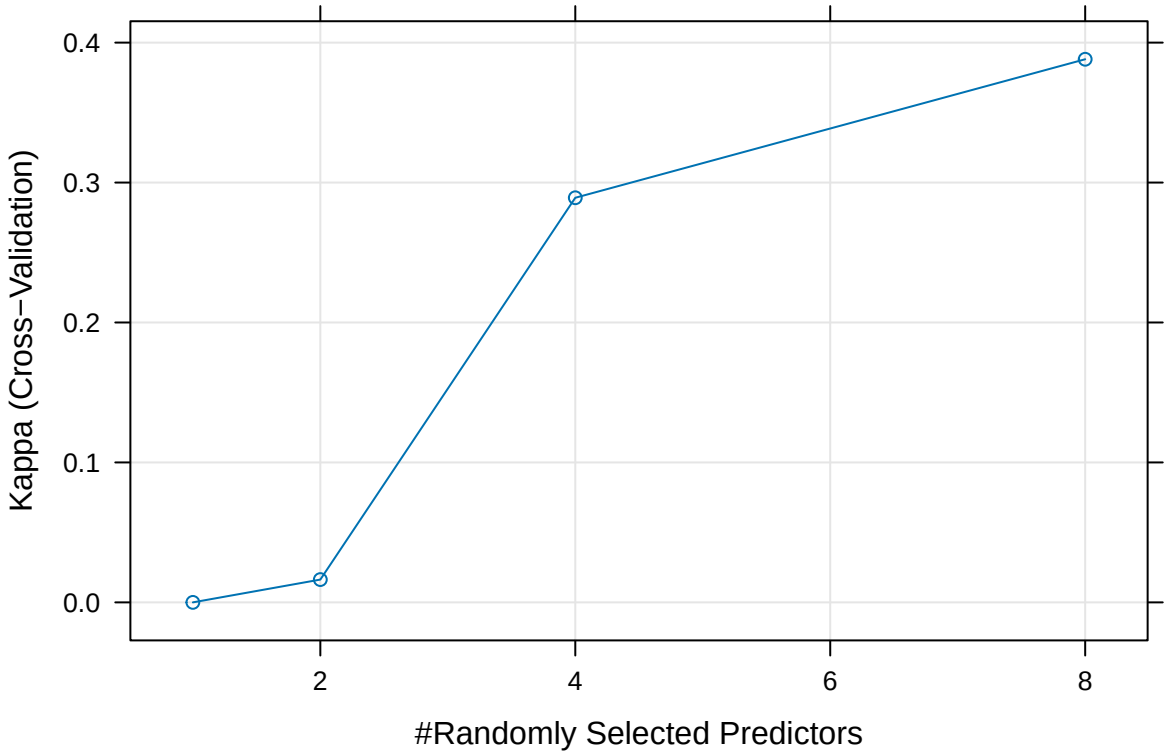
a. Seleccionar el número de predictores empleados en cada división `mtry = c(1, 2, 4, 8)` de forma que se minimice el error de validación cruzada con 5 grupos.

```r
library(caret)

set.seed(semilla)
control <- trainControl(method = "cv", number = 5)
tunegrid <- expand.grid(mtry = c(1, 2, 4, 8))
rf_model <- caret::train(hit ~ ., data = train, method = "rf",
                  tuneGrid = tunegrid,
                  trControl = control,
                  ntree = 300,
                  localImp = TRUE,
                  metric = "Kappa")
# Utilizamos como métrica objetivo Kappa en lugar
# de Accuracy para manejar el desbalanceo

rf_model
```

```
## Random Forest
##
## 632 samples
##  16 predictor
##   2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 506, 505, 506, 506, 505
```

```
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.7341832  0.00000000
##   2     0.7341957  0.01628226
##   4     0.7785902  0.28913337
##   8     0.7927884  0.38811466
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```

```
plot(rf_model)
```



Escogemos la métrica Kappa para optimizar la selección de hiperparámetros, ya que está métrica es más robusta al desbalanceo de la clase objetivo que la precisión.

Vemos que el modelo con `mtry = 8` es el que maximiza el valor de Kappa y, en este caso, también la precisión aunque no es concluyente debido al desbalanceo de la respuesta.

```
rf_model$xlevels
```

```
## $artist_count
## [1] "1" "2" "3" "4" "5" "6" "7" "8"
##
## $released_year
##  [1] "2007" "2008" "2010" "2011" "2012" "2013" "2014" "2015" "2016" "2017"
```

```
## [11] "2018" "2019" "2020" "2021" "2022" "2023"
##
## $released_month
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
##
## $mode
## [1] "Major" "Minor"
##
## $released_week_day
## [1] "lunes"     "martes"    "miércoles" "jueves"    "viernes"   "sábado"
## [7] "domingo"
##
## $released_month_period
## [1] "Inicio" "Medio"  "Final"
```

```
rf_model$contrasts
```

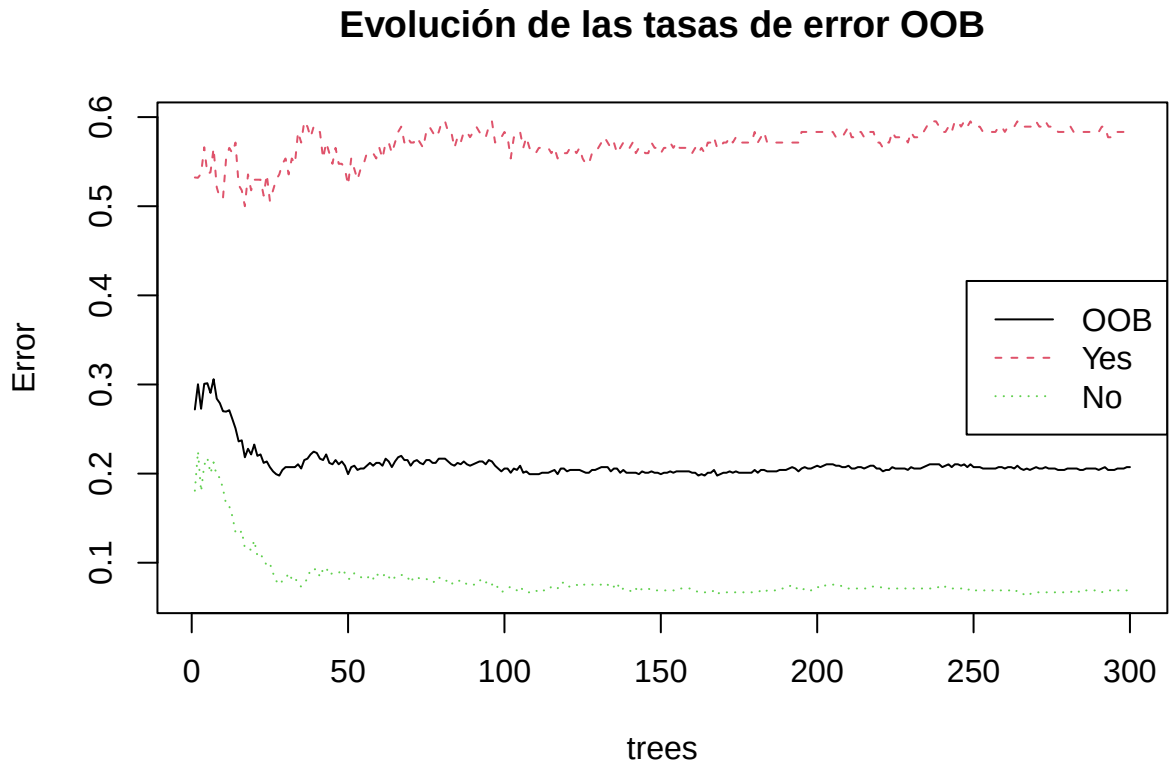
```
## $artist_count
## [1] "contr.treatment"
##
## $released_year
## [1] "contr.treatment"
##
## $released_month
## [1] "contr.treatment"
##
## $mode
## [1] "contr.treatment"
##
## $released_week_day
## [1] "contr.treatment"
##
## $released_month_period
## [1] "contr.treatment"
```

Caret ajusta automáticamente los niveles de las variables categóricas, no hace falta usar model.matrix:

```
# si ejecutamos esto obtenemos el mismo modelo:
# X <- model.matrix(hit ~ ., data = train)
# set.seed(semilla)
# rf_model <- caret::train(X[, -1], train$hit, method = "rf",
#                    tuneGrid = tunegrid,
#                    trControl = control,
#                    ntree = 300,
#                    localImp = TRUE,
#                    metric = "Kappa")
#
# rf_model
```

b. Representar la convergencia del error en las muestras OOB en el modelo final.

```r
oob_errors <- rf_model$finalModel$err.rate
plot(rf_model$finalModel, main = "Evolución de las tasas de error OOB")
legend("right", colnames(oob_errors), lty = 1:5, col = 1:6)
```



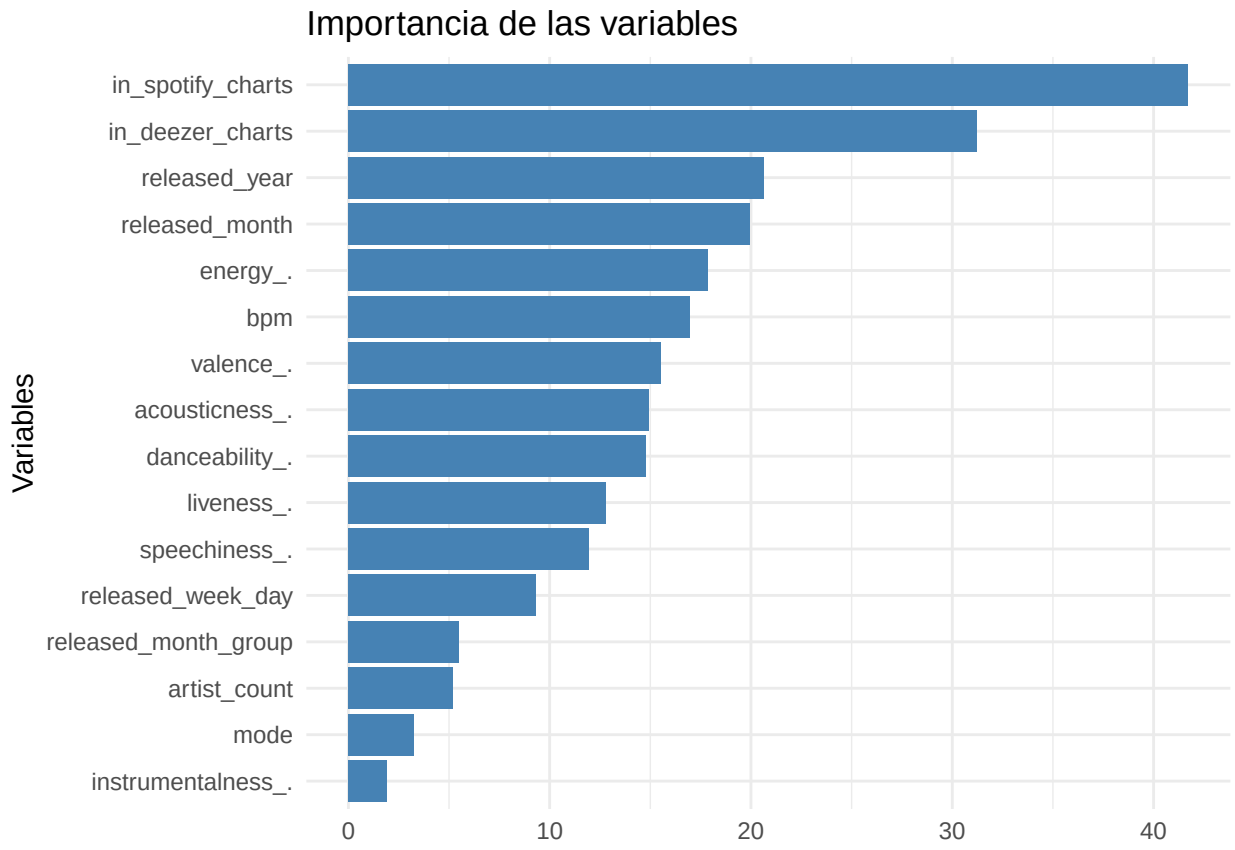
Vemos cómo el error OOB (Out-Of-Bag) disminuye rápidamente al principio y se estabiliza alrededor de 0.2 a medida que se añaden más árboles. Esto sugiere que un modelo más sencillo con menos árboles podría alcanzar un error similar. Aunque el error no aumenta con más árboles (no hay sobreajuste), usar 300 puede ser computacionalmente costoso. Dado que el dataset es pequeño, mantener los 300 árboles es aceptable.

Respecto al rendimiento por clases, la línea de error para la clase "Yes" (éxitos) es notablemente más alta que para la clase "No". El error para los éxitos converge alrededor de 0.55, mientras que para los no-éxitos se estabiliza por debajo de 0.1. Esto indica que el modelo tiene más dificultad para clasificar correctamente las canciones que son éxitos, coincidiendo con lo observado en el árbol de decisión del ejercicio anterior.

c. Estudiar la importancia de las variables y representar el efecto parcial de la variable más importante. Analizar también las interacciones de segundo orden entre las 4 variables más importantes (se pueden emplear las funciones de la librería `randomForestExplainer`).

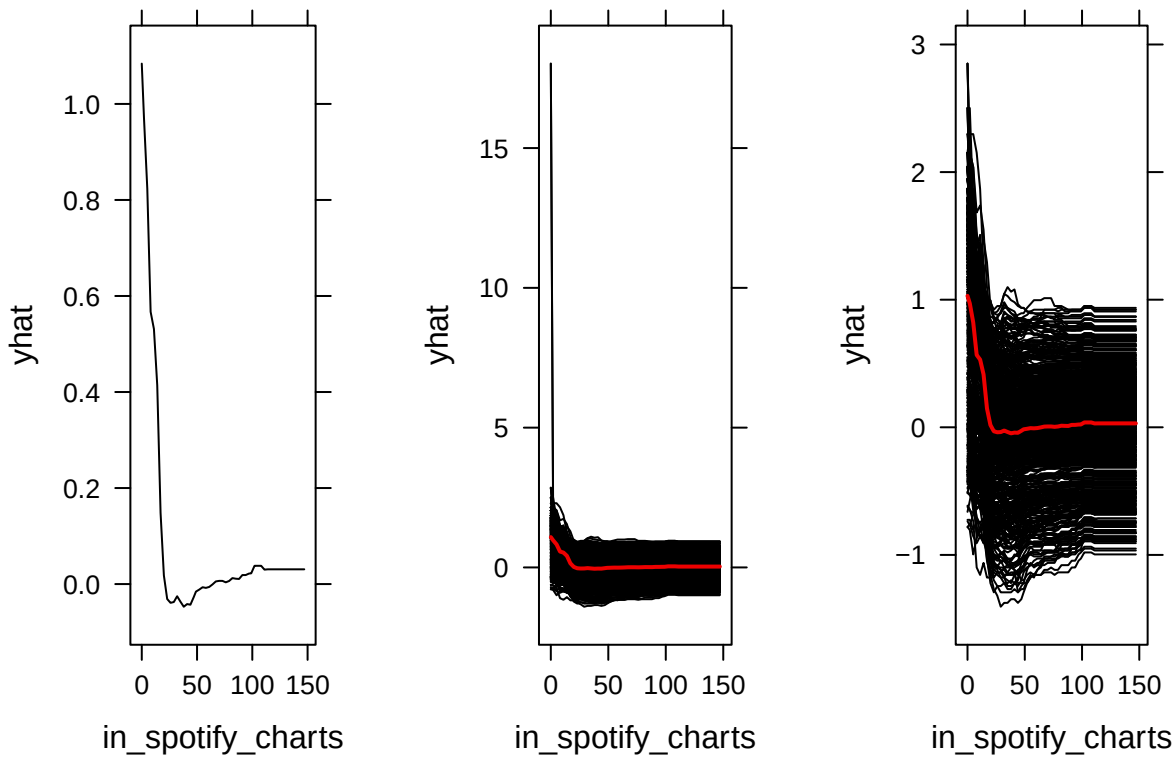
```r
library(dplyr)
library(randomForest)

var_Imp <- as.data.frame(randomForest::importance(rf_model$finalModel))


# Sumamos la importancia de las variables según categorías y graficamos
```

```r
importance_sum <- data.frame(
    variable = rownames(var_Imp[,4, drop=FALSE]),
    Importance = var_Imp[,4, drop=FALSE][,1]
) %>%
    mutate(group = case_when(
        grepl("^artist_count", variable) ~ "artist_count",
        grepl("^released_month_period", variable) ~ "released_month_group",
        grepl("^released_month", variable) ~ "released_month",
        grepl("^released_year", variable) ~ "released_year",
        grepl("^mode", variable) ~ "mode",
        grepl("^released_week_day", variable) ~ "released_week_day",
        TRUE ~ variable
    )) %>%
    group_by(group) %>%
    summarise(total_importance = sum(Importance)) %>%
    arrange(desc(total_importance))


ggplot(importance_sum, aes(x = reorder(group, total_importance), y = total_importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Importancia de las variables",
       x = "Variables",
       y = NULL) +
  theme_minimal()
```



Las cuatro variables más destacables según su importancia son: `in_spotify_charts`, `in_deezer_charts`, `released_year` y `released_month` (`in_spotify_charts` y `in_deezer_charts` bastante por encima de las otras).

```
library(pdp)

# Efecto parcial de in_spotify_charts sobre la respuesta
pdp1 <- partial(rf_model, "in_spotify_charts" , which.class = "No")
p1 <- plotPartial(pdp1)
# which.class = "No" para evitar valores negativos de yhat

# Efecto individual en cada observación de in_spotify_charts sobre la respuesta
ice1 <- partial(rf_model, pred.var = "in_spotify_charts", ice = TRUE, which.class = "No")
p2 <- plotPartial(ice1)

ice1_max <- ice1[ice1$yhat == max(ice1$yhat), ]
p3 <- plotPartial(ice1[ice1$yhat.id != ice1_max$yhat.id, ])

gridExtra:::grid.arrange(p1, p2, p3,  ncol = 3)
```



El gráfico de la izquierda muestra el efecto parcial promedio (PDP) de la variable `in_spotify_charts` sobre la probabilidad de que una canción no sea un *hit* (manteniendo constantes las demás variables). El gráfico del medio representa las curvas de efecto individual (ICE) para cada observación del conjunto de datos, y el gráfico de la derecha muestra las curvas ICE tras eliminar las dos observaciones que alcanzaban el valor máximo en la predicción.

Recordemos que `in_spotify_charts` es una métrica que combina presencia y *ranking* en las listas de Spotify,

donde valores bajos indican posiciones altas y valores altos indican posiciones más bajas.

En el PDP podemos ver que valores muy bajos tienen una alta probabilidad de que una canción no sea un *hit*. La probabilidad cae rápidamente y, a partir de valores superiores a 20 (aprox.), esta probabilidad se estabiliza en torno a cero. Es decir, valores altos de esta variable reflejan una baja probabilidad de que una canción no sea un *hit*, o lo que es lo mismo, valores altos de esta variable reflejan una alta probabilidad de que la canción sea un *hit*.

El gráfico ICE confirma que este patrón es consistente. Únicamente se pueden ver dos muestras que tienen inicialmente una probabilidad de `hit = No` muy superior a las demás. El resto de curvas siguen el patrón comentado en el gráfico PDP.

```
rf_model$trainingData[ice1_max$yhat.id, ]
```

```
##     .outcome artist_count released_year released_month in_spotify_charts
## 516       No            1          2022            Jan                 0
## 777       No            1          2022            May                 0
##     in_deezer_charts bpm  mode danceability_. valence_. energy_. acousticness_.
## 516                0  87 Minor             49        49       59             44
## 777                0 134 Major             78        51       43             69
##     instrumentalness_. liveness_. speechiness_. released_week_day
## 516                  0         35            21           viernes
## 777                  0         14             9           viernes
##     released_month_period
## 516                Inicio
## 777                 Medio
```

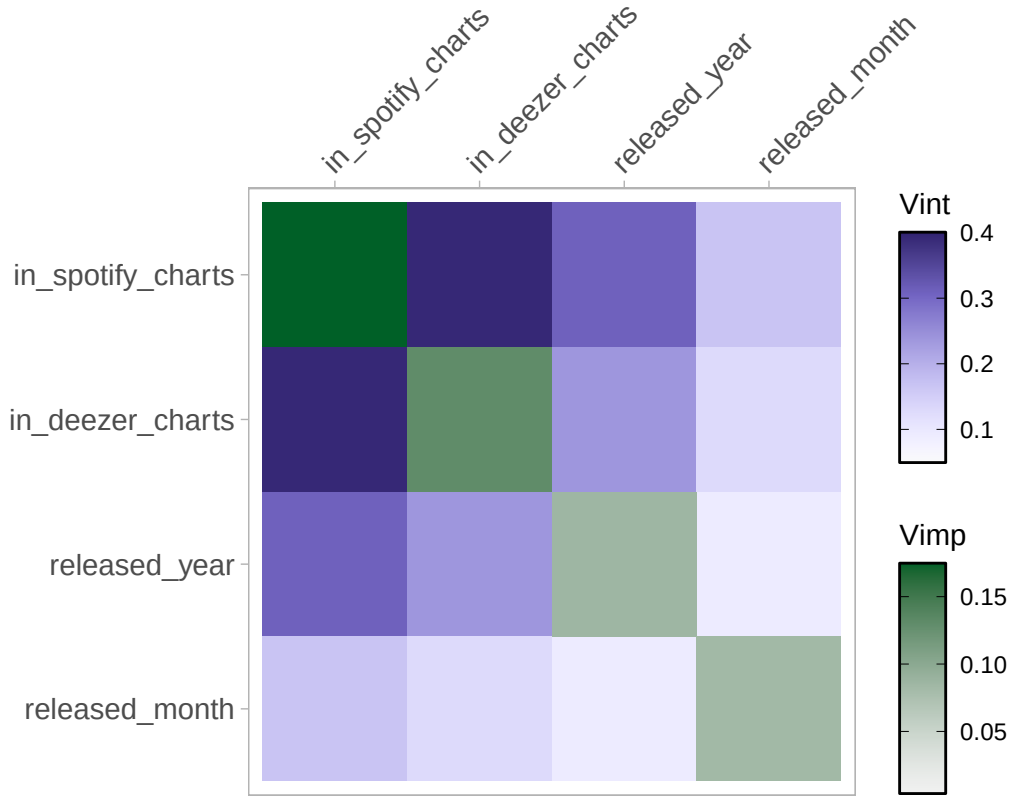
Las observaciones que alcanzan el valor máximo de la predicción en el ICE correspondiente a la variable `in_spotify_charts` son las canciones situadas en las filas 259 y 344 del conjunto de entrenamiento.

Ambas canciones presentan `in_spotify_charts = 0` y están etiquetadas como "No éxito". El modelo interpreta que para valores muy cercanos a 0 de `in_spotify_charts` la probabilidad de pertenecer a la clase no éxito es muy alta. Aumentando el valor de esta variable, y manteniendo las demás variables, la clase más probable es éxito.

```
library(vivid)
set.seed(semilla)
fit_rf <- vivi(data = train,
               fit = rf_model,
               response = "hit",
               importanceType = "%IncMSE")
```

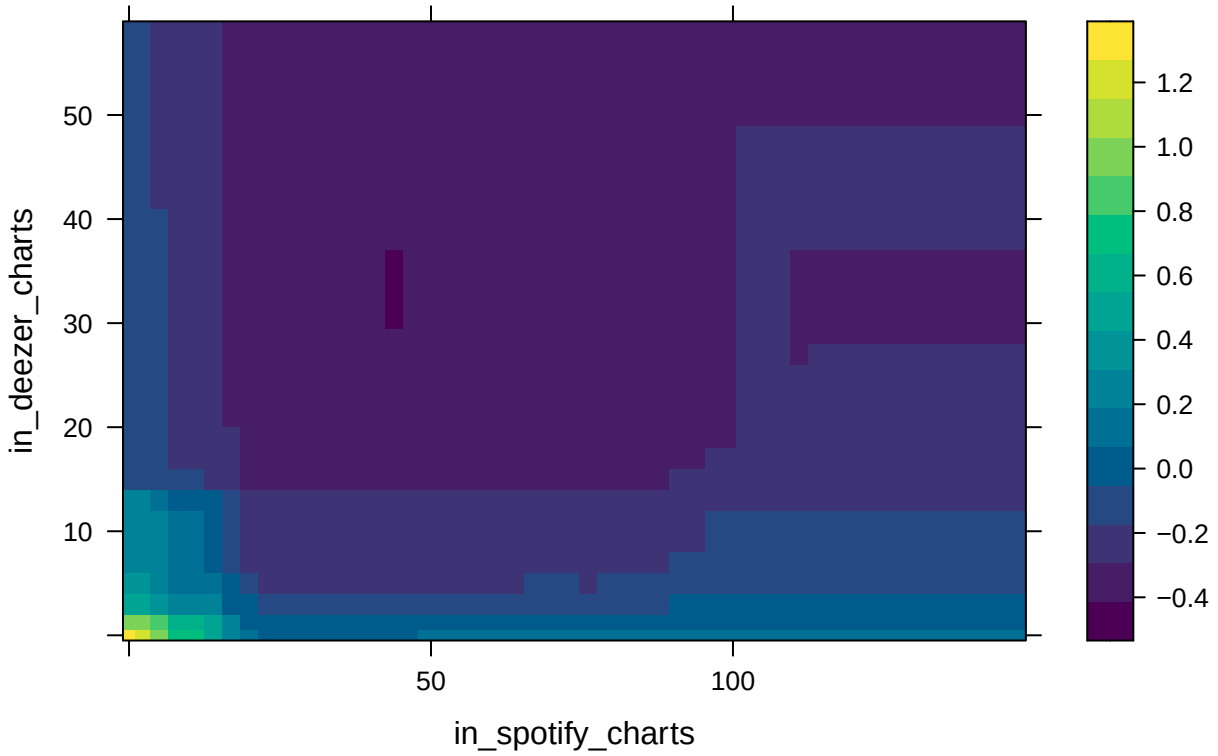
```
# reemplazar la diagonal con valores de importancia guardados en importance_sum creado previemente

for(i in 1:nrow(fit_rf)) {
  var_name <- rownames(fit_rf)[i]

  if(var_name %in% importance_sum$group) {
    real_value <- importance_sum$total_importance[importance_sum$group == var_name]
    fit_rf[var_name, var_name] <- real_value
  }
}

# Normalizar diagonal:
diag(fit_rf) <- diag(fit_rf) / sum(diag(fit_rf)) # la importancia de todas las variables suma 1
```

```
viviHeatmap(mat = fit_rf[c(1, 2, 3, 6), c(1, 2, 3, 6)],
            intPal = rev(colorspace::sequential_hcl(palette = "Purples 3", n = 100)),
            impPal = rev(colorspace::sequential_hcl(palette = "Greens 2", n = 100)),
            impLims = c(range(diag(fit_rf))),
            angle = 45)
```



La iteración entre `in_spotify_charts` e `in_deezer_charts` es aparentemente la más importante, indicando que su presencia conjunta tiene un efecto relevante sobre la predicción del modelo.

```
pdp12 <- partial(rf_model, c("in_spotify_charts", "in_deezer_charts"), which.class = "No")
plotPartial(pdp12)
```



Manteniendo las demás variables constantes, para valores bajos de `in_spotify_charts` e `in_deezer_charts` el modelo predice `no éxito`. Esto se visualiza mediante los colores amarillos, que indican una alta probabilidad de pertenecer a la clase `no éxito` (hemos seleccionado which.class = "No" en la función).

Cuando los valores son bajos en cualquiera de las dos variables, es decir, en las zonas cercanas a los ejes, predominan los tonos claros como amarillos, verdes y azules turquesa. Esto confirma que, incluso con valores altos en las posiciones en solo una de las plataformas, el modelo continúa prediciendo `no éxito`.

La transición hacia predicciones de `éxito` comienza a observarse a partir de valores superiores a 15 en `in_deezer_charts` y de 20 en `in_spotify_charts` (aprox.). A partir de estos umbrales, la intensidad de los colores se oscurece, el modelo asigna una mayor probabilidad a la clase `éxito`.

d. Evaluar la precisión de las predicciones en la muestra de test y comparar los resultados con los obtenidos con el modelo del ejercicio anterior.

```
obs <- test$hit
pred <- predict(rf_model, newdata = test)
caret::confusionMatrix(pred, obs)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  16   7
##        No   16 120
##
```

```
##                Accuracy : 0.8553
##                  95% CI : (0.7909, 0.906)
##     No Information Rate : 0.7987
##     P-Value [Acc > NIR] : 0.04238
##
##                   Kappa : 0.4972
##
##  Mcnemar's Test P-Value : 0.09529
##
##             Sensitivity : 0.5000
##             Specificity : 0.9449
##          Pos Pred Value : 0.6957
##          Neg Pred Value : 0.8824
##              Prevalence : 0.2013
##          Detection Rate : 0.1006
##    Detection Prevalence : 0.1447
##       Balanced Accuracy : 0.7224
##
##        'Positive' Class : Yes
##
```

Comparando con los árboles de decisión anteriores, este modelo presenta la especificidad más alta (0.937) entre los tres, mejorando la capacidad de identificar correctamente los no éxitos. Sin embargo, alcanza la sensibilidad más baja (0.500), detectando solo la mitad de los éxitos reales.

El coeficiente Kappa (0.482) es similar al del árbol de decisión original (0.481), pero inferior al del árbol de decisión con pesos (0.493). Además, la precisión balanceada (0.718) resulta ser la más baja de los tres modelos, reflejando el peor equilibrio entre sensibilidad y especificidad.

Ejercicio 3

Realizar la clasificación anterior empleando Boosting mediante la función `ada()` del paquete `ada`.

a. Ajustar el modelo (sin emplear el paquete `caret`) seleccionando los valores óptimos de los hiperparámetros minimizando el error OOB, considerando las posibles combinaciones de `iter = c(10, 20)`, `maxdepth = 2:3` y `nu = c(0.1, 0.5)`.

```r
library(ada)

grid <- expand.grid(
  iter = c(10, 20),
  maxdepth = 2:3,
  nu = c(0.1, 0.5)
)

oob_errors <- numeric(nrow(grid))

for (i in 1:nrow(grid)) {
  # Configurar control para rpart con la profundidad máxima
  control <- rpart.control(maxdepth = grid$maxdepth[i], cp = 0,
                           minsplit = 10, xval = 0)
```

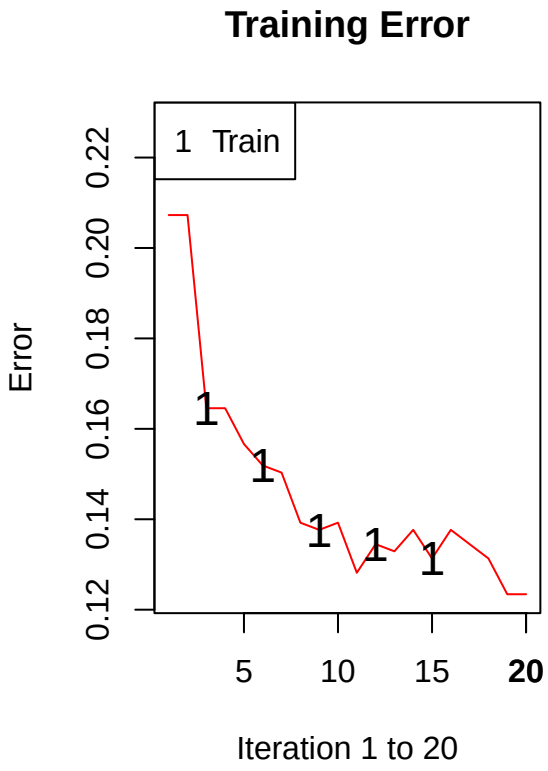
```
  model <- ada(
    hit ~ .,
    data = train,
    type = "discrete",
    control = control,
    iter = grid$iter[i],
    nu = grid$nu[i]
  )

  # Error OOB (Out-of-Bag)
  oob_errors[i] <- model$confusion[2, 1] + model$confusion[1, 2]
}

best_index <- which.min(oob_errors)
best_params <- grid[best_index, ]
best_params
```

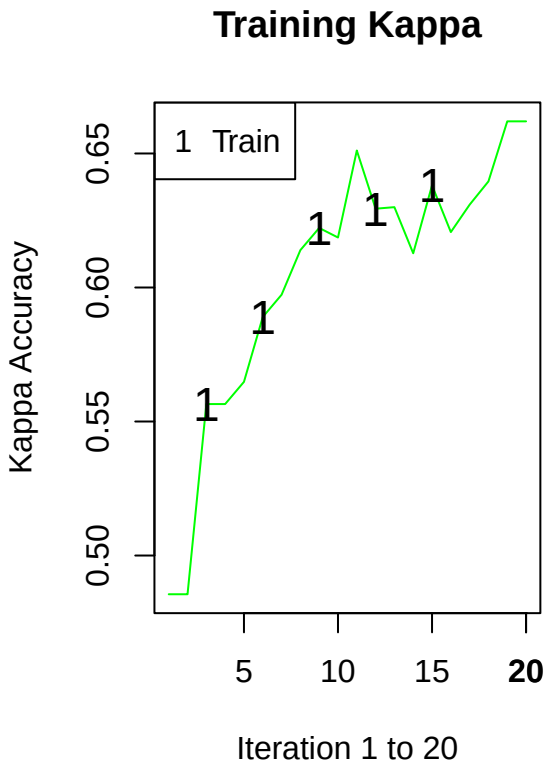
```
##   iter maxdepth  nu
## 8   20        3 0.5
```

El modelo ajustado que minimiza el error OOB se obtiene con 20 iteraciones, considerando interacciones de orden 3 entre los predictores y un parámetro de regularización lambda=0.5.

b. Representar la evolución del error OOB respecto al número de iteraciones en el modelo final.

```
set.seed(semilla)
control_final <- rpart.control(maxdepth = best_params$maxdepth, cp = 0,
                               minsplit = 10, xval = 0)

final_model <- ada(
  hit ~ .,
  data = train,
  type = "discrete",
  control = control_final,
  iter = best_params$iter,
  nu = best_params$nu
)

plot(final_model, k = 1, test = TRUE)
```

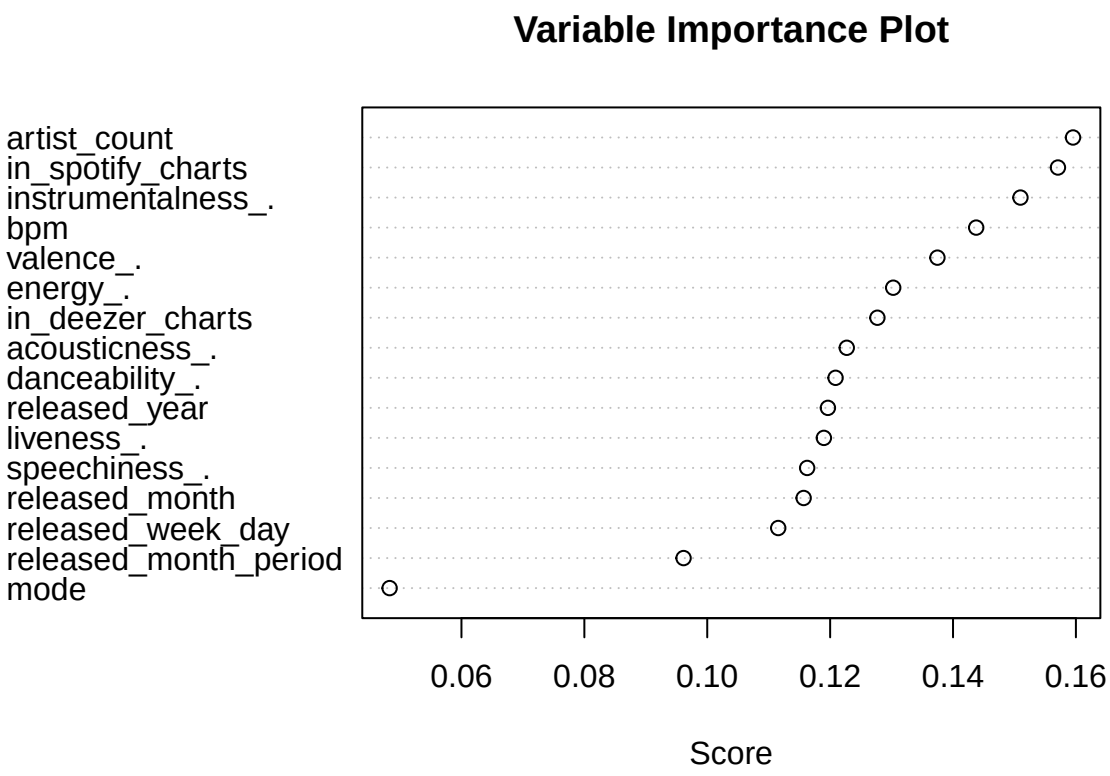




Se puede ver como el menor error de entrenamiento y el mejor índice Kappa se obtiene en la penúltima iteración del modelo, la iteración 19. No parece haberse estabilizado al llegar a la última iteración, lo que puede significar que aumentando el número de iteraciones podría obtenerse un menor error en el modelo.

c. Estudiar la importancia de los predictores del paquete `ada`.

```
varplot(final_model)
```



Como se puede ver en el gráfico, el número de artistas que participan en la canción es la variable más importante para el modelo a la hora de clasificar dicha canción como un `hit` o no, seguida por su posición en las listas de Spotify y la cantidad de contenido instrumental. Por otro lado, variables como el modo de la canción y el periodo del mes en el que se lanzó aportan poco al modelo.

d. Evaluar la precisión, de las predicciones y de las estimaciones de la probabilidad, en la muestra de test y comparar los resultados con los obtenidos con el modelo del ejercicio anterior.

```
pred <- predict(final_model, newdata = test)
p_est <- predict(final_model, newdata = test, type = "probs")
caret::confusionMatrix(pred, test$hit, positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Yes  No
##        Yes  17  11
##        No   15 116
##
##                Accuracy : 0.8365
##                  95% CI : (0.7697, 0.8903)
##     No Information Rate : 0.7987
##     P-Value [Acc > NIR] : 0.1373
##
##                   Kappa : 0.4664
```

```
##
##  Mcnemar's Test P-Value : 0.5563
##
##             Sensitivity : 0.5312
##             Specificity : 0.9134
##          Pos Pred Value : 0.6071
##          Neg Pred Value : 0.8855
##              Prevalence : 0.2013
##          Detection Rate : 0.1069
##    Detection Prevalence : 0.1761
##       Balanced Accuracy : 0.7223
##
##        'Positive' Class : Yes
##
```

La precisión global del modelo es ligeramente inferior al del modelo de bosques aleatorio (83.65% frente a 85.53%), pero de nuevo fuertemente influido por el desbalanceo entre las clases, como indica otra vez el valor de 0.7987 de *No Information Rate*. Corrigiendo este efecto, el modelo actual obtiene una precisión del 72.23%. El índice Kappa (0.4664) indica de nuevo una moderada concordancia entre las predicciones y los valores reales.

La especifidad es menor que en el modelo anterior (0.9134 frente a 0.9449) pero la sensibilidad aumenta ligeramente (0.5312 frente a 0.5), pero indican una vez más que este modelo también clasifica de forma correcta practicamente todas las canciones que no son éxitos pero cuando tiene dudas la clasifica como un "No".