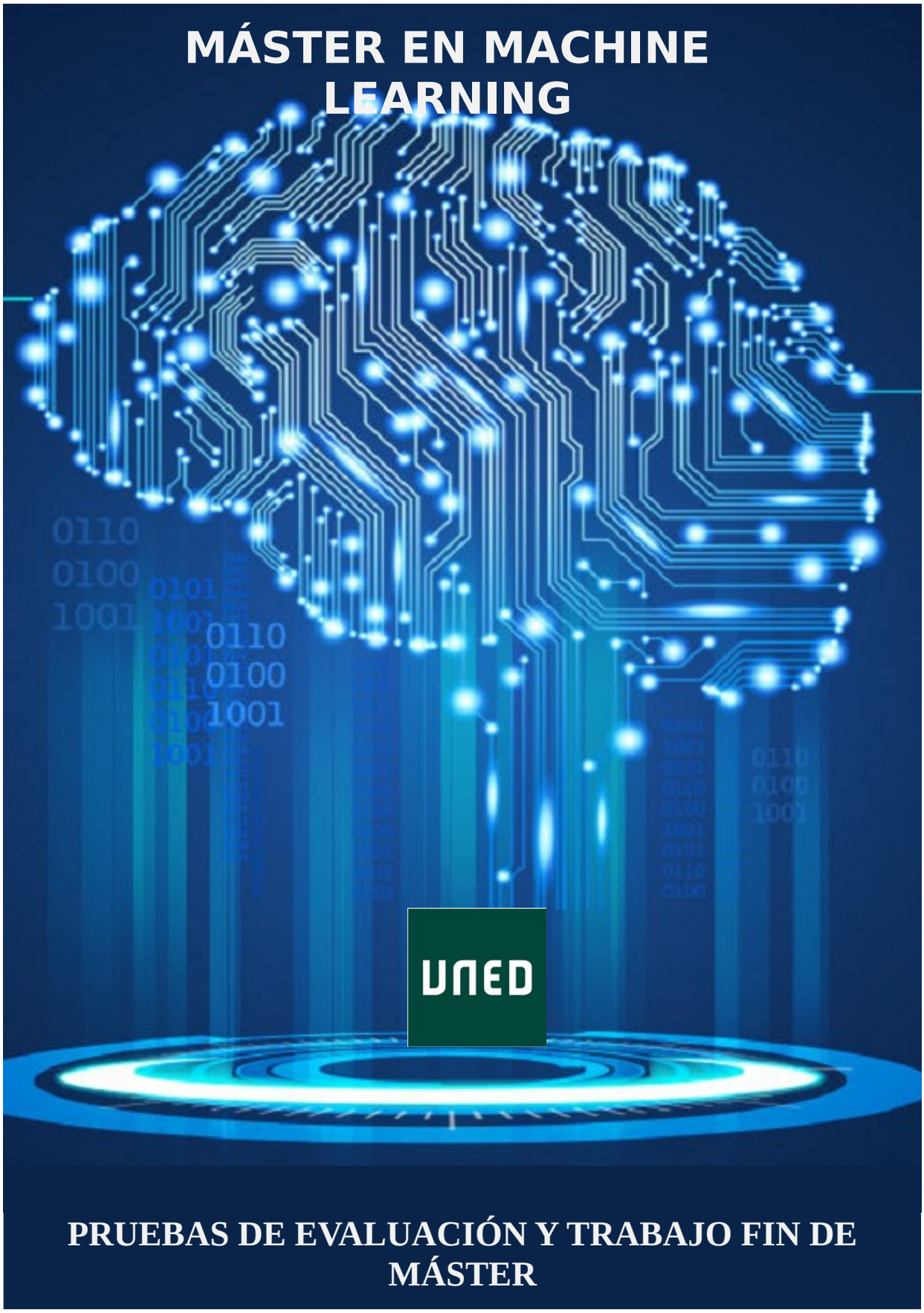


Francisco Camacho – fcamacho80@alumno.uned.es

Master UNED en Machine Learning

Pruebas de Evaluación y Trabajo Fin de Máster

Index

EVALUACIÓN TEMA 1 - INTRODUCCIÓN AL MACHINE LEARNING

1. Señale 5 aplicaciones actuales del Machine Learning.

   - Traducción automática de textos
   - Detección de fraude en operaciones financieras
   - En el sector de la salud, detección de cáncer en pacientes
   - Segmentación de clientes (marketing)
   - Detección de amenazas en ciberseguridad

2. ¿Cuáles son las 4 fases de un proceso básico de aprendizaje? Descríbalas brevemente y proponga un ejemplo de cada una de ellas.

   i) - Almacenamiento de información

   Se necesita una gran cantidad de información que sirva de base factual para los “razonamientos” del modelo. Un ejemplo puede ser toda la información que recogen los sensores de la red meteorológica nacional.

   ii) - Abstracción

   Consiste en transladar la información contenida en los datos en un concepto/idea genérico: el modelo.
   Un ejemplo bastante típico es un modelo matemático basado en ecuaciones.

   iii) – Generalización

   Una vez que se tiene información disponible y un modelo, se trata de darle una utilidad a todo eso, poder aplicar esa información/conocimiento a otros datos nuevos para poder ejecutar alguna acción. Esa capacidad de entender datos nuevos, desconocidos hasta ahora, es lo que se llama generalización.
   Un ejemplo puede ser un algoritmo que ha sido entrenado para detectar imágenes de vehículos de motor. Si se le presentan nuevas imágenes de automóviles o camiones (o bicicletas) debe detectar que sí son vehículos a motor (o no).

   iv) – Evaluación

   Se trata de medir el grado de precisión del algoritmo de aprendizaje, qué nivel de exactitud tiene la información que proporciona.
   Se puede utilizar por ejemplo en regresión lineal la función que calcula la raiz del error cuadrático medio (en inglés RMSE “Root Mean Squared Error").

3. Indique 3 eventos que podrían generar ruido en los datos.

   - errores de medida en los sensores utilizados para recoger los datos
   - datos corruptos almacenados o valores que faltan
   - etiquetar los datos de manera incorrecta

4. Enumere y describa brevemente los 5 pasos necesarios para implementar un algoritmo de Machine Learning.

   i) - Adquirir los datos

   Obtener la información base que va a ser utilizada por el algoritmo de Machine Learning.

   ii) - Exploración de datos y preparación

   Puede ser que la información recopilada tenga errores, falten valores ... que no todos sus atributos sean necesarios ... Se trata de entender y transformar la información de manera que pueda ser utilizada de la manera más óptima posible por el algoritmo.

   Iii) - Entrenar el modelo

   Se trata de alimentar al modelo con los datos preparados para que pueda ajustar sus parámetros.

   iv) - Evaluar los resultados del modelo

   Se trata de medir la precisión del modelo. Normalmente se utilizan conjuntos de datos diferentes para entrenar y para evaluar un modelo.

   v) - Mejorar el modelo

   Dependendiendo de los resultados del paso anterior, quizá sea necesario refinar el modelo con pasos adicionales o técnicas más avanzadas, proporcionarle más datos ... Puede darse el caso también por supuesto de tener que descartar el modelo por completo.

5. En el algoritmo del traductor automático de Google, ¿cuáles serían las unidades de observación, los ejemplos (*examples*) y las características (*features*)?

   Unidades de observación: las frases en cada idioma
   Ejemplos: cada frase concreta
   Características: las propias de cada frase (si es principal, subordinada, voz pasiva o activa, el sujeto, el verbo ..)
   Imagino que también las características de cada una de las palabras que la componen (si es un verbo, su persona, tiempo y número .., si es un adjetivo, si es un nombre su género y número ..)

6. Señale los tipos de algoritmos de Machine Learning y los diferentes métodos de entrenamiento asociados a ellos.

   - Modelos predictivos.

Predicen/calculan un valor (no necesariamente en el futuro). Se trata de predecir un valor no conocido en base a valores ya conocidos. Este valor a predecir puede ser numérico, o una clase o categoría, (sí/no, o un nivel dentro de una escala de valores nominales: leve, medio, grave …)

El método de entrenamiento es supervisado: se llama así porque se dispone de información sobre los datos (están etiquetados), y estos datos le dan “pistas“ al algoritmo sobre qué resultados debe obtener.

   - Modelos descriptivos.

Sirven para analizar y describir la información contenida en los datos.
El método de entrenamiento es no supervisado: se llama así porque no se dispone de información sobre los datos. No se trata de predecir/calcular un valor desconocido hasta ahora, sino de descubrir las relaciones entre las diferentes entidades existentes (por ejemplo, segmentar clientes según su comportamiento, ver qué productos suelen ser comprados a la vez …)

   - Meta-aprendizaje/aprendizaje reforzado.

El algoritmo recibe una recompensa dependiendo de su respuesta. De esta manera irá corrigiendo y mejorando las acciones que propone.

EVALUACIÓN TEMA 2 - GESTIÓN Y EXPLORACIÓN DE DATOS CON R

1 Análisis exploratorio de datos

1. Utilizando cualquiera de los *datasets* incluidos en la librería ISLR, realice un análisis exploratorio del *dataset* elegido, incluyendo una visualización gráfica de los datos, comentando los principales resultados obtenidos. Para ello, puede utilizar todas las funciones que se explican en el capítulo 2 del texto base, así como otras que desee el alumno.
   Se puede encontrar una descripción detallada de los *datasets* de ISLR en https://cran.r-project.org/web/packages/ISLR/ISLR.pdf

1 Cargar datos del paquete ISLR

```
install.packages("ISLR")

## The following package(s) will be installed:
## - ISLR [1.4]
## These packages will be installed into
"~/git/machineLearningUNED/renv/library/linux-debian-bookworm/R-4.4/x86_64-pc-
linux-gnu".
## # Installing packages
------------------------------------------------------
## - Installing ISLR ...                            OK [linked from cache]
## Successfully installed 1 package in 9.2 milliseconds.

library("ISLR")
```

2 Elección de un dataset para examinar su contenido

Elegimos el dataset Credit. Vemos su estructura y resumen estadístico:

```
#Structure
str(Credit)
##'data.frame':   400   obs. of 12 variables:
## $ ID       : int  1 2 3 4 5 6 7 8 9 10 ...
## $ Income   : num  14.9 106 104.6 148.9 55.9 ...
## $ Limit    : int  3606 6645 7075 9504 4897 8047 3388 7114 3300 6819 ...
## $ Rating   : int  283 483 514 681 357 569 259 512 266 491 ...
## $ Cards    : int  2 3 4 3 2 4 2 2 5 3 ...
## $ Age      : int  34 82 71 36 68 77 37 87 66 41 ...
## $ Education: int  11 15 11 11 16 10 12 9 13 19 ...
## $ Gender   : Factor w/ 2 levels " Male","Female": 1 2 1 2 1 1 2 1 2 2 ...
## $ Student  : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 1 1 1 1 2 ...
## $ Married  : Factor w/ 2 levels "No","Yes": 2 2 1 1 2 1 1 1 1 2 ...
## $ Ethnicity: Factor w/ 3 levels "African American",..: 3 2 2 2 3 3 1 2 3 ...
## $ Balance  : int  333 903 580 964 331 1151 203 872 279 1350 ...
```

Este data frame contiene 400 observaciones, con 12 variables cada una de ellas (en la documentación se mencionan 10.000 y 4 variables ..) Eliminamos el ID, mostramos algunos registros para ver su aspecto, y finalmente el resumen estadístico con “summary”.

```
credit <- Credit[-1]

head(credit,10)
```

```
##     Income Limit Rating Cards Age Education Gender Student Married
## 1   14.891  3606    283     2  34        11   Male      No     Yes
## 2  106.025  6645    483     3  82        15 Female     Yes     Yes
## 3  104.593  7075    514     4  71        11   Male      No      No
## 4  148.924  9504    681     3  36        11 Female      No      No
## 5   55.882  4897    357     2  68        16   Male      No     Yes
## 6   80.180  8047    569     4  77        10   Male      No      No
## 7   20.996  3388    259     2  37        12 Female      No      No
## 8   71.408  7114    512     2  87         9   Male      No      No
## 9   15.125  3300    266     5  66        13 Female      No      No
## 10  71.061  6819    491     3  41        19 Female     Yes     Yes
##           Ethnicity Balance
## 1         Caucasian     333
## 2             Asian     903
## 3             Asian     580
## 4             Asian     964
## 5         Caucasian     331
## 6         Caucasian    1151
## 7  African American     203
## 8             Asian     872
## 9         Caucasian     279
## 10 African American    1350
```

```
tail(credit,10)
```

```
##      Income Limit Rating Cards Age Education Gender Student Married
## 391 135.118 10578    747     3  81        15 Female      No     Yes
## 392  73.327  6555    472     2  43        15 Female      No      No
## 393  25.974  2308    196     2  24        10   Male      No      No
## 394  17.316  1335    138     2  65        13   Male      No      No
## 395  49.794  5758    410     4  40         8   Male      No      No
## 396  12.096  4100    307     3  32        13   Male      No     Yes
## 397  13.364  3838    296     5  65        17   Male      No      No
## 398  57.872  4171    321     5  67        12 Female      No     Yes
## 399  37.728  2525    192     1  44        13   Male      No     Yes
## 400  18.701  5524    415     5  64         7 Female      No      No
##            Ethnicity Balance
## 391            Asian    1393
## 392        Caucasian     721
## 393            Asian       0
## 394 African American       0
## 395        Caucasian     734
## 396        Caucasian     560
## 397 African American     480
## 398        Caucasian     138
## 399        Caucasian       0
## 400            Asian     966
```

```
#Summary
summary(credit)
```

```
##      Income           Limit           Rating          Cards      
##  Min.   : 10.35   Min.   :  855   Min.   : 93.0   Min.   :1.000  
##  1st Qu.: 21.01   1st Qu.: 3088   1st Qu.:247.2   1st Qu.:2.000  
##  Median : 33.12   Median : 4622   Median :344.0   Median :3.000  
##  Mean   : 45.22   Mean   : 4736   Mean   :354.9   Mean   :2.958  
##  3rd Qu.: 57.47   3rd Qu.: 5873   3rd Qu.:437.2   3rd Qu.:4.000  
##  Max.   :186.63   Max.   :13913   Max.   :982.0   Max.   :9.000  

##       Age          Education        Gender    Student   Married  
##  Min.   :23.00   Min.   : 5.00    Male :193   No :360   No :155  
##  1st Qu.:41.75   1st Qu.:11.00   Female:207   Yes: 40   Yes:245  
##  Median :56.00   Median :14.00                                   
##  Mean   :55.67   Mean   :13.45                                   
##  3rd Qu.:70.00   3rd Qu.:16.00                                   
##  Max.   :98.00   Max.   :20.00                                   

##             Ethnicity      Balance       
##  African American: 99   Min.   :   0.00  
##  Asian           :102   1st Qu.:  68.75  
##  Caucasian       :199   Median : 459.50  
##                         Mean   : 520.01  
##                         3rd Qu.: 863.00  
##                         Max.   :1999.00
```

Todos los registros están completos (no hay NAs, ni nulos). Algunos balances están a 0 (según la documentación es un valor promedio). También me llama mucho la atención la columna “Etnia” …

A la función “summary” se le puede pasar un vector con las variables a analizar, si se quiere poner el foco en algunas de ellas. Elijo tres que considero bastante relevantes en este dataset:

```
#Summary of Income, Rating and Balance
summary(credit[c("Income", "Rating", "Balance")])
```

```
##      Income           Rating          Balance       
##  Min.   : 10.35   Min.   : 93.0   Min.   :   0.00  
##  1st Qu.: 21.01   1st Qu.:247.2   1st Qu.:  68.75  
##  Median : 33.12   Median :344.0   Median : 459.50  
##  Mean   : 45.22   Mean   :354.9   Mean   : 520.01  
##  3rd Qu.: 57.47   3rd Qu.:437.2   3rd Qu.: 863.00  
##  Max.   :186.63   Max.   :982.0   Max.   :1999.00
```

No me parece que la media y la mediana estén muy separadas para estas variables (la media se ve afectada mucho más por valores extremos, así que entiendo que no hay muchos valores de esa clase).

3 Cuantiles y rango intercuartil (IQR: interquartile range)

La diferencia entre el cuartil 1 (Q1) y el cuartil 3 (Q3) es conocido como “rango intercuartil”, y es de interés porque representa una medida simple de la dispersión de los datos.

```
#IQR Income
IQR(credit$Income)
## [1] 36.4635

#IQR Rating
IQR(credit$Rating)
## [1] 190

#IQR Balance
IQR(credit$Balance)
## [1] 794.25
```

Aquí se puede ver por ejemplo que el 50% de los ingresos están en el rango que va desde los 21.000$ del Q1 a los 57.000 del Q3 (IQR=36.464$).

La función que nos devuelve los cuantiles es “quantile”. Sin parámetros nos devuelve los mismos valores que la función “summary”. Si por ejemplo queremos averiguar los ingresos del 10% de la población con menores y mayores ingresos, debemos llamar a la función quantile con estos parámetros:

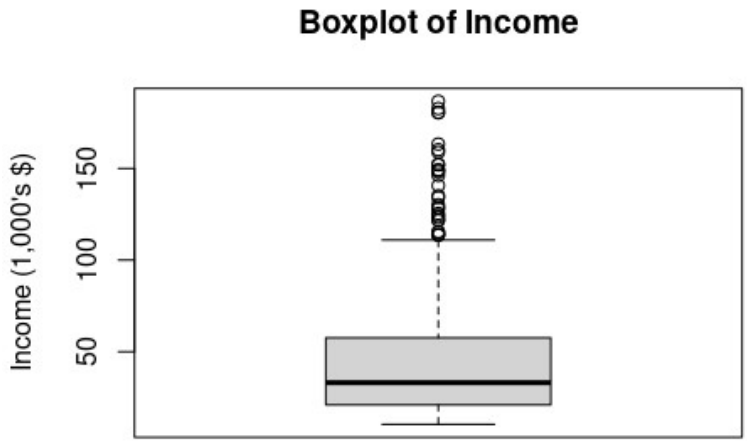
```
quantile(credit$Income, probs = c(0.1, 0.9))
##     10%     90%
## 14.5834 92.4513
```

El 10% de la población tiene ingresos inferiores a 14.5583$, y el 10% con más ingresos gana más de 92.451$.

4 Visualización de variables numéricas con diagramas de cajas (“boxplots”)

Los diagramas de caja permiten una visualización de los datos que puede ayudar a diagnosticar problemas que afecten al rendimiento de los algoritmos. Gracias a ellos se puede ver a simple vista la mediana, el rango de los valores, los valores atípicos … (también los cuantiles obtenidos al llamar a la función “summary”).

```
#Boxplot for variable income
boxplot(credit$Income, main = "Boxplot of Income", ylab = "Income (1,000's $)")
```



Al analizar la variable ingresos con la ayuda de su diagrama de cajas, se puede ver que la mediana está más cerca del Q1 que del Q3, y que sí hay bastantes valores atípicos.

Estos valores atípicos (superiores) se suelen calcular de la siguiente manera:

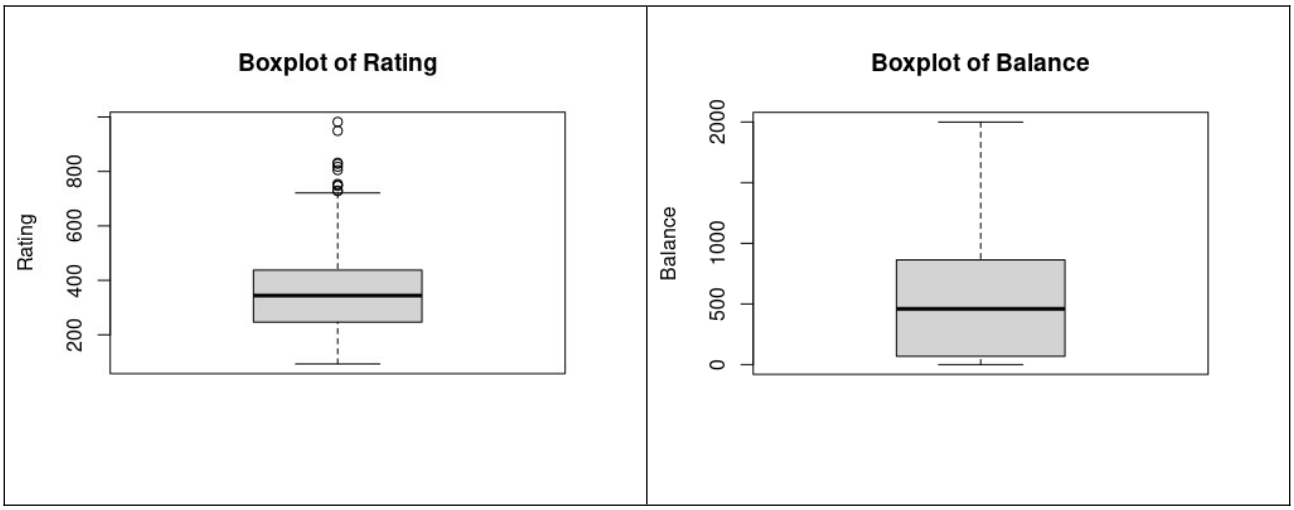
```
    Q3 + 1.5 * IQR
57470 + 1.5 * 36464 = 112.000
```

Todos los salarios por encima de la línea que marca los 112.000$, son considerados por tanto valores atípicos (“outliers”).

Si analizamos las otras dos variables “rating” y balance medio en la cuenta:

```
#Boxplot for variables rating and balance
boxplot(credit$Rating, main = "Boxplot of Rating", ylab = "Rating")
boxplot(credit$Balance, main = "Boxplot of Balance", ylab = "Balance")
```

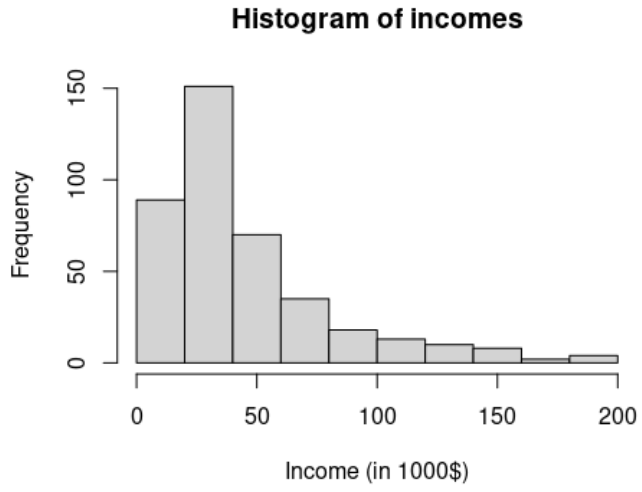
(Gráficos en la siguiente página)



Vemos que el diagrama de cajas del rating es bastante parecido al de los ingresos, mientras que el del balance no muestra valores atípicos (debido probablemente a que es un valor medio, y a que las tarjetas de crédito suelen tener unos límites estándares).

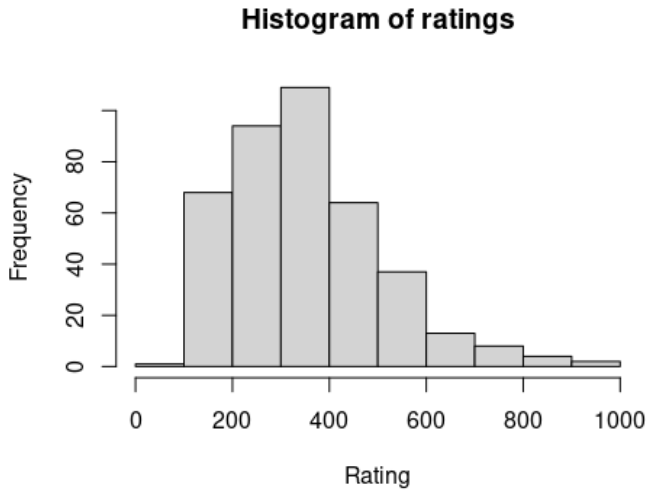
5 Visualización de variables numéricas con histogramas

```
hist(credit$Income, main = "Histogram of incomes", xlab = "Income (in 1000$)")
```



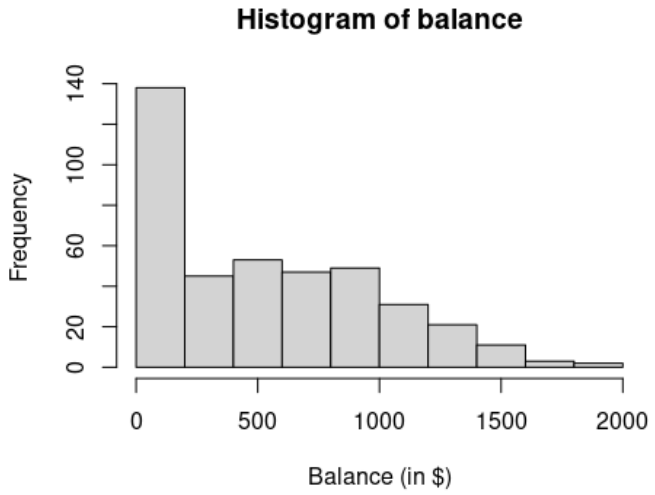
Se puede ver claramente como en los ingresos hay un sesgo hacia la derecha (los valores en el extremo derecho están más dispersos).

```r
hist(credit$Rating, main = "Histogram of ratings", xlab = "Rating")
```



Para el rating también hay sesgo a la derecha, pero es menos pronunciado, los valores se agrupan algo más por el centro.

```r
hist(credit$Balance, main = "Histogram of balance", xlab = "Balance (in $)")
```



También se da el sesgo a la derecha para el balance. Entiendo que las tres variables están muy relacionadas, lógicamente.

6 Varianza y desviación estándar

La varianza y la desviación estándard proporcionan información sobre la dispersión de una variable. Una varianza con un valor elevado indica una dispersión elevada. La desviación estándard indica la diferencia de cada valor con la media. Estas variables están intimamente relacionadas (la varianza es el cuadrado de la desviación estándard).

```
# variance and standard deviation of income
var(credit$Income)
## [1] 1242.159
sd(credit$Income)
## [1] 35.24427

# variance and standard deviation of rating
var(credit$Rating)
## [1] 23939.56
sd(credit$Rating)
## [1] 154.7241

# variance and standard deviation of balance
var(credit$Balance)
## [1] 211378.2
sd(credit$Balance)
## [1] 459.7589
```

La varianza del balance, me parece completamente exagerada. Solo puedo deducir que se debe a la presencia de muchas personas que tienen una media de 0 en sus balances.

7 Exploración de variables categóricas

En el dataset Credit hay 4 variables categóricas:género, “etnia”, estado civil, y “estado estudiantil” (la educación viene en años cursados, no por el grado/título obtenido, así que es numérica en este caso).

Para analizar variables categóricas se suelen utilizar tablas en lugar de funciones estadísticas:

```
# Gender
table(credit$Gender)
##
##   Male Female
##    193    207
```

```
# Married
table(credit$Married)
##
##  No Yes
## 155 245

# Ethnicity
table(credit$Ethnicity)
##
## African American            Asian        Caucasian
##               99              102              199
```

Si se quiere ver el porcentaje de observaciones de cada categoría:

```
# Ethnicity
prop.table(table(credit$Ethnicity))*100
##
## African American            Asian        Caucasian
##            24.75            25.50            49.75
```

2 Análisis de relación entre variables en el dataset Wage

2. Tomando como referencia el *dataset* Wage incluido en la librería ISLR, analice visualmente mediante diagramas de dispersión la relación entre la variable *wage* (última variable del dataset) y otras variables como *age, maritl, race, education, etc.*

1 Cargar datos del paquete ISLR

```
install.packages("ISLR")
## The following package(s) will be installed:
## - ISLR [1.4]
## These packages will be installed into
"~/git/machineLearningUNED/renv/library/linux-debian-bookworm/R-4.4/x86_64-pc-
linux-gnu".
##
## # Installing packages
------------------------------------------------------------
## - Installing ISLR ...                            OK [linked from cache]
## Successfully installed 1 package in 8.3 milliseconds.
library("ISLR")
#install.packages("gmodels")
#library("gmodels")
```

2 Examinar su contenido (estructura y algunos registros) del dataset Wage

```
#Structure
str(Wage)
## 'data.frame':    3000 obs. of  11 variables:
##  $ year    : int  2006 2004 2003 2003 2005 2008 2009 2008 2006 2004 ...
##  $ age     : int  18 24 45 43 50 54 44 30 41 52 ...
##  $ maritl  : Factor w/ 5 levels "1. Never Married",..:1 1 2 2 4 2 2 1 1 2 ...
##  $ race    : Factor w/ 4 levels "1. White","2. Black",..: 1 1 1 3 1 1 4 3 2 1
...
##  $ education : Factor w/ 5 levels "1. < HS Grad",..: 1 4 3 4 2 4 3 3 3 2 ...
##  $ region    : Factor w/ 9 levels "1. New England",..: 2 2 2 2 2 2 2 2 2 2...
##  $ jobclass  : Factor w/ 2 levels "1. Industrial",..: 1 2 1 2 2 2 1 2 2 2 ...
##  $ health    : Factor w/ 2 levels "1. <=Good","2. >=Very Good": 1 2 1 2 1 2 2
1 2 2 ...
##  $ health_ins: Factor w/ 2 levels "1. Yes","2. No": 2 2 1 1 1 1 1 1 1 1 ...
##  $ logwage   : num  4.32 4.26 4.88 5.04 4.32 ...
##  $ wage      : num  75 70.5 131 154.7 75 …

#Have a look at some data
head(Wage,10)
##        year age         maritl       race       education         region
## 231655 2006  18 1. Never Married 1. White 1. < HS Grad    2. Middle Atlantic
## 86582  2004  24 1. Never Married 1. White 4. College Grad 2. Middle Atlantic
## 161300 2003  45 2. Married       1. White 3. Some College 2. Middle Atlantic
## 155159 2003  43 2. Married       3. Asian 4. College Grad 2. Middle Atlantic
## 11443  2005  50 4. Divorced      1. White 2. HS Grad      2. Middle Atlantic
## 376662 2008  54 2. Married       1. White 4. College Grad 2. Middle Atlantic
## 450601 2009  44 2. Married       4. Other 3. Some College 2. Middle Atlantic
## 377954 2008  30 1. Never Married 3. Asian 3. Some College 2. Middle Atlantic
## 228963 2006  41 1. Never Married 2. Black 3. Some College 2. Middle Atlantic
## 81404  2004  52 2. Married       1. White 2. HS Grad      2. Middle Atlantic
##          jobclass           health    health_ins  logwage   wage
## 231655  1. Industrial    1. <=Good       2. No   4.318063  75.04315
## 86582   2. Information   2. >=Very Good  2. No   4.255273  70.47602
## 161300  1. Industrial    1. <=Good       1. Yes  4.875061 130.98218
## 155159  2. Information   2. >=Very Good  1. Yes  5.041393 154.68529
## 11443   2. Information   1. <=Good       1. Yes  4.318063  75.04315
## 376662  2. Information   2. >=Very Good  1. Yes  4.845098 127.11574
## 450601  1. Industrial    2. >=Very Good  1. Yes  5.133021 169.52854
## 377954  2. Information   1. <=Good       1. Yes  4.716003 111.72085
## 228963  2. Information   2. >=Very Good  1. Yes  4.778151 118.88436
## 81404   2. Information   2. >=Very Good  1. Yes  4.857332 128.68049
```

3 Relación entre edad y salario

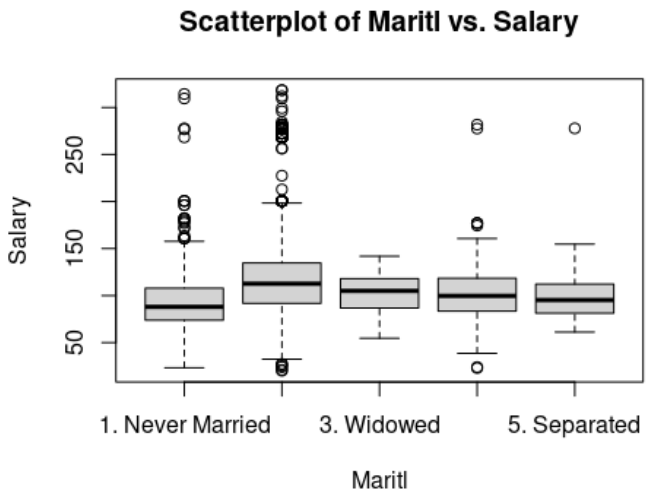
```
plot(x = Wage$age, y = Wage$wage, main = "Scatterplot of Age vs. Salary",
xlab = "Age",
ylab = "Salary")
```

No se aprecia una influencia importante de la edad en el salario (se ve que los trabajadores más jóvenes tienen salarios más bajos, a partir de 65 años hay menos datos ...)

El resto de variables que pueden influir en el salario son categóricas. Si utilizamos el mismo diagrama para intentar relacionar el estado civil con el salario, se obtiene lo siguiente:

4 Relación entre estado civil y salario

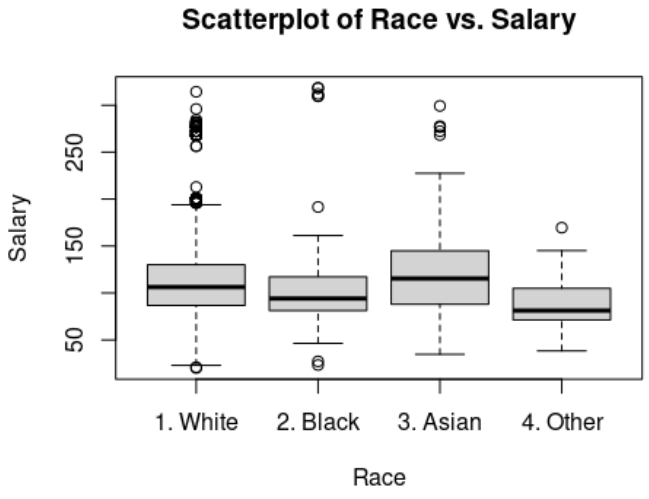
```
plot(x = Wage$maritl, y = Wage$wage, main = "Scatterplot of Maritl vs.
Salary",
xlab = "Maritl",
ylab = "Salary")
```



Se puede ver que los casados tienen mejores sueldos, y los nunca casados los peores. Esto es coherente con la relación entre edad y salario: hay en general menos casados entre los hombres más jóvenes.

5 Relación entre raza y salario

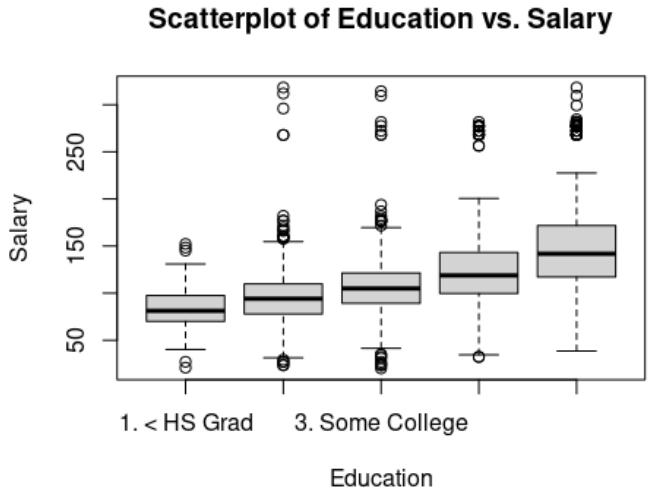
```
plot(x = Wage$race, y = Wage$wage, main = "Scatterplot of Race vs. Salary",
xlab = "Race",
ylab = "Salary")
```



Se puede apreciar como en este dataset los trabajadores de origen asiático tienen en general mejores sueldos, pero como en cambio entre las personas blancas se dan más casos de salarios atípicamente altos.

6 Relación entre educación y salario

```
plot(x = Wage$education, y = Wage$wage, main = "Scatterplot of Education vs.
Salary",
xlab = "Education",
ylab = "Salary")
```



Aquí se puede ver claramente una correlación positiva entre educación y salario: a mejor educación, mejor salario.

EVALUACIÓN TEMA 3 - LAZY LEARNING. CLASIFICACIÓN USANDO NEAREST NEIGHBORS

Nota: El alumno deberá presentar todo el código utilizado para resolver el ejercicio.

La base de datos Diabetes in Pima Indian Women, incluida en la librería MASS, contiene una muestra de 532 pacientes en edad adulta utilizada para realizar un estudio sobre la diabetes. A este respecto, puede encontrar una descripción detallada de este *dataset* en https://cran.r-project.org/web/packages/MASS/MASS.pdf
Utilizando los datos contenidos en dicho *dataset*, elabore un modelo basado en el algoritmo kNN para determinar el valor de la variable *type* (que toma valor Yes si el paciente es diabético y No en caso contrario) en base al resto de variables incluidas. [Importante: observe que para conseguir los 532 registros de pacientes debe agregar los datasets Pima.tr y Pima.te usando la función `rbind`)
Para elaborar el modelo, deberá usar un 70% de la muestra como conjunto de entrenamiento, dejando el resto para validar el modelo. Analice las matrices de confusión para diferentes valores del parámetro *k*, y comente en detalle todos los resultados obtenidos.

1 Cargar datos del paquete MASS

(Y también otros paquetes para funciones diversas).

```
#install.packages("MASS")
library(MASS)
#install.packages("gmodels")   # cross tables
library(gmodels)
#install.packages("caret")     # data partitioning, confusion matrix
library(caret)
## Lade nötiges Paket: ggplot2
## Lade nötiges Paket: lattice
#install.packages("class")     # knn algorithm
library(class)
```

2 Paso 2: Explorar y preparar los datos

Examinar contenido (estructura y algunos registros) de los datasets Pima.tr y Pima.te:

```
#Structure
str(Pima.tr)

## 'data.frame':    200 obs. of  8 variables:
##  $ npreg: int  5 7 5 0 0 5 3 1 3 2 ...
##  $ glu  : int  86 195 77 165 107 97 83 193 142 128 ...
##  $ bp   : int  68 70 82 76 60 76 58 50 80 78 ...
##  $ skin : int  28 33 41 43 25 27 31 16 15 37 ...
##  $ bmi  : num  30.2 25.1 35.8 47.9 26.4 35.6 34.3 25.9 32.4 43.3 ...
##  $ ped  : num  0.364 0.163 0.156 0.259 0.133 ...
##  $ age  : int  24 55 35 26 23 52 25 24 63 31 ...
##  $ type : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 1 1 2 ...

str(Pima.te)

## 'data.frame':    332 obs. of  8 variables:
##  $ npreg: int  6 1 1 3 2 5 0 1 3 9 ...
##  $ glu  : int  148 85 89 78 197 166 118 103 126 119 ...
##  $ bp   : int  72 66 66 50 70 72 84 30 88 80 ...
##  $ skin : int  35 29 23 32 45 19 47 38 41 35 ...
##  $ bmi  : num  33.6 26.6 28.1 31 30.5 25.8 45.8 43.3 39.3 29 ...
##  $ ped  : num  0.627 0.351 0.167 0.248 0.158 0.587 0.551 0.183 0.704
0.263 ...
##  $ age  : int  50 31 21 26 53 51 31 33 27 29 ...
##  $ type : Factor w/ 2 levels "No","Yes": 2 1 1 2 2 2 2 1 1 2 ...

#Have a look at some data
head(Pima.tr,5)

##   npreg glu bp skin  bmi   ped age type
## 1     5  86 68   28 30.2 0.364  24   No
## 2     7 195 70   33 25.1 0.163  55  Yes
## 3     5  77 82   41 35.8 0.156  35   No
## 4     0 165 76   43 47.9 0.259  26   No
## 5     0 107 60   25 26.4 0.133  23   No

head(Pima.te,5)

##   npreg glu bp skin  bmi   ped age type
## 1     6 148 72   35 33.6 0.627  50  Yes
## 2     1  85 66   29 26.6 0.351  31   No
## 3     1  89 66   23 28.1 0.167  21   No
## 4     3  78 50   32 31.0 0.248  26  Yes
## 5     2 197 70   45 30.5 0.158  53  Yes
```

Se comprueba que los datasets tienen exactamente la misma estructura, así que podemos unirlos en un único dataset como se pide:

```
#Join datasets
pima <-  rbind(Pima.tr,Pima.te)

#write.csv(pima,file='Chapter03/pima.csv',na='') <- export data to be used
lately with Python & scikit-learn


str(pima)
```

```
## 'data.frame':    532 obs. of  8 variables:
##  $ npreg: int  5 7 5 0 0 5 3 1 3 2 ...
##  $ glu  : int  86 195 77 165 107 97 83 193 142 128 ...
##  $ bp   : int  68 70 82 76 60 76 58 50 80 78 ...
##  $ skin : int  28 33 41 43 25 27 31 16 15 37 ...
##  $ bmi  : num  30.2 25.1 35.8 47.9 26.4 35.6 34.3 25.9 32.4 43.3 ...
##  $ ped  : num  0.364 0.163 0.156 0.259 0.133 ...
##  $ age  : int  24 55 35 26 23 52 25 24 63 31 ...
##  $ type : Factor w/ 2 levels "No","Yes": 1 2 1 1 1 2 1 1 1 2 ...
```

Variable dependiente

El atributo “type” es de especial interés, ya que es la variable que se quiere predecir.

Como dicho nombre “type” es demasiado genérico, le damos otro nombre más semántico (“diagnóstico”). También aprovechamos para cambiar el nombre de los niveles (a “positivo” y “negativo”)

```
#Rename target variable and its levels
colnames(pima)[8] = "diagnosis"
levels(pima$diagnosis) <- c("Negative", "Positive")
```

Resumen estadístico de la variable dependiente

```
#Statistics of type/diagnosis
table(pima$diagnosis)
##
## Negative Positive
##      355      177
round(prop.table(table(pima$diagnosis))*100, digits = 2)
##
## Negative Positive
##    66.73    33.27
```

Normalización de los datos

El resultado del algoritmo KNNs depende en gran medida del modo en que se calculan las distancias entre las observaciones. Una variable con un rango de valores más grande, tendrá mucha mayor influencia que una con un rango más acotado. Es por tanto conveniente proceder a normalizar las variables numéricas.

Resumen estadístico antes de normalizar:

```
#Summary before normalization
summary(pima)
```

```
##      npreg             glu               bp              skin      
##  Min.   : 0.000   Min.   : 56.00   Min.   : 24.00   Min.   : 7.00  
##  1st Qu.: 1.000   1st Qu.: 98.75   1st Qu.: 64.00   1st Qu.:22.00  
##  Median : 2.000   Median :115.00   Median : 72.00   Median :29.00  
##  Mean   : 3.517   Mean   :121.03   Mean   : 71.51   Mean   :29.18  
##  3rd Qu.: 5.000   3rd Qu.:141.25   3rd Qu.: 80.00   3rd Qu.:36.00  
##  Max.   :17.000   Max.   :199.00   Max.   :110.00   Max.   :99.00  

##       bmi              ped              age           diagnosis  
##  Min.   :18.20   Min.   :0.0850   Min.   :21.00   Negative:355  
##  1st Qu.:27.88   1st Qu.:0.2587   1st Qu.:23.00   Positive:177  
##  Median :32.80   Median :0.4160   Median :28.00                 
##  Mean   :32.89   Mean   :0.5030   Mean   :31.61                 
##  3rd Qu.:36.90   3rd Qu.:0.6585   3rd Qu.:38.00                 
##  Max.   :67.10   Max.   :2.4200   Max.   :81.00
```

Normalización (“min-max scaling”):

```
# Normalization (with the help of package caret):
# The preProcess() function scales the values to a range of 0 to 1 using method
= c('range') as an argument.
# The predict() method applies the actions of the preProcess() function on the
entire data
process <- preProcess(pima, method=c("range"))
pima_norm <- predict(process, pima)
```

Resumen estadístico después de normalizar:

```
#Summary after normalization
summary(pima_norm)
```

```
##      npreg              glu               bp              skin        
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.05882   1st Qu.:0.2990   1st Qu.:0.4651   1st Qu.:0.1630  
##  Median :0.11765   Median :0.4126   Median :0.5581   Median :0.2391  
##  Mean   :0.20688   Mean   :0.4548   Mean   :0.5524   Mean   :0.2411  
##  3rd Qu.:0.29412   3rd Qu.:0.5962   3rd Qu.:0.6512   3rd Qu.:0.3152  
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

##       bmi              ped              age            diagnosis  
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Negative:355  
##  1st Qu.:0.1979   1st Qu.:0.07441   1st Qu.:0.03333   Positive:177  
##  Median :0.2986   Median :0.14176   Median :0.11667                 
##  Mean   :0.3004   Mean   :0.17900   Mean   :0.17691                 
##  3rd Qu.:0.3824   3rd Qu.:0.24561   3rd Qu.:0.28333                 
##  Max.   :1.0000   Max.   :1.00000   Max.   :1.00000
```

Se comprueba que todos los valores están ahora entre 0 y 1.

```
#Summary after normalization with aux. function
#
# normalize <- function(x) {
#   return ((x - min(x)) / (max(x) - min(x)))
# }
# pima_norm_2 <- as.data.frame(lapply(pima[1:7], normalize))
#
# summary(pima_norm_2)  <- se obtienen los mismos resultados, claro
```

Crear conjuntos de datos para entrenamiento y para test

Creamos los dos conjuntos con el siguiente código:

```
#Set seed to make the process reproducible
set.seed(9)

#partitioning data frame into training and testing sets
train_indices <- createDataPartition(pima_norm$diagnosis, times=1, p=.7,
list=FALSE)

#create training set
pima_norm_train <- pima_norm[train_indices, 1:7]

#create testing set
pima_norm_test  <- pima_norm[-train_indices, 1:7]

#view number of rows in each set
#nrow(pima_norm_train)  # 373
#nrow(pima_norm_test)   # 159
```

Crear el etiquetado.
(Lo necesitaremos posteriormente para evaluar el algoritmo).

```
#Labels for training and tests
pima_norm_train_labels <- pima_norm[train_indices , 8]
pima_norm_test_labels <- pima_norm[-train_indices, 8]
#length(pima_norm_train_labels)  # 373
#length(pima_norm_test_labels)   # 159
```

3 Paso 3: Entrenamiento

El algoritmo KNN, no construye un modelo, el proceso de entrenamiento consiste simplemente en almacenar los datos en un un formato estructurado. Para clasificar las instancias del conjunto test, utilizamos la implementación del algoritmo knn del paquete “class” (importado en el primer paso). Elegimos para k el valor 19, por ser 19 un valor aproximado a la raíz cuadrada del total de observaciones (373).

```
#> ?knn
pima_pred <- knn(train = pima_norm_train, test = pima_norm_test,
cl = pima_norm_train_labels, k = 19)
```

4 Paso 4: Evaluación

Para evaluar el algoritmo, comparamos el resultado obtenido, con el vector donde guardamos anteriormente los datos etiquetados (pima_norm_test_labels). Para realizar dicha comparación, utilizaremos primero la función **CrossTable** del paquete “gmodels”:

```
# CrossTable/ConfusionMatrix
CrossTable(x = pima_norm_test_labels, y = pima_pred, prop.chisq = FALSE)

##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  159
##
##                       | pima_pred
## pima_norm_test_labels |  Negative |  Positive | Row Total |
## ----------------------|-----------|-----------|-----------|
##              Negative |        94 |        12 |       106 |
##                       |     0.887 |     0.113 |     0.667 |
##                       |     0.810 |     0.279 |           |
##                       |     0.591 |     0.075 |           |
## ----------------------|-----------|-----------|-----------|
##              Positive |        22 |        31 |        53 |
##                       |     0.415 |     0.585 |     0.333 |
##                       |     0.190 |     0.721 |           |
##                       |     0.138 |     0.195 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |       116 |        43 |       159 |
##                       |     0.730 |     0.270 |           |
## ----------------------|-----------|-----------|-----------|
+
##
```

Análisis de la tabla:

- El algoritmo ha predicho 94 resultados negativos de manera correcta.
- Pero en cambio ha devuelto 22 resultados como negativos, cuando en realidad son positivos (falsos negativos). Esto puede tener consecuencias

bastante importantes, porque los pacientes pueden pensar que no tienen un problema (diabetes en este caso), y retrasar el tratamiento.

- El algoritmo también ha devuelto bastantes falsos positivos: ha devuelto 12 como casos de diabetes que en realidad no lo son.
- Sí ha devuelto 31 positivos de manera correcta.

1 Evaluación utilizando la función confusionMatrix del paquete “caret”

```
# confusionMatrix from package caret
confusionMatrix(reference = pima_norm_test_labels, data = pima_pred,
mode = "everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Negative Positive
##   Negative       94       22
##   Positive       12       31
##
##                Accuracy : 0.7862
##                  95% CI : (0.7142, 0.8471)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 0.0006448
##
##                   Kappa : 0.495
##
##  Mcnemar's Test P-Value : 0.1227126
##
##             Sensitivity : 0.8868
##             Specificity : 0.5849
##          Pos Pred Value : 0.8103
##          Neg Pred Value : 0.7209
##               Precision : 0.8103
##                  Recall : 0.8868
##                      F1 : 0.8468
##              Prevalence : 0.6667
##          Detection Rate : 0.5912
##    Detection Prevalence : 0.7296
##       Balanced Accuracy : 0.7358
##
##        'Positive' Class : Negative
##
```

Esta función nos devuelve directamente el valor de la **exactitud** (**“accuracy”**):

```
TP+TN / TP+FP+TN+FN (total) = 31 + 94 / 31 + 12 + 94 + 22 = 31 + 94 / 159 =
0.7862
```

Es decir, los clasificados correctamente sobre el total.

Con el parámetro mode = “everything”, devuelve también la Precisión, el Recall y la puntuación F1:

Precisión

Es la proporción de todas las clasificaciones positivas que realmente son positivas.

Esta función está considerando que lo "positivo" es no tener diabetes ("'Positive' Class: Negative"), así que el valor se obtiene de esta manera

Precision = TN / TN + FN = 94 / 94 + 22 = 0.8103

Recall (Recuperación, o Tasa de verdaderos positivos (TPR))

Proporción de todos los positivos reales que se clasificaron correctamente como positivos, también se conoce como recuperación.

Recall = TN / TN + FP = 94 / 94 + 12 = 0.8868

F1 score (Puntuación F1)

La puntuación F1 es la media armónica (un tipo de promedio) de la precisión y la recuperación. Se obtiene de la siguiente manera:

```
F1 = 2 x (precision x recall) / (precision + recall) = 2TP / 2TP+FP+FN = 0.8468
```

(

Como en los dos casos anteriores, se considera como positivo a la clase negativa

2x94 / 2x94 + 12 + 22 = 0.8468

)

“Esta métrica equilibra la importancia de la precisión y la recuperación, y es preferible a la precisión para los conjuntos de datos con desequilibrio de clases. Cuando la precisión y la recuperación tienen puntuaciones perfectas de 1.0, F1 también tendrá una puntuación perfecta de 1.0. En términos más generales, cuando la precisión y la recuperación sean similares en valor, F1 estará cerca de su valor. Cuando la precisión y la recuperación estén muy separadas, F1 será similar a la métrica que sea peor.”

Tomado de:

https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

5 Paso 5: intentar mejorar el modelo

1 Usar otra normalización

Ahora probamos a normalizar utilizando la normalización “z-score”. Para ello usamos la función “scale”:

```
# use the scale() function to z-score standardize a data frame (all columns but
the target variable)
pima_z <- as.data.frame(scale(pima[-8]))
pima_z <- cbind(pima_z,pima[8])
```

Resumen estadístico después de normalizar con z:

```
#Summary after normalization z-score
summary(pima_z)

##      npreg              glu                bp                skin        
##  Min.   :-1.0619   Min.   :-2.0978   Min.   :-3.85903   Min.   :-2.10781  
##  1st Qu.:-0.7599   1st Qu.:-0.7187   1st Qu.:-0.60971   1st Qu.:-0.68248  
##  Median :-0.4580   Median :-0.1945   Median : 0.04016   Median :-0.01733  
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000  
##  3rd Qu.: 0.4478   3rd Qu.: 0.6523   3rd Qu.: 0.69002   3rd Qu.: 0.64783  
##  Max.   : 4.0709   Max.   : 2.5152   Max.   : 3.12702   Max.   : 6.63422  

##       bmi                ped               age             diagnosis  
##  Min.   :-2.13486   Min.   :-1.2131   Min.   :-0.9863   Negative:355  
##  1st Qu.:-0.72884   1st Qu.:-0.7088   1st Qu.:-0.8005   Positive:177  
##  Median :-0.01311   Median :-0.2524   Median :-0.3359                 
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000                 
##  3rd Qu.: 0.58272   3rd Qu.: 0.4514   3rd Qu.: 0.5933                 
##  Max.   : 4.97155   Max.   : 5.5639   Max.   : 4.5890                 
```

Los valores no están entre 0 y 1, como ocurre con la normalizacón min-max.

Repetimos ahora los mismos pasos:

Dividir en 2 conjuntos (train y test)

```
#Set seed to make the process reproducible
set.seed(9)

train_indices_z <- createDataPartition(pima_z$diagnosis, times=1, p=.7,
list=FALSE)

pima_z_train <- pima_z[train_indices, 1:7]
pima_z_test  <- pima_z[-train_indices, 1:7]
```

Etiquetado

```
pima_z_train_labels <- pima_z[train_indices, 8]
pima_z_test_labels <- pima_z[-train_indices, 8]
```

Entrenamiento

```
pima_z_pred <- knn(train = pima_z_train, test = pima_z_test,
cl = pima_z_train_labels, k = 19)
```

Evaluación

```
confusionMatrix(reference = pima_z_test_labels, data = pima_z_pred,
mode = "everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Negative Positive
##   Negative       96       22
##   Positive       10       31
##
##                Accuracy : 0.7987
##                  95% CI : (0.7279, 0.8581)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 0.000168
##
##                   Kappa : 0.52
##
##  Mcnemar's Test P-Value : 0.051830
##
##             Sensitivity : 0.9057
##             Specificity : 0.5849
##          Pos Pred Value : 0.8136
##          Neg Pred Value : 0.7561
##               Precision : 0.8136
##                  Recall : 0.9057
##                      F1 : 0.8571
##              Prevalence : 0.6667
##          Detection Rate : 0.6038
##    Detection Prevalence : 0.7421
##       Balanced Accuracy : 0.7453
##
##        'Positive' Class : Negative
##
```

Con esta normalización se consigue una pequeña mejora, se han obtenido 2 TN más y 2 FN menos.

La comparación entre las 2 normalizaciones para k=19 queda:

	min-max	z-score
Accuracy	0.7862	0.7987
Precision	0.8103	0.8136
Recall	0.8868	0.9057
F1	0.8468	0.8571

2 Usando otros valores de k

K = 15, min-max

```
pima_pred_k15 <- knn(train = pima_norm_train, test = pima_norm_test, cl =
pima_norm_train_labels, k = 15)

confusionMatrix(reference = pima_norm_test_labels, data = pima_pred_k15, mode =
"everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Negative Positive
##   Negative       93       19
##   Positive       13       34
##
##                Accuracy : 0.7987
##                  95% CI : (0.7279, 0.8581)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 0.000168
##
##                   Kappa : 0.534
##
##  Mcnemar's Test P-Value : 0.376759
##
##             Sensitivity : 0.8774
##             Specificity : 0.6415
##          Pos Pred Value : 0.8304
##          Neg Pred Value : 0.7234
##               Precision : 0.8304
##                  Recall : 0.8774
##                      F1 : 0.8532
##              Prevalence : 0.6667
##          Detection Rate : 0.5849
##    Detection Prevalence : 0.7044
##       Balanced Accuracy : 0.7594
##
##        'Positive' Class : Negative
##
```

k = 25, min-max

```
pima_pred_k25 <- knn(train = pima_norm_train, test = pima_norm_test, cl =
pima_norm_train_labels, k = 25)

confusionMatrix(reference = pima_norm_test_labels, data = pima_pred_k25, mode =
"everything")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Negative Positive
##   Negative       96       21
##   Positive       10       32
```

```
##
##                Accuracy : 0.805
##                  95% CI : (0.7348, 0.8635)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 8.118e-05
##
##                   Kappa : 0.5373
##
##  Mcnemar's Test P-Value : 0.07249
##
##             Sensitivity : 0.9057
##             Specificity : 0.6038
##          Pos Pred Value : 0.8205
##          Neg Pred Value : 0.7619
##               Precision : 0.8205
##                  Recall : 0.9057
##                      F1 : 0.8610
##              Prevalence : 0.6667
##          Detection Rate : 0.6038
##    Detection Prevalence : 0.7358
##       Balanced Accuracy : 0.7547
##
##        'Positive' Class : Negative
##
```

k = 15, z-score

```
pima_pred_z_k15 <- knn(train = pima_z_train, test = pima_z_test, cl =
pima_z_train_labels, k = 15)

confusionMatrix(reference = pima_z_test_labels, data = pima_pred_z_k15, mode =
"everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Negative Positive
##   Negative       94       21
##   Positive       12       32
##
##                Accuracy : 0.7925
##                  95% CI : (0.7211, 0.8526)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 0.0003351
##
##                   Kappa : 0.5123
##
##  Mcnemar's Test P-Value : 0.1637344
##
##             Sensitivity : 0.8868
##             Specificity : 0.6038
##          Pos Pred Value : 0.8174
##          Neg Pred Value : 0.7273
##               Precision : 0.8174
##                  Recall : 0.8868
##                      F1 : 0.8507
##              Prevalence : 0.6667
```

```
##          Detection Rate : 0.5912
##    Detection Prevalence : 0.7233
##       Balanced Accuracy : 0.7453
##
##        'Positive' Class : Negative
##
```

k = 25, z-score

```
pima_pred_z_k25 <- knn(train = pima_z_train, test = pima_z_test, cl =
pima_z_train_labels, k = 25)

confusionMatrix(reference = pima_z_test_labels, data = pima_pred_z_k25, mode =
"everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Negative Positive
##   Negative       98       22
##   Positive        8       31
##
##                Accuracy : 0.8113
##                  95% CI : (0.7417, 0.8689)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 3.776e-05
##
##                   Kappa : 0.5455
##
##  Mcnemar's Test P-Value : 0.01762
##
##             Sensitivity : 0.9245
##             Specificity : 0.5849
##          Pos Pred Value : 0.8167
##          Neg Pred Value : 0.7949
##               Precision : 0.8167
##                  Recall : 0.9245
##                      F1 : 0.8673
##              Prevalence : 0.6667
##          Detection Rate : 0.6164
##    Detection Prevalence : 0.7547
##       Balanced Accuracy : 0.7547
##
##        'Positive' Class : Negative
##
```

3 Resumen tabular de los resultados y conclusiones

Por curiosidad (por aprender ML con Python también), he realizado el mismo análisis utilizando Python y la librería Scikit-learn. Los resultados son muy parecidos).

	R		**Python & Scikit-learn**	
K=15	**min-max**	**z-score**	**min-max**	**z-score**
Accuracy	0.7987	0.7925	0.7687	0.7938
Precision	**0.8304**	0.8174	0.7021	0.6304
Recall	0.8774	0.8868	0.5893	0.6444
F1	0.8532	0.8507	0.6408	0.6374
K=19	**min-max**	**z-score**	**min-max**	**z-score**
Accuracy	0.7862	0.7987	0.7750	**0.8438**
Precision	0.8103	0.8136	**0.7647**	0.7500
Recall	0.8868	0.9057	0.4643	**0.6667**
F1	0.8468	0.8571	0.5778	**0.7059**
K=25	**min-max**	**z-score**	**min-max**	**z-score**
Accuracy	0.805	**0.8113**	0.7625	0.8250
Precision	0.8205	0.8167	0.6875	0.7073
Recall	0.9057	**0.9245**	0.5893	0.6444
F1	0.8610	**0.8673**	0.6346	0.6744

Con un k igual a 25, y con la normalización z-score, se han obtenido mejores resultados que con la k y la normalización iniciales.
El volumen de datos no me permite realizar una valoración más profunda que estas simples conclusiones. He probado con otros valores para k, y no he visto que se mejoren estos resultados.

```
# Other Ks (not better results obtained)
#
# k = 10
#pima_pred_z_k10 <- knn(train = pima_z_train, test = pima_z_test, cl = pima_z_train_labels, k = 10)
#confusionMatrix(reference = pima_z_test_labels, data = pima_pred_z_k10, mode = "everything")
# ->  Accuracy : 0.7799
#
# k = 30
#pima_pred_z_k30 <- knn(train = pima_z_train, test = pima_z_test, cl = pima_z_train_labels, k = 30)
#confusionMatrix(reference = pima_z_test_labels, data = pima_pred_z_k30, mode = "everything")
# -> Accuracy : 0.805
#
# k = 35
#pima_pred_z_k35 <- knn(train = pima_z_train, test = pima_z_test, cl = pima_z_train_labels, k = 35)
#confusionMatrix(reference = pima_z_test_labels, data = pima_pred_z_k35, mode = "everything")
# -> Accuracy : 0.8113, Recall : 0.9434 (same accuracy, better recall than k=25)
```

EVALUACIÓN TEMA 4 - APRENDIZAJE PROBABILÍSTICO. CLASIFICACIÓN MEDIANTE NAIVE BAYES

Nota: El alumno deberá presentar todo el código utilizado para resolver el ejercicio.

El fichero "movie-pang02.csv", disponible en la carpeta de Pruebas de Evaluación del Máster, contiene una muestra de 2000 reviews de películas de la página web IMDB utilizada en el artículo de Pang, B. y Lee, L., A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, *Proceedings of ACL 2004*. Dichas reviews están etiquetadas mediante la variable *class* como positivas ("Pos") o negativas ("Neg").
Utilizando dicho *dataset*, elabore un modelo de clasificación de reviews en base a su texto siguiendo el procedimiento descrito en el capítulo 4 del texto base en el ejemplo de SMS Spam. En particular, genere las nubes de palabras para las reviews positivas y negativas, y obtenga el modelo asignando los valores 0 y 1 al parámetro *laplace* de la función *naiveBayes()*, comparando las matrices de confusión de cada variante del modelo.

1 Paso 1: Cargar datos

Cargar datos del fichero Movie_pang02.csv

```
# import the CSV file
movies_raw <- read.csv(file.path("Chapter04", "Movie_pang02.csv"))
```

2 Paso 2: Explorar y preparar los datos

Carga de paquetes que son necesarios para diversas funciones.

```
#install.packages("tm")           # text mining
library(tm)
#install.packages("SnowballC")    # stemming
library(SnowballC)
#install.packages("wordcloud")    # wordclouds
library(wordcloud)
#install.packages("RColorBrewer") # color palettes
library(RColorBrewer)

#install.packages("naivebayes")
library(naivebayes)

#install.packages("caret")     # data partitioning, confusion matrix
library(caret)
```

Examinamos la estructura y el aspecto del fichero importado:

```
#See the structure
str(movies_raw)

## 'data.frame':    2000 obs. of  2 variables:
##  $ class: chr  "Pos" "Pos" "Pos" "Pos" ...
##  $ text : chr  " films adapted from comic books have had plenty of success
whether they re about superheroes   batman   super"| __truncated__ " every now
and then a movie comes along from a suspect studio   with every indication that
it will be a stinker"| __truncated__ " you ve got mail works alot better than it
deserves to    in order to make the film a success   all they had to"|
__truncated__ "    jaws   is a rare film that grabs your attention before it
shows you a single image on screen    the movie o"| __truncated__ …

#See some records
#head(movies_raw)  #omitted for brevity

#See some records
#tail(movies_raw)  #omitted for brevity
```

La columna “class” es de tipo carácter. Ya que se trata en realidad de una variable categórica, es conveniente transformarla en un factor:

```
#Convert class into a factor
movies_raw$class <- factor(movies_raw$class)
```

Y examinamos el resultado

```
table(movies_raw$class)

##
##  Neg  Pos
## 1000 1000
```

Vemos que hay tantas críticas positivas como negativas.

1 Procesado del texto de las críticas

Procedemos ahora a preparar el fichero para que pueda ser procesado mediante el algoritmo Naive Bayes: Hay que eliminar mayúsculas y signos de puntuación, números, realizar la lematización (“stemming” en inglés) …

El primer paso consiste en la creación del objeto corpus con todas las críticas:

```
#create corpus
movies_corpus <- VCorpus(VectorSource(movies_raw$text))
print(movies_corpus)
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2000
#inspect(movies_corpus) omitted for brevity
# examine the movies corpus
# lapply(movies_corpus[1:5], as.character)
# we omit the output for brevity
```

Limpieza de los textos:

```
# Process the reviews (we will use some functions from the packate tm)

#To lowercase (content seems to be already in lowercase and without punctuation
signs, but just in case)
movies_corpus_clean <- tm_map(movies_corpus, content_transformer(tolower))

#Check the result (output omitted for brevity)
#as.character(movies_corpus[[1]])
#as.character(movies_corpus_clean[[1]])

#Remove numbers
movies_corpus_clean <- tm_map(movies_corpus_clean, removeNumbers)

#Remove stopwords
# check words and languages with ?stopwords
movies_corpus_clean <- tm_map(movies_corpus_clean, removeWords, stopwords())
#Remove punctuation signs
movies_corpus_clean <- tm_map(movies_corpus_clean, removePunctuation)

#Carry out the stemming:
# To apply the wordStem() function to an entire corpus of text documents, the tm
package includes
# the stemDocument() transformation.
movies_corpus_clean <- tm_map(movies_corpus_clean, stemDocument)

#Finally eliminate unneeded whitespace produced by previous steps
movies_corpus_clean <- tm_map(movies_corpus_clean, stripWhitespace)

#Check the final result (output omitted for brevity)
#before cleaning
#as.character(movies_corpus[[1]])
#after
#as.character(movies_corpus_clean[[1]])
```

Finalmente, se procede a la **“tokenización”** de los textos de las críticas. Mediante la función DocumentTermMatrix() del paquete “tm”, se crea una estructura llamada **“document-term matrix (DTM)”**, que como su nombre indica es una matriz, cuyas filas consisten en los textos(documentos) y cuyas columnas son las palabras que aparecen en esos documentos. Los valores en cada celda son el número de ocurrencias de cada palabra en cada documento.

```
movies_dtm <- DocumentTermMatrix(movies_corpus_clean)
movies_dtm
## <<DocumentTermMatrix (documents: 2000, terms: 24951)>>
## Non-/sparse entries: 501761/49400239
## Sparsity           : 99%
## Maximal term length: 53
## Weighting          : term frequency (tf)
```

Tenemos 2000 documentos, y 24951 palabras.

Ahora hay que crear los conjuntos de entrenamiento y de test. Las críticas vienen ordenadas, primero las 1000 críticas positivas, y después las 1000 críticas negativas. Por tanto hay que crear estos dos conjuntos de manera aleatoria.

```
#Set seed to make the process reproducible
set.seed(8)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(movies_raw$class, times=1, p=.75,
list=FALSE)

#create training set
movies_dtm_train <- movies_dtm[train_indices, ]

#create testing set
movies_dtm_test  <- movies_dtm[-train_indices, ]

#create labels sets
movies_train_labels <- movies_raw[train_indices, ]$class
movies_test_labels <- movies_raw[-train_indices, ]$class

#view number of rows in each set
#nrow(movies_dtm_train)  # 1500
#nrow(movies_dtm_test)   # 500
#length(movies_train_labels)  # 1500
#length(movies_test_labels)   # 500
```

Vamos a comprobar ahora que los dos conjuntos tienen la misma proporción de críticas positivas y negativas (si no, el entrenamiento no serviría para nada):

```
prop.table(table(movies_train_labels))

movies_train_labels
Neg Pos
0.5 0.5

prop.table(table(movies_test_labels))

movies_test_labels
Neg Pos
0.5 0.5
```

La proporción se mantiene en los dos conjuntos.

2 Visualización mediante nubes de palabras (wordclouds)

Se puede obtener una nube con las palabras de todas las críticas, y también una nube para las críticas positivas y otra para las negativas.

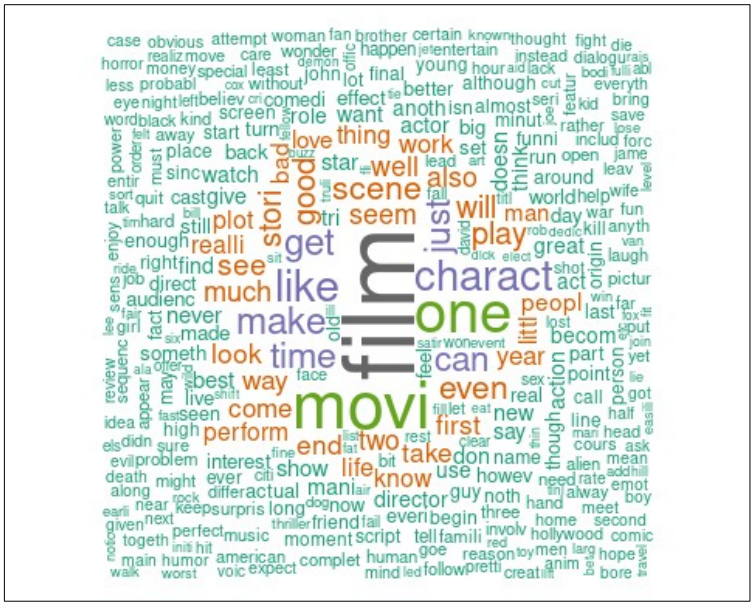
Total de críticas:

```
wordcloud(movies_raw$tex, min.freq = 50, random.order = FALSE)
```

[

Usando el corpus y con coloreado se obtiene una nube muy parecida:

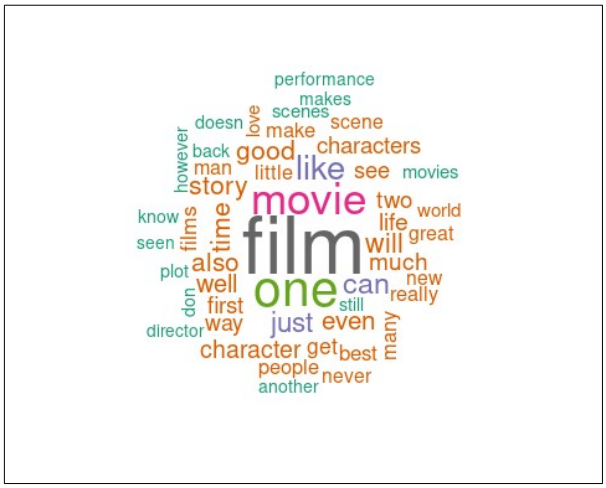
```
#Using the corpus:
#wordcloud(movies_corpus_clean, min.freq = 50, random.order = FALSE)
#Using the corpus with color:
wordcloud(movies_corpus_clean, min.freq = 50, random.order = FALSE,
colors=brewer.pal(8,"Dark2"))
```



]

Críticas positivas y críticas negativas por separado:

```
pos <- subset(movies_raw, class == "Pos")
neg <- subset(movies_raw, class == "Neg")

#Positive reviews
wordcloud(pos$text, max.words = 50, random.order = FALSE,
colors=brewer.pal(8,"Dark2"))
```




```
#Negative reviews
wordcloud(neg$text, max.words = 50, random.order = FALSE,
colors=brewer.pal(8,"Dark2"))
```

La única diferencia que aprecio entre las dos nubes, es que en la nube “negativa“ se encuentra la palabra “bad” (también “never”). Palabras positivas como “good”, like”, “well”, aparecen en las 2 agrupaciones. (“like” es también una preposición/conjunción, no tiene solo el significado de “gustar”).

Finalizamos ahora la preparación de los datos.

Se necesita obtener un listado con las palabras más utilizadas. Utilizamos la función findFreqTerms indicando que queremos los términos que aparezcan al menos 100 veces:

```
# Data preparation – creating indicator features for frequent words
# the function findFreqTerms() in the tm package takes a DTM and returns a
character vector containing words that appear at least a  minimum number of
times
movies_freq_words <- findFreqTerms(movies_dtm_train, 100)

movies_freq_words
##    [1] "abil"         "abl"          "absolut"      "accept"
##    [5] "achiev"       "across"       "act"          "action"
...
## [1017] "worth"        "wouldn"       "write"        "writer"
## [1021] "written"      "wrong"        "year"         "yes"
## [1025] "yet"          "york"         "young"
```

Y ahora utilizamos ese listado para limitar el número de columnas/features de los conjuntos de entrenamiento y de test:

```
#> ncol(movies_dtm_train)
#[1] 24951
movies_dtm_freq_train <- movies_dtm_train[ , movies_freq_words]
movies_dtm_freq_test <- movies_dtm_test[ , movies_freq_words]
#> ncol(movies_dtm_freq_train)
#[1] 1027
```

Ahora los conjuntos de entrenamiento y test tienen 1027 columnas/features, en lugar de 24951.

Finalmente, ya que las matrices DTM tienen valores numéricos, mientras que el algoritmo de clasificación basado en naive bayes en su versión bernoulli necesita operar sobre variables categóricas, se realiza una última tranformación: pasar los valores numéricos a Sí/No

```
#We need a function that converts counts to a factor
convert_counts <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}

#and we apply it
movies_train <- apply(movies_dtm_freq_train, MARGIN = 2, convert_counts)  #
MARGIN = 2 <- columns
movies_test <- apply(movies_dtm_freq_test, MARGIN = 2, convert_counts)
# The result will be two matrices, each with cells indicating "Yes" or "No" for
#whether the word represented by the column appears at any point in the message
#represented by the row.
```

3 Paso 3: Entrenamiento del modelo

```
movies_classifier <- naive_bayes(movies_train, movies_train_labels)

movies_classifier
```

```
##
## ================================== Naive Bayes ==================================
##
## Call:
## naive_bayes.default(x = movies_train, y = movies_train_labels)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: 0
##
## ---------------------------------------------------------------------------------
##
## A priori probabilities:
##
## Neg Pos
## 0.5 0.5
##
## ---------------------------------------------------------------------------------
##
## Tables:
##
## ---------------------------------------------------------------------------------
## :: abil (Bernoulli)
## ---------------------------------------------------------------------------------
##
## abil         Neg        Pos
##   No  0.91866667 0.91600000
##   Yes 0.08133333 0.08400000
##
## ---------------------------------------------------------------------------------
## :: abl (Bernoulli)
## ---------------------------------------------------------------------------------
##
## abl         Neg       Pos
##   No  0.8760000 0.8266667
##   Yes 0.1240000 0.1733333
##
##
...
## ---------------------------------------------------------------------------------
## :: achiev (Bernoulli)
## ---------------------------------------------------------------------------------
##
## achiev        Neg        Pos
##    No  0.94800000 0.91466667
##    Yes 0.05200000 0.08533333
##
## ---------------------------------------------------------------------------------
##
## # ... and 1022 more tables
##
## ---------------------------------------------------------------------------------
```

4 Paso 4: Evaluación del modelo

Realizamos la predicción

```
#predict
movies_test_pred <- predict(movies_classifier, movies_test)
```

Y comparamos lo predicho por el algoritmo con los datos etiquetados anteriormente

```
#confusion matrix
confusionMatrix(reference = movies_test_labels, data = movies_test_pred, mode =
"everything", positive = "Pos")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Neg Pos
##        Neg 207  59
##        Pos  43 191
##
##                Accuracy : 0.796
##                  95% CI : (0.758, 0.8305)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.592
##
##  Mcnemar's Test P-Value : 0.1375
##
##             Sensitivity : 0.7640
##             Specificity : 0.8280
##          Pos Pred Value : 0.8162
##          Neg Pred Value : 0.7782
##               Precision : 0.8162
##                  Recall : 0.7640
##                      F1 : 0.7893
##              Prevalence : 0.5000
##          Detection Rate : 0.3820
##    Detection Prevalence : 0.4680
##       Balanced Accuracy : 0.7960
##
##        'Positive' Class : Pos
##
```

Como intuía, al no haber diferencias tan evidentes entre las nubes de palabras de las críticas positivas y negativas como en las nubes del ejercicio del libro sobre el spam de los mensajes SMS, el resultado no es tan expectacular como en ese otro caso. Al algoritmo Naive Bayes le cuesta aquí algo más predecir correctamente.

Exactitud (**“accuracy”**):

Es decir, los clasificados correctamente sobre el total.

```
TP + TN / TP + FP + TN + FN (total) = 191 + 207 / 191 + 43 + 207 + 59  = 191 + 207 / 500 = 0.796
```

Precisión

Proporción de de positivos reales del total de positivos devuelto:

```
Precision = TP / TP + FP = 191 / 191 + 43 = 0.8162
```

Recall (Recuperación, o Tasa de verdaderos positivos (TPR))

Proporción de todos los positivos reales que se clasificaron correctamente como positivos, también se conoce como recuperación.

```
Recall = TP / TP + FN = 191 / 191 + 59 = 0.7640
```

F1 score (Puntuación F1)

La puntuación F1 es la media armónica (un tipo de promedio) de la precisión y la recuperación. Se obtiene de la siguiente manera:

```
F1 = 2x(precisionxrecall) / (precision+recall) = 2TP / 2TP + FP + FN = 0.7893
```

5 Paso 5: Mejora del modelo

Configuramos el estimador de Laplace (parametro “Laplace smoothing”) con el valor 1. De esta manera se evita que las palabras que solo aparecen en uno de los 2 conjuntos (entrenamiento o tes), empeoren el resultado total.

```
#  laplace = 1
movies_classifier_laplace <- naive_bayes(movies_train, movies_train_labels,
laplace = 1)
movies_classifier_laplace

##
## ============================== Naive Bayes ==================================
##
## Call:
## naive_bayes.default(x = movies_train, y = movies_train_labels,
##     laplace = 1)
##
## -------------------------------------------------------------------------------
##
## Laplace smoothing: 1
##
## -------------------------------------------------------------------------------
##
## A priori probabilities:
##
## Neg Pos
## 0.5 0.5
##
```

```
## ------------------------------------------------------------------------------
##
## Tables:
##
## ------------------------------------------------------------------------------
## :: abil (Bernoulli)
## ------------------------------------------------------------------------------
##
## abil         Neg        Pos
##   No  0.91755319 0.91489362
##   Yes 0.08244681 0.08510638
##
## ------------------------------------------------------------------------------
## :: abl (Bernoulli)
## ------------------------------------------------------------------------------
##
## abl         Neg       Pos
##   No  0.8750000 0.8257979
##   Yes 0.1250000 0.1742021
##
...
## ------------------------------------------------------------------------------
## :: achiev (Bernoulli)
## ------------------------------------------------------------------------------
##
## achiev        Neg        Pos
##    No  0.94680851 0.91356383
##    Yes 0.05319149 0.08643617
##
## ------------------------------------------------------------------------------
##
## # ... and 1022 more tables
##
## ------------------------------------------------------------------------------
```

Volvemos a ejecutar el algoritmo

```
#predict
movies_test_pred_laplace <- predict(movies_classifier_laplace, movies_test)
```

Y volvemos a comparar lo predicho por el algoritmo con los datos etiquetados

```
#confusion matrix
confusionMatrix(reference = movies_test_labels, data = movies_test_pred_laplace,
mode = "everything", positive = "Pos")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Neg Pos
##        Neg 207  61
##        Pos  43 189
##
##                Accuracy : 0.792
##                  95% CI : (0.7537, 0.8268)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.584
##
##  Mcnemar's Test P-Value : 0.09552
##
##             Sensitivity : 0.7560
##             Specificity : 0.8280
```

```
##          Pos Pred Value : 0.8147
##          Neg Pred Value : 0.7724
##               Precision : 0.8147
##                  Recall : 0.7560
##                      F1 : 0.7842
##              Prevalence : 0.5000
##          Detection Rate : 0.3780
##    Detection Prevalence : 0.4640
##       Balanced Accuracy : 0.7920
##
##        'Positive' Class : Pos
##
```

Para mí sorpresa, el resultado es peor: ahora se han clasificado 2 críticas positivas más de manera errónea como negativas.

Dado que el resultado no ha sido todo lo positivo que esperábamos, vamos a probar ahora utilizando el clasificador naive bayes multinomial (MNB), en lugar del anterior (bernoulli).

La diferencia entre ambos radica en que la versión bernoulli trabaja con características binarias, y la versión multinomial con características numéricas (por eso no tenemos que aplicar la función conver_counts).

Paso 5.1: Utilización de la versión multinomial del algoritmo Naive Bayes

Entrenamiento del modelo:

```
#model training with laplace = 0 first
movies_classifier_mnb <- multinomial_naive_bayes(movies_dtm_freq_train,
movies_train_labels, laplace = 0)

movies_classifier_mnb
```

```
##
## ============================ Multinomial Naive Bayes ============================
##
## Call:
## multinomial_naive_bayes(x = movies_dtm_freq_train, y = movies_train_labels,
##     laplace = 0)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: 0
##
## ---------------------------------------------------------------------------------
##
## A priori probabilities:
## Neg Pos
## 0.5 0.5
##
## ---------------------------------------------------------------------------------
##
##          Classes
## Features           Neg          Pos
##   abil    0.0004125129 0.0004228878
##   abl     0.0006562705 0.0009021607
##   absolut 0.0005875184 0.0004849114
```

```
##   accept  0.0004500141 0.0004736344
##   achiev  0.0003250102 0.0004228878
##   across  0.0004937654 0.0004623573
##   act     0.0027438357 0.0022102937
##   action  0.0033688553 0.0024076413
##   actor   0.0027375855 0.0025937119
##   actress 0.0004687646 0.0004736344
##
## ------------------------------------------------------------------------------
##
## # ... and 1017 more features
##
## ------------------------------------------------------------------------------
```

Ejecución/predicción:

Al llamar al método predict directamente con la dtm de test, a diferencia del paso anterior (entrenamiento), da un error.

```
# predict
#movies_dtm_freq_test
#inspect(movies_dtm_freq_test[1:10,1:10])
#    Terms
#Docs abil abl absolut accept achiev across act action actor actress
#  10    0   0       0      0      0      0   0      4     0       0
#  12    1   0       0      0      0      0   0      0     1       0
#  18    0   0       0      0      0      0   4      4     2       0
# ...
#movies_test_pred_mnb <- predict(movies_classifier_mnb, newdata =
movies_matrix_freq_test, type='class')
# It returns
# Error: predict.multinomial_naive_bayes(): newdata must be a numeric matrix or
dgCMatrix (Matrix package) with at least one row and two named columns.
```

Así que convertimos la dtm de test en una matriz, y ahora sí obtenemos la predicción:

```
#So I try again transforming the dtm into a matrix
movies_test_pred_mnb <- predict(movies_classifier_mnb, newdata =
as.matrix(movies_dtm_freq_test), type='class')

#movies_test_pred_mnb
#  [1] Neg Neg Pos Pos Pos Pos Pos Pos Neg Pos Pos Pos Pos Pos Pos Pos Pos Pos
Pos Pos Pos Neg Neg Pos Pos Pos Neg Pos Neg Pos Pos Pos Neg Pos Neg Pos Pos Pos
# [40] Pos Pos Pos Pos Pos Pos Pos Pos Pos Pos Pos Pos Pos Pos Pos Neg Neg Pos Pos
Pos Neg Pos Neg Pos Neg Pos Pos Pos Pos Pos Neg Pos Pos Neg Pos Pos Pos Pos Pos
#
…
#
# Converting to a dgCMatrix also works
#movies_dgCMatrix_freq_test <-  Matrix::sparseMatrix(i=movies_dtm_freq_test$i,
#                           j=movies_dtm_freq_test$j,
#                           x=movies_dtm_freq_test$v,
#                           dims=c(movies_dtm_freq_test$nrow,
movies_dtm_freq_test$ncol),
#                           dimnames = movies_dtm_freq_test$dimnames)
#movies_dgCMatrix_freq_test
# 500 x 1027 sparse Matrix of class "dgCMatrix"
#
```

```
#2  . . . . 1 . . . . . . . . . 1 . . 2 . . . . . . . . . . . . 1 . . . . . . . .
1 .  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .....
#5  . . . . . 2 . . 1 . . . . . . . . . . 1 . . . . 1 . 1 . . . 1 .
1 . . . . . .  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
1 2 ......
#...
#movies_test_pred_mnb <- predict(movies_classifier_mnb, newdata =
movies_dgCMatrix_freq_test, type='class')
```

Volvemos a comparar lo predicho por el algoritmo con los datos etiquetados

```
#confusion matrix
confusionMatrix(reference = movies_test_labels, data = movies_test_pred_mnb,
mode = "everything", positive = "Pos")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Neg Pos
##        Neg 194  41
##        Pos  56 209
##
##                Accuracy : 0.806
##                  95% CI : (0.7686, 0.8398)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.612
##
##  Mcnemar's Test P-Value : 0.1552
##
##             Sensitivity : 0.8360
##             Specificity : 0.7760
##          Pos Pred Value : 0.7887
##          Neg Pred Value : 0.8255
##               Precision : 0.7887
##                  Recall : 0.8360
##                      F1 : 0.8117
##              Prevalence : 0.5000
##          Detection Rate : 0.4180
##    Detection Prevalence : 0.5300
##       Balanced Accuracy : 0.8060
##
##        'Positive' Class : Pos
##
```

Se obtienen resultados ligeramente mejores que con la versión bernoulli.

Finalmente, probamos con laplace = 1 también para esta versión de Naive Bayes:

```
#model training with laplace = 1
movies_classifier_mnb_laplace <- multinomial_naive_bayes(movies_dtm_freq_train,
movies_train_labels, laplace = 1)

movies_classifier_mnb_laplace

##
## ======================== Multinomial Naive Bayes ============================
##
## Call:
## multinomial_naive_bayes(x = movies_dtm_freq_train, y = movies_train_labels,
##     laplace = 1)
##
## ----------------------------------------------------------------------------
…

#prediction
movies_test_pred_mnb_laplace <- predict(movies_classifier_mnb_laplace, newdata =
as.matrix(movies_dtm_freq_test), type='class')
#confusion matrix
confusionMatrix(reference = movies_test_labels, data =
movies_test_pred_mnb_laplace, mode = "everything", positive = "Pos")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Neg Pos
##        Neg 194  42
##        Pos  56 208
##
##                Accuracy : 0.804
##                  95% CI : (0.7664, 0.8379)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.608
##
##  Mcnemar's Test P-Value : 0.1891
##
##             Sensitivity : 0.8320
##             Specificity : 0.7760
##          Pos Pred Value : 0.7879
##          Neg Pred Value : 0.8220
##               Precision : 0.7879
##                  Recall : 0.8320
##                      F1 : 0.8093
##              Prevalence : 0.5000
##          Detection Rate : 0.4160
##    Detection Prevalence : 0.5280
##       Balanced Accuracy : 0.8040
##
##        'Positive' Class : Pos
##
```

6 Comparativa final

Se adjuntan los resultados obtenidos tanto en R como usando Python.

(En el anexo se pueden encontrar los enlaces a los repositorios en GitHub).

R	Bernoullin laplace 0	Bernoullin laplace 1	MNB laplace 0	MNB laplace 1
Matriz de confusión	Reference Pred. Neg Pos Neg **207** 59 Pos 43 **191**	Reference Pred. Neg Pos Neg **207** **61** Pos 43 **189**	Reference Pred. Neg Pos Neg **194** 41 Pos 56 **209**	Reference Pred. Neg Pos Neg **194** **42** Pos 56 **208**
Exactitud	0.796	0.792	**0.806**	0.804
Precisión	**0.8162**	0.8147	0.7887	0.7879
Recall	0.7640	0.7560	**0.8360**	0.8320
F1	0.7893	0.7842	**0.8117**	0.8093
Python&Scikit-learn	Bernoullin laplace 0	Bernoullin laplace 1.3	MNB laplace 0	MNB laplace 1.3
Matriz de confusión	Confusion Matrix: [[**200** 50] [77 **173**]]	Confusion Matrix: [[**202** 48] [77 **173**]]	Confusion Matrix: [[**194** 56] [46 **204**]]	Confusion Matrix: [[**197** 53] [50 **200**]]
Exactitud	0.746	0.75	0.796	0.794
Precisión				
Recall				
F1				

Los resultados en esta tarea vuelven a ser parecidos en R y en Python, rozando el 80% de exactitud.

No parece que el factor de laplace influya demasiado tampoco.

La única explicación que se me ocurre, es que no se da el caso de que haya palabras en el conjunto de test que no están en el conjunto de entrenamiento.

EVALUACIÓN TEMA 5 - CLASIFICACIÓN MEDIANTE ÁRBOLES DE DECISIÓN

Nota: El alumno deberá presentar todo el código utilizado para resolver el ejercicio.

El *dataset* Carseats incluido en la librería *ISLR* incluye datos relativos a las ventas de sillitas de coche para niños de 400 establecimientos. Puede encontrarse información detallada sobre cada variable incluida el dataset en https://www.rdocumentation.org/packages/ISLR/versions/1.4/topics/Carseats.

Usando dicho *dataset*, construya un árbol de decisión, utilizando un 75% de la muestra como conjunto de entrenamiento, para predecir la variable *Sales* en base al resto de variables e interprete los resultados, comentando las reglas obtenidas.

Para realizar esta prueba, previamente se recomienda convertir *Sales* en una variable categórica usando la función *ifelse*. Para ello, será necesario establecer un punto de corte usando algún criterio predefinido (ie, valor por encima o por debajo de la media o la mediana).

1 Paso 1: Carga de los datos

```
#Load data from CRAN package ISLR
#install.packages("ISLR")
library("ISLR")
```

[
Descripción de los campos:

Carseats: Sales of Child Car Seats Description
A **simulated** data set containing sales of child car seats at 400 different stores.
Sales - Unit sales (in thousands) at each location
CompPrice - Price charged by competitor at each location
Income - Community income level (in thousands of dollars)
Advertising - Local advertising budget for company at each location (in thousands of dollars)
Population - Population size in region (in thousands)
Price - Price company charges for car seats at each site
ShelveLoc - A factor with levels Bad, Good and Medium indicating the quality of the shelving location for the car seats at each site
Age - Average age of the local population
Education - Education level at each location
Urban - A factor with levels No and Yes to indicate whether the store is in an urban or rural location
US - A factor with levels No and Yes to indicate whether the store is in the US or not
]

2 Paso 2: Explorar y preparar los datos

Carga de paquetes necesarios para diversas funciones.

```
#install.packages("C50")  # Decision trees C5.0 algorithm
library(C50)

#install.packages("caret")  # data partitioning, confusion matrix
library(caret)
## Lade nötiges Paket: ggplot2
## Lade nötiges Paket: lattice
```

Examinamos la estructura y el aspecto del dataset importado:

```
#See the structure
str(Carseats)

## 'data.frame':    400 obs. of  11 variables:
##  $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...

#Summary
summary(Carseats)

##      Sales          CompPrice       Income        Advertising
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000

##    Population        Price        ShelveLoc        Age          Education
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00   Min.   :10.0
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75   1st Qu.:12.0
##  Median :272.0   Median :117.0   Medium:219   Median :54.50   Median :14.0
##  Mean   :264.8   Mean   :115.8                Mean   :53.32   Mean   :13.9
##  3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00   3rd Qu.:16.0
##  Max.   :509.0   Max.   :191.0                Max.   :80.00   Max.   :18.0

##  Urban       US
##  No :118   No :142
##  Yes:282   Yes:258
##
```

La variable dependiente “Sales” es numérica. Para poder predecir si un carrito se venderá o no en función de las variables independientes, debemos transformarla en una variable categórica tipo Sí/No. Consideramos que si las ventas están por encima de la media será un “Sí”, y si no, un “No”:

```
#transform Sales into SalesFactor
sales_mean <- mean(Carseats$Sales)  # mean and median are almost the same

Carseats$SalesFactor <- factor(ifelse(Carseats$Sales>sales_mean,"Yes","No"))

table(Carseats$SalesFactor)
##
##  No Yes
## 201 199

#Check the result of the conversion
sum(Carseats$Sales > sales_mean)
## [1] 199
```

Eliminamos ahora la columna original “sales”

```
#Remove sales variable
CarseatsNew <- Carseats[-1]
```

Ahora hay que crear los conjuntos de entrenamiento y de test. Aunque los datos en principio no vienen ordenados, para estar seguros vamos a crear estos dos conjuntos de manera aleatoria.

```
#Set seed to make the process reproducible
set.seed(9)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(CarseatsNew$SalesFactor, times=1, p=.75,
list=FALSE)

#create training set
CarseatsNew_train <- CarseatsNew[train_indices, ]

#create testing set
CarseatsNew_test  <- CarseatsNew[-train_indices, ]

#create labels sets
CarseatsNew_train_labels <- CarseatsNew[train_indices, ]$SalesFactor
CarseatsNew_test_labels <- CarseatsNew[-train_indices, ]$SalesFactor

#view number of rows in each set
#nrow(CarseatsNew_train)  # 301
#nrow(CarseatsNew_test)   # 99
#length(CarseatsNew_train_labels)  # 301
#length(CarseatsNew_test_labels)   # 99
```

Comprobamos que la proporción se mantiene en los dos conjuntos:

```
#Check the proportion in both sets
prop.table(table(CarseatsNew_train$SalesFactor))
##
##        No       Yes
## 0.5016611 0.4983389
prop.table(table(CarseatsNew_test$SalesFactor))
##
##        No       Yes
## 0.5050505 0.4949495
```

3 Paso 3: Entrenamiento del modelo

```
# For the first iteration of the model, we use the default C5.0 settings

sales_model <- C5.0(SalesFactor ~ ., data = CarseatsNew_train)

sales_model

##
## Call:
## C5.0.formula(formula = SalesFactor ~ ., data = CarseatsNew_train)
##
## Classification Tree
## Number of samples: 301
## Number of predictors: 10
##
## Tree size: 19
##
## Non-standard options: attempt to group attributes
```

Para examinar el modelo (el árbol), utilizamos la función summary:

```
# To see the tree's decisions, we can call the summary() function on the model:
summary(sales_model)
##
## Call:
## C5.0.formula(formula = SalesFactor ~ ., data = CarseatsNew_train)
##
## C5.0 [Release 2.07 GPL Edition]      Mon Feb  3 13:28:42 2025
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 301 cases (11 attributes) from undefined.data
##
## Decision tree:
##
## ShelveLoc = Good:
## :...Price <= 135: Yes (51/1)
## :   Price > 135:
## :   :...Income <= 75: No (7/1)
## :       Income > 75: Yes (5/1)
## ShelveLoc in {Bad,Medium}:
```

```
## :...Price > 105:
##     :...CompPrice <= 142:
##     :   :...Advertising <= 10: No (86/6)
##     :   :   Advertising > 10:
##     :   :   :...Price > 126: No (22/2)
##     :   :       Price <= 126:
##     :   :       :...CompPrice <= 121: No (9/1)
##     :   :           CompPrice > 121: Yes (15)
##     :   CompPrice > 142:
##     :   :...Urban = No: No (4/1)
##     :       Urban = Yes:
##     :       :...Education <= 13: Yes (6)
##     :           Education > 13:
##     :           :...Price <= 127: Yes (5)
##     :               Price > 127: No (9/3)
##     Price <= 105:
##     :...CompPrice > 123: Yes (20)
##         CompPrice <= 123:
##         :...Income > 100: Yes (12)
##             Income <= 100:
##             :...Age <= 35: Yes (8)
##                 Age > 35:
##                 :...Price <= 70: Yes (5)
##                     Price > 70:
##                     :...US = No: No (14/1)
##                         US = Yes:
##                         :...Population > 272: No (10/1)
##                             Population <= 272:
##                             :...CompPrice <= 103: No (3)
##                                 CompPrice > 103: Yes (10/1)
##
## Evaluation on training data (301 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##      19   19( 6.3%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     148     3    (a): class No
##      16   134    (b): class Yes
##
##  Attribute usage:
##
##  100.00% Price
##  100.00% ShelveLoc
##   79.07% CompPrice
##   43.85% Advertising
##   24.58% Income
##   16.61% Age
##   12.29% US
##    7.97% Urban
##    7.64% Population
##    6.64% Education
##
## Time: 0.0 secs
```

Podemos examinar las primeras ramas (reglas) del árbol:

```
Decision tree:

ShelveLoc = Good:
:...Price <= 135: Yes (51/1)             <- (i)
:   Price > 135:
          :...Income <= 75: No (7/1)     <- (ii)
          :   Income > 75: Yes (5/1)
ShelveLoc in {Bad,Medium}:
:...Price > 105:
    :...CompPrice <= 142:                <- (iii)
```

(i)
: Si el estado del expositor/estantería es bueno y el precio es menor o igual de 135, 51 sillitas de coche se venden (1 en realidad no, hay un falso positivo aquí)

(ii)
: Si el estado del expositor/estantería es bueno y el precio es superior a 135, entonces ya influyen los ingresos del comprador

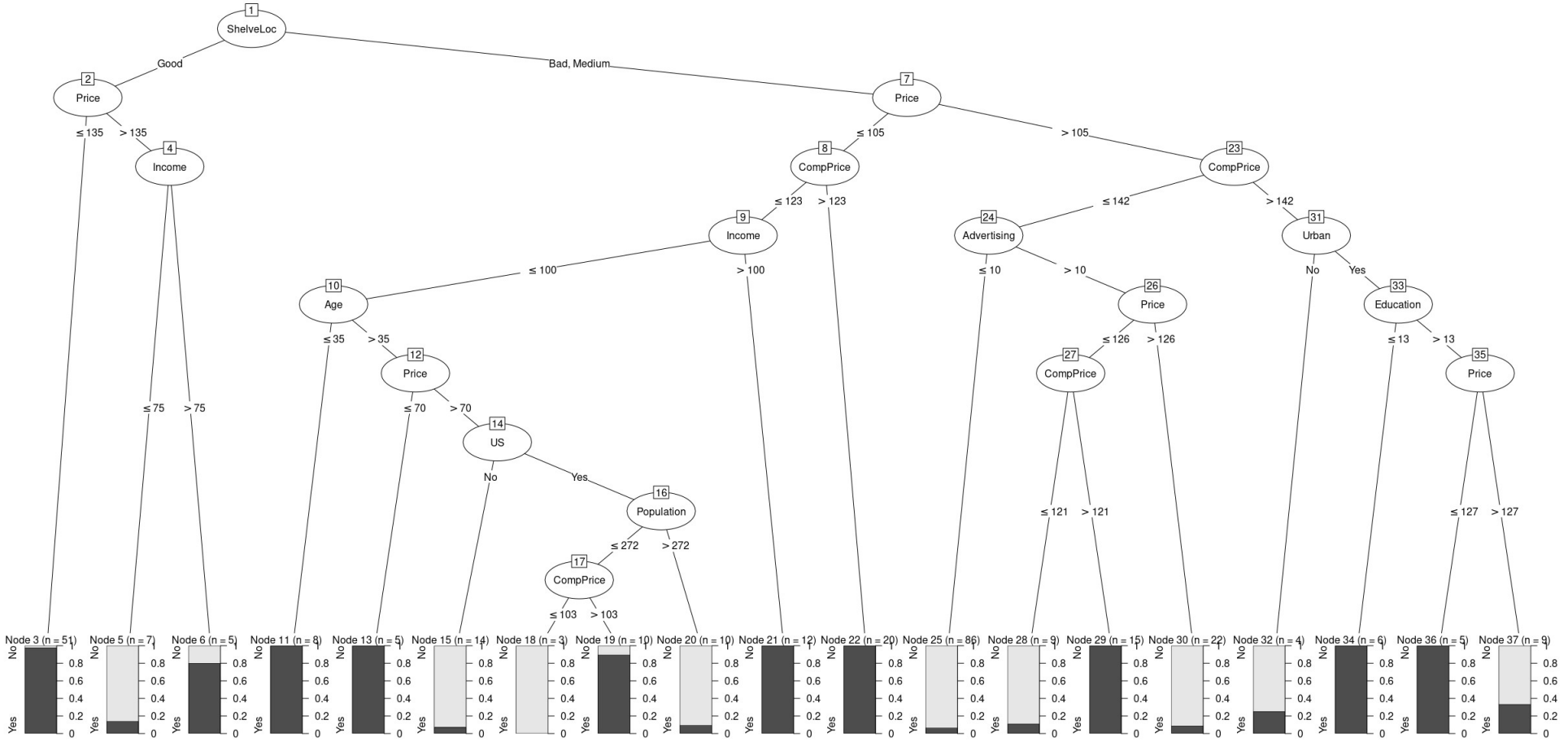
(iii)
: Si el estado del expositor/estantería no es bueno y el precio es superior a 105, entonces entran en juego otros factores como el precio del mismo carrito en otro comercio, la publicidad ..

El resultado con el conjunto de entrenamiento parece bastante bueno, solo un 6.3% de errores. (Teniendo en cuenta esto, que se trata del conjunto de entrenamiento y los árboles de decisión son propensos al sobreajuste).

Las variables más importantes han sido el precio y el estado de la estantería/expositor (si he entendido bien la descripción de “ShelveLoc”). Variables como como el total de la población, el nivel de su educación, si es un area urbana .. no parecen muy influyentes.

Visualización del árbol:

```
#plotting the model
#plot(sales_model)  <- done in another R script for better visualization
```




```
## Tree size: 19   →  19 nodos = 19 reglas
```

4 Paso 4: Evaluación del modelo

Realizamos la predicción con los datos que reservamos para test.

```
#Prediction

sales_pred <- predict(sales_model, CarseatsNew_test)
```

Y comparamos lo predicho por el algoritmo con los datos etiquetados anteriormente

```
#confusion matrix
confusionMatrix(reference = CarseatsNew_test_labels, data = sales_pred, mode =
"everything", positive = "Yes")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  41  16
##        Yes  9  33
##
##                Accuracy : 0.7475
##                  95% CI : (0.6502, 0.8294)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 7.008e-07
##
##                   Kappa : 0.4942
##
##  Mcnemar's Test P-Value : 0.2301
##
##             Sensitivity : 0.6735
##             Specificity : 0.8200
##          Pos Pred Value : 0.7857
##          Neg Pred Value : 0.7193
##               Precision : 0.7857
##                  Recall : 0.6735
##                      F1 : 0.7253
##              Prevalence : 0.4949
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.4242
##       Balanced Accuracy : 0.7467
##
##        'Positive' Class : Yes
##
```

Al aplicar el modelo a datos nuevos, en cambio se obtiene una exactitud del 75%, (un error ahora del 25%). Como sospechábamos, se da un caso de sobreajuste a los datos de entrenamiento.

5 Paso 5: Mejora del modelo

Entiendo que este caso, en el que se intenta predecir las ventas de sillitas para bebés, no es equivalente al ejemplo del libro, donde se puede dar el caso de conceder un préstamo a alguien que no lo va a poder devolver, o algún tema relacionado con la salud, donde también puede tener consecuencias muy graves que el modelo devuelva muchos falsos negativos. Así que solo voy a aplicar la técnica de boosting, y no la de considerar algunos errores más costosos que otros.

La función C5.0 permite aplicar la técnica de boosting simplemente añadiendo un parámetro como se puede ver a continuación. Este parámetro indica el número de árboles a usar. Es un límite "por arriba", el algoritmo dejará de añadir árboles en cuanto detecte que no se está mejorando la exactitud.

```
# boosting, we use the C5.0 parameter trials and set it to 100

sales_model_boost100 <- C5.0(SalesFactor ~ ., data = CarseatsNew_train, trials =
100)  # trials = 10 (first try)

sales_model_boost100
```

```
##
## Call:
## C5.0.formula(formula = SalesFactor ~ ., data = CarseatsNew_train, trials = 100)
##
## Classification Tree
## Number of samples: 301
## Number of predictors: 10
##
## Number of boosting iterations: 100
## Average tree size: 16.2
##
## Non-standard options: attempt to group attributes
```

```
# To see the tree's decisions, we can call the summary() function on the model:

summary(sales_model_boost100)
```

```
##
## Call:
## C5.0.formula(formula = SalesFactor ~ ., data = CarseatsNew_train, trials = 100)
##
##
## C5.0 [Release 2.07 GPL Edition]      Mon Feb  3 13:28:43 2025
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 301 cases (11 attributes) from undefined.data
##
## -----  Trial 0:  -----
##
## Decision tree:
##
## ShelveLoc = Good:
## :...Price <= 135: Yes (51/1)
## :   Price > 135:
```

```
## :   :...Income <= 75: No (7/1)
## :       Income > 75: Yes (5/1)
## ShelveLoc in {Bad,Medium}:
## :...Price > 105:
##     :...CompPrice <= 142:
##     :   :...Advertising <= 10: No (86/6)
##     :   :   Advertising > 10:
##     :   :   :...Price > 126: No (22/2)
##     :   :       Price <= 126:
##     :   :       :...CompPrice <= 121: No (9/1)
##     :   :           CompPrice > 121: Yes (15)
##     :   CompPrice > 142:
##     :   :...Urban = No: No (4/1)
##     :       Urban = Yes:
##     :       :...Education <= 13: Yes (6)
##     :           Education > 13:
##     :           :...Price <= 127: Yes (5)
##     :               Price > 127: No (9/3)
##     Price <= 105:
##     :...CompPrice > 123: Yes (20)
##         CompPrice <= 123:
##         :...Income > 100: Yes (12)
##             Income <= 100:
##             :...Age <= 35: Yes (8)
##                 Age > 35:
##                 :...Price <= 70: Yes (5)
##                     Price > 70:
##                     :...US = No: No (14/1)
##                         US = Yes:
##                         :...Population > 272: No (10/1)
##                             Population <= 272:
##                             :...CompPrice <= 103: No (3)
##                                 CompPrice > 103: Yes (10/1)
##
## -----  Trial 1:  -----
##
## Decision tree:
##
## Age > 76: No (18.4/2.3)
## Age <= 76:
## :...ShelveLoc = Bad:
##     :...CompPrice <= 151: No (47.5/11.5)
##     :   CompPrice > 151: Yes (6/0.8)
##     ShelveLoc in {Good,Medium}:
##     :...Price <= 100: Yes (48/3.8)
##         Price > 100:
##         :...CompPrice <= 114: No (35.6/8.4)
##             CompPrice > 114:
##             :...ShelveLoc = Good:
##                 :...Price <= 156: Yes (28.2)
##                 :   Price > 156: No (2.3)
##                 ShelveLoc = Medium:
##                 :...Age <= 33: Yes (27.7/1.5)
##                     Age > 33:
##                     :...CompPrice > 142: Yes (24.1/3.1)
##                         CompPrice <= 142:
##                         :...Price > 127: No (23)
##                             Price <= 127:
##                             :...Population <= 199: No (10.7/3.1)
##                                 Population > 199: Yes (29.3/5.4)
##
## -----  Trial 2:  -----
##
## Decision tree:
##
## Advertising > 7:
## :...ShelveLoc = Good: Yes (29/1.2)
## :   ShelveLoc in {Bad,Medium}:
## :   :...Price <= 105: Yes (39.4/3)
## :       Price > 105:
## :       :...US = No: Yes (3.5)
## :           US = Yes:
## :           :...Advertising > 21: Yes (10.1)
## :               Advertising <= 21:
```

```
## :               :...Education <= 10: Yes (10.2/1.2)
## :                   Education > 10:
## :                   :...Price > 126: No (15.7)
## :                       Price <= 126:
## :                       :...Price <= 111: No (6.7)
## :                           Price > 111: Yes (21.8/4.9)
## Advertising <= 7:
## :...Price > 144: No (19.9/1.2)
##     Price <= 144:
##     :...ShelveLoc = Good: Yes (19.9/4.1)
##         ShelveLoc in {Bad,Medium}:
##         :...US = Yes:
##             :...Age <= 41: Yes (10.1/1.8)
##             :   Age > 41: No (30.8/4.2)
##             US = No:
##             :...Price <= 72: Yes (7.3)
##                 Price > 72:
##                 :...CompPrice <= 120: No (20/1.8)
##                     CompPrice > 120:
##                     :...Advertising > 2: Yes (13/1.8)
##                         Advertising <= 2:
##                         :...Price > 127: No (6.6)
##                             Price <= 127:
##                             :...Age <= 66: Yes (31.6/7.3)
##                                 Age > 66: No (5.5)
##
## -----  Trial 3:  -----
##
## Decision tree:
##
## Price <= 100:
## :...Income <= 25: No (4)
## :   Income > 25: Yes (66.3/11.6)
## Price > 100:
## :...ShelveLoc = Bad:
##     :...US = No: No (22.1/0.5)
##     :   US = Yes:
##     :   :...Education > 16: Yes (7/1.4)
##     :       Education <= 16:
##     :       :...Advertising <= 15: No (20.5)
##     :           Advertising > 15: Yes (5.7/1.4)
##     ShelveLoc in {Good,Medium}:
##     :...Income <= 61:
##         :...Advertising <= 3: No (32.1/1.9)
##         :   Advertising > 3:
##         :   :...CompPrice <= 110: No (6.8)
##         :       CompPrice > 110:
##         :       :...Price <= 141: Yes (35.1/12.3)
##         :           Price > 141: No (5.4)
##         Income > 61:
##         :...ShelveLoc = Good: Yes (25.4/3.2)
##             ShelveLoc = Medium:
##             :...Age <= 33: Yes (12/0.5)
##                 Age > 33:
##                 :...Population <= 391: No (31.4/8)
##                     Population > 391:
##                     :...Advertising <= 1: No (3.3)
##                         Advertising > 1: Yes (23.7/2.8)
##
##


…


## -----  Trial 98:  -----
##
## Decision tree:
##
## Age > 73:
## :...Education > 17: Yes (4.4/0.4)
## :   Education <= 17:
## :   :...Price <= 84: Yes (2.6)
## :       Price > 84: No (36.8/1.7)
## Age <= 73:
```

```
## :...Price <= 104:
##     :...Advertising > 7: Yes (27.2/1.7)
##     :   Advertising <= 7:
##     :   :...CompPrice > 121: Yes (9.6)
##     :       CompPrice <= 121:
##     :       :...Price <= 70: Yes (6.8)
##     :           Price > 70: No (32/13)
##     Price > 104:
##     :...ShelveLoc = Good:
##         :...Income <= 41: No (14.9/4)
##         :   Income > 41: Yes (22.7/1.7)
##         ShelveLoc in {Bad,Medium}:
##         :...Education <= 10: Yes (13.4/2.8)
##             Education > 10:
##             :...Advertising > 20: Yes (7.1)
##                 Advertising <= 20:
##                 :...Price > 129: No (35.7/1.2)
##                     Price <= 129:
##                     :...Price <= 111: No (13.6/0.2)
##                         Price > 111:
##                         :...Price <= 115: Yes (8.5/0.4)
##                             Price > 115:
##                             :...CompPrice <= 121: No (12.5)
##                                 CompPrice > 121:
##                                 :...Advertising > 10: Yes (12.1/2)
##                                     Advertising <= 10:
##                                     :...Population > 402: Yes (4.7)
##                                         Population <= 402:
##                                         :...Age > 71: Yes (2.5)
##                                             Age <= 71:
##                                             :...Advertising > 4: No (7.6)
##                                                 Advertising <= 4:
##                                                 :...CompPrice > 146: Yes (4.6)
##                                                     CompPrice <= 146:
##                                                     :...Age <= 33: Yes (4.2/1.2)
##                                                         Age > 33: No (17.5)
##
## -----  Trial 99:  -----
##
## Decision tree:
##
## Price <= 105:
## :...ShelveLoc = Good: Yes (13.3)
## :   ShelveLoc in {Bad,Medium}:
## :   :...Age > 73: No (12.6/2)
## :       Age <= 73:
## :       :...Urban = No: Yes (24/1.9)
## :           Urban = Yes:
## :           :...Advertising > 19: No (3.5)
## :               Advertising <= 19:
## :               :...Advertising <= 0: No (15.9/6.1)
## :                   Advertising > 0: Yes (34.1/4.8)
## Price > 105:
## :...ShelveLoc = Good:
##     :...Price > 156: No (5.9)
##     :   Price <= 156:
##     :   :...Urban = No: No (13.8/4.8)
##     :       Urban = Yes: Yes (25.6/1.5)
##     ShelveLoc in {Bad,Medium}:
##     :...Urban = No:
##         :...Income <= 94: No (34.3/2)
##         :   Income > 94: Yes (10.6/3.9)
##         Urban = Yes:
##         :...Advertising > 21: Yes (5.5)
##             Advertising <= 21:
##             :...Price > 139: No (18.1)
##                 Price <= 139:
##                 :...CompPrice <= 121: No (21.8/2.8)
##                     CompPrice > 121:
##                     :...Education <= 10: Yes (8.2)
##                         Education > 10:
##                         :...ShelveLoc = Bad:
##                             :...Education <= 16: No (21/4.6)
##                             :   Education > 16: Yes (5.2/0.9)
```

```
##                             ShelveLoc = Medium:
##                             :...CompPrice <= 141: No (21.8/8.1)
##                                 CompPrice > 141: Yes (5.8)
##
##
## Evaluation on training data (301 cases):
##
## Trial        Decision Tree
## -----      ----------------
##    Size      Errors
##
##    0     19   19( 6.3%)
##    1     12   52(17.3%)
##    2     18   39(13.0%)
##    3     15   50(16.6%)
##    4     17   54(17.9%)
##    5     15   36(12.0%)
##    6     13   64(21.3%)
##    7     14   49(16.3%)
##    8     23   36(12.0%)
##    9     14   52(17.3%)
##   10     13   57(18.9%)
##   11     17   41(13.6%)
##   12     13   73(24.3%)
##   13     17   37(12.3%)
##   14     18   47(15.6%)
##   15     17   46(15.3%)
##   16     16   57(18.9%)
##   17     15   25( 8.3%)
##   18     15   56(18.6%)
##   19     19   39(13.0%)
##   20     18   40(13.3%)
##   21     17   45(15.0%)
##   22     17   43(14.3%)
##   23     19   37(12.3%)
##   24     15   35(11.6%)
##   25     19   27( 9.0%)
##   26     14   70(23.3%)
##   27     14   40(13.3%)
##   28     19   61(20.3%)
##   29     15   33(11.0%)
##   30     14   52(17.3%)
##   31     15   46(15.3%)
##   32     15   46(15.3%)
##   33     17   49(16.3%)
##   34     14   57(18.9%)
##   35     15   34(11.3%)
##   36     13   61(20.3%)
##   37     15   61(20.3%)
##   38     15   51(16.9%)
##   39     11   44(14.6%)
##   40     19   32(10.6%)
##   41     21   42(14.0%)
##   42     13   59(19.6%)
##   43     21   40(13.3%)
##   44     14   43(14.3%)
##   45     16   43(14.3%)
##   46     18   52(17.3%)
##   47     16   46(15.3%)
##   48     20   47(15.6%)
##   49     16   41(13.6%)
##   50     17   54(17.9%)
##   51      8   48(15.9%)
##   52     14   59(19.6%)
##   53     17   46(15.3%)
##   54     16   43(14.3%)
##   55     14   46(15.3%)
##   56     12   50(16.6%)
##   57     20   38(12.6%)
##   58     19   64(21.3%)
##   59     22   49(16.3%)
##   60     14   44(14.6%)
##   61      9   58(19.3%)
##   62     17   55(18.3%)
```

```
##   63     16   53(17.6%)
##   64     18   32(10.6%)
##   65     11   65(21.6%)
##   66     18   43(14.3%)
##   67     16   54(17.9%)
##   68     16   39(13.0%)
##   69     19   56(18.6%)
##   70     17   34(11.3%)
##   71     17   47(15.6%)
##   72     19   45(15.0%)
##   73     17   49(16.3%)
##   74     17   37(12.3%)
##   75     18   33(11.0%)
##   76     12   54(17.9%)
##   77     21   42(14.0%)
##   78     14   47(15.6%)
##   79     13   62(20.6%)
##   80     13   54(17.9%)
##   81     17   50(16.6%)
##   82     18   30(10.0%)
##   83     16   48(15.9%)
##   84     14   52(17.3%)
##   85     15   34(11.3%)
##   86     20   40(13.3%)
##   87     18   43(14.3%)
##   88     15   36(12.0%)
##   89     15   39(13.0%)
##   90     17   33(11.0%)
##   91     16   51(16.9%)
##   92     19   44(14.6%)
##   93     18   37(12.3%)
##   94     12   40(13.3%)
##   95     14   51(16.9%)
##   96     15   69(22.9%)
##   97     20   32(10.6%)
##   98     22   35(11.6%)
##   99     19   52(17.3%)
## boost             0( 0.0%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     151          (a): class No
##           150    (b): class Yes
##
##
##  Attribute usage:
##
##  100.00% CompPrice
##  100.00% Income
##  100.00% Advertising
##  100.00% Population
##  100.00% Price
##  100.00% ShelveLoc
##  100.00% Age
##  100.00% Education
##  100.00% Urban
##   94.35% US
##
##
## Time: 0.1 secs
```

La técnica de boosting hace que en el conjunto de entrenamiento, la tasa de errores se reduzca a 0.

Volvemos a realizar la predicción:

```
#Prediction

sales_pred_boost100 <- predict(sales_model_boost100, CarseatsNew_test)
```

Y comparamos lo predicho por el algoritmo con los datos etiquetados anteriormente:

```
#confusion matrix
confusionMatrix(reference = CarseatsNew_test_labels, data = sales_pred_boost100,
mode = "everything", positive = "Yes")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  43  15
##        Yes  7  34
##
##                Accuracy : 0.7778
##                  95% CI : (0.6831, 0.8552)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 2.178e-08
##
##                   Kappa : 0.5548
##
##  Mcnemar's Test P-Value : 0.1356
##
##             Sensitivity : 0.6939
##             Specificity : 0.8600
##          Pos Pred Value : 0.8293
##          Neg Pred Value : 0.7414
##               Precision : 0.8293
##                  Recall : 0.6939
##                      F1 : 0.7556
##              Prevalence : 0.4949
##          Detection Rate : 0.3434
##    Detection Prevalence : 0.4141
##       Balanced Accuracy : 0.7769
##
##        'Positive' Class : Yes
##
```

La mejora obtenida fue mínima con 10 iteraciones, pero con 100 es más apreciable. (Sigue siendo bastante pequeña. Me pregunto si el hecho de que los datos hayan sido generados artificialmente, puede influir en que esta técnica no produzca una mejora espectacular).

6 Comparativa final

Se adjuntan los resultados obtenidos tanto en R como en Python (también con la librería R-Weka, que permite acceder en R a las implementaciones de los algoritmos de la libreria Java WEKA, desarrollada por la Universidad de Waikato -Nueva Zelanda-).

En el anexo se pueden encontrar los enlaces a los repositorios en GitHub.

Sin Boosting	R (C5.0)	R-Weka (J48)	Python & Scikit-learn
Matriz de confusión	Reference Pred. No Yes No **41** 16 Yes 9 **33**	Reference Pred. No Yes No **37** 15 Yes 13 **34**	Reference Pred. No Yes No **38** 12 Yes 16 **34**
Exactitud	**0.7475**	0.7172	0.72
Precisión	0.6735	0.7234	
Recall	0.7640	0.6939	
F1	0.7253	0.7083	
Con Boosting (100 iteraciones)	R	R-Weka (AdaBoostM1)	Python & Scikit-learn (AdaBoostClassifier)
Matriz de confusión	Reference Pred. No Yes No **43** 15 Yes 7 **34**	Reference Pred. No Yes No **41** 12 Yes 9 **37**	Reference Pred. No Yes No **44** 6 Yes 13 **37**
Exactitud	0.7778	0.7879	**0.81**
Precisión	0.8293	0.8043	
Recall	0.6939	0.7551	
F1	0.7556	0.7789	

Los resultados en esta tarea vuelven a ser parecidos en R y en Python. En esta ocasión los resultados han sido ligeramente mejores con R sin utilizar boosting, y utilizando esta técnica han sido mejores con Python (he tenido que jugar bastante con algunos parámetros para conseguirlo, no ha sido inmediato).

EVALUACIÓN TEMA 6 - MÉTODOS DE REGRESIÓN

En la librería MASS podemos encontrar el *dataset* Boston, el cual incluye 506 observaciones de 14 variables relacionadas con el mercado de la vivienda de dicha ciudad estadounidense. Puede encontrarse información detallada sobre el contenido de dichas variables en el siguiente enlace: https://www.rdocumentation.org/packages/MASS/versions/7.3-54/topics/Boston

A partir de los datos de dicho *dataset*, construya un modelo de regresión lineal para predecir el precio de la vivienda en Boston utilizando como output la variable *medv* (valor mediano de viviendas ocupadas por sus propietarios en miles de dólares).

Para realizar el ejercicio, se recomienda crear un modelo inicial con todas las variables para seguidamente eliminar de forma secuencial aquellas variables no significativas del modelo siguiendo alguno de los métodos explicados en el material complementario del Tema 6.

Una vez obtenido el modelo final, realice una diagnosis similar a la presentada en los ejemplos del material complementario, comentando en detalle los resultados obtenidos.

Por último, utilizando este modelo realice un ejercicio de predicción asignando 5 valores diferentes a cada una de las variables que componen el modelo. Valore el resultado de estas predicciones indicando si el resultado tiene sentido económico.

1 Paso 1: Carga de los datos

```
#Load data from CRAN package MASS
#install.packages("MASS")
library(MASS)
```

2 Paso 2: Explorar y preparar los datos

Carga de paquetes que son necesarios para diversas funciones.

```
if (!require(GGally)) install.packages('GGally', dependencies = T)
library(GGally)

if (!require(gridExtra)) install.packages('gridExtra', dependencies = T
library(gridExtra)

if (!require(lmtest)) install.packages('lmtest', dependencies = T)
library(lmtest)

if (!require(car)) install.packages('car', dependencies = T)
library(car)
```

Examinamos la estructura y el aspecto del dataset importado:

```
#Structure
str(Boston)

## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524
0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...

#Summary
summary(Boston)

##       crim                 zn               indus             chas
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000
##  1st Qu.: 0.08205   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000
##       nox               rm              age              dis
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127
##       rad              tax           ptratio          black
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90
##      lstat            medv
##  Min.   : 1.73   Min.   : 5.00
##  1st Qu.: 6.95   1st Qu.:17.02
##  Median :11.36   Median :21.20
##  Mean   :12.65   Mean   :22.53
##  3rd Qu.:16.95   3rd Qu.:25.00
##  Max.   :37.97   Max.   :50.00
```

La variable dependiente en este estudio es medv (valor medio de la vivienda en 1000 de $). Al ser numérica, está incluida en el resumen estadístico.

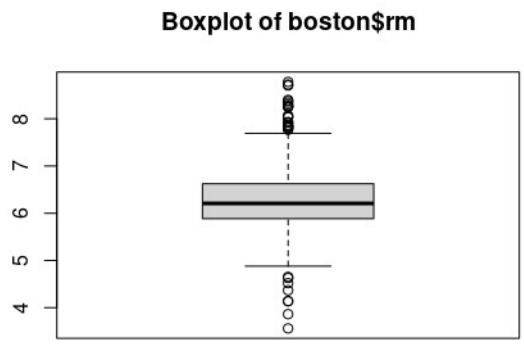
```
#check there are no nulls

boston <- Boston
write.csv(boston,file='Chapter06/boston.csv') #<- export data to be used lately
with Python & scikit-learn

#sum(is.na(boston)) -> 0
#count total missing values in each column of data frame
#sapply(boston, function(x) sum(is.na(x)))
colSums(is.na(boston))
##    crim      zn   indus    chas     nox      rm     age     dis     rad
tax
##       0       0       0       0       0       0       0       0       0
0
## ptratio   black   lstat    medv
##       0       0       0       0
```

En el sumario estadístico, se puede ver que algunas variables tienen una media y mediana bastante separadas: crim, zn. Las examinamos con más detalle mediante diagramas de caja.

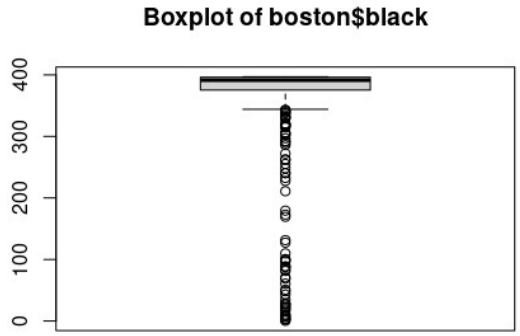
```
boxplot(x = boston$crim, main = "Boxplot of boston$crim")
```


```
boxplot(x = boston$zn, main = "Boxplot of boston$zn")
```

```
#boxplot(x = boston$indus, main = "Boxplot of boston$indus")
#boxplot(x = boston$nox, main = "Boxplot of boston$nox")
boxplot(x = boston$rm, main = "Boxplot of boston$rm")
```




```
#boxplot(x = boston$age, main = "Boxplot of boston$age")
#boxplot(x = boston$dis, main = "Boxplot of boston$dis")
#boxplot(x = boston$tax, main = "Boxplot of boston$tax")
#boxplot(x = boston$ptratio, main = "Boxplot of boston$ptratio")
boxplot(x = boston$black, main = "Boxplot of boston$black")
```




```
#boxplot(x = boston$lstat, main = "Boxplot of boston$lstat")
```

Al hacer los diagramas de caja de todos los predictores (por si acaso), he visto que también hay muchos valores atípicos para black y rm. (rm no creo que afecte tanto, porque el rango es muy pequeño -de hecho la mediana y la media de rm no son muy diferentes-). Entiendo que estas variables pueden provocar efectos indeseados en el resultado final.

1 Análisis de correlación

Para poder establecer un modelo de regresión lineal múltiple, lo primero es estudiar la relación que existe entre las variables independientes y la variable

dependiente (y entre ellas también, claro). Para ello, comenzamos obteniendo la matriz de correlación entre todas las variables disponibles:

```
corr_matrix <- round(cor(x = boston, method = "pearson"), 3)

corr_matrix
##           crim     zn  indus   chas    nox     rm    age    dis    rad   tax ptratio  black  lstat   medv
## crim     1.000 -0.200  0.407 -0.056  0.421 -0.219  0.353 -0.380  0.626  0.583  0.290 -0.385  0.456 -0.388
## zn      -0.200  1.000 -0.534 -0.043 -0.517  0.312 -0.570  0.664 -0.312 -0.315 -0.392  0.176 -0.413  0.360
## indus    0.407 -0.534  1.000  0.063  0.764 -0.392  0.645 -0.708  0.595  0.721  0.383 -0.357  0.604 -0.484
## chas    -0.056 -0.043  0.063  1.000  0.091  0.091  0.087 -0.099 -0.007 -0.036  0.122  0.049 -0.054  0.175
## nox      0.421 -0.517  0.764  0.091  1.000 -0.302  0.731 -0.769  0.611  0.668  0.189 -0.380  0.591 -0.427
## rm      -0.219  0.312 -0.392  0.091 -0.302  1.000 -0.240  0.205 -0.210 -0.292 -0.356  0.128 -0.614  0.695
## age      0.353 -0.570  0.645  0.087  0.731 -0.240  1.000 -0.748  0.456  0.506  0.262 -0.274  0.602 -0.377
## dis     -0.380  0.664 -0.708 -0.099 -0.769  0.205 -0.748  1.000 -0.495 -0.534 -0.232  0.292 -0.497  0.250
## rad      0.626 -0.312  0.595 -0.007  0.611 -0.210  0.456 -0.495  1.000  0.910  0.465 -0.444  0.489 -0.382
## tax      0.583 -0.315  0.721 -0.036  0.668 -0.292  0.506 -0.534  0.910  1.000  0.461 -0.442  0.544 -0.469
## ptratio  0.290 -0.392  0.383 -0.122  0.189 -0.356  0.262 -0.232  0.465  0.461  1.000 -0.177  0.374 -0.508
## black   -0.385  0.176 -0.357  0.049 -0.380  0.128 -0.274  0.292 -0.444 -0.442 -0.177  1.000 -0.366  0.333
## lstat    0.456 -0.413  0.604 -0.054  0.591 -0.614  0.602 -0.497  0.489  0.544  0.374 -0.366  1.000 -0.738
## medv    -0.388  0.360 -0.484  0.175 -0.427  0.695 -0.377  0.250 -0.382 -0.469 -0.508  0.333 -0.738  1.000


#corr_matrix[,14]
corr_matrix[,14][order(abs(corr_matrix[,14]))]


##    chas     dis   black      zn     age     rad    crim     nox     tax   indus  ptratio      rm   lstat    medv
##   0.175   0.250   0.333   0.360  -0.377  -0.382  -0.388  -0.427  -0.469  -0.484   -0.508   0.695  -0.738   1.000
```

Los valores más relacionados con el precio de la vivienda son ‘lstat’ (correlación negativa) y ‘rm’ (positiva).

(Parece lógico -y un poco clasista también, pero bueno-: cuantas más habitaciones, mayor precio, cuanto más porcentaje de ‘clase baja’ en la zona, menor precio).

(Por no hablar de la variable ‘black’ ...)

Las variables que parecen menos relacionados con el precio de la vivienda son ‘chas’, ‘dis’, ‘black’ y ‘zn’.

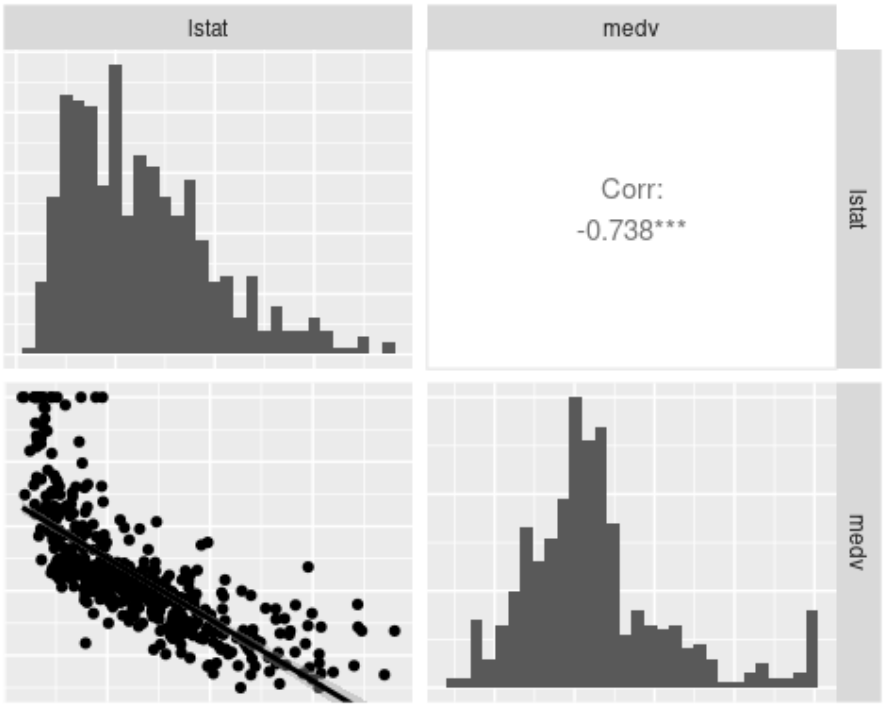
También se ve que están muy relacionadas entre sí ‘indus’ y ‘nox’, ‘tax’ y ‘rad’. (Investigando en internet he visto que la multicolinealidad puede ser un problema).

Como se indica en el material complementario de este tema, también se puede utilizar la función “ggpairs” de la libreria **GGally**. Con esta función, además de los valores de correlación para cada par de variables, también se obtiene los diagramas de dispersión y la distribución de cada una de las variables:

No puedo ver bien aquí el resultado poniendo todas las variables juntas, así que elijo algunas de ellas.

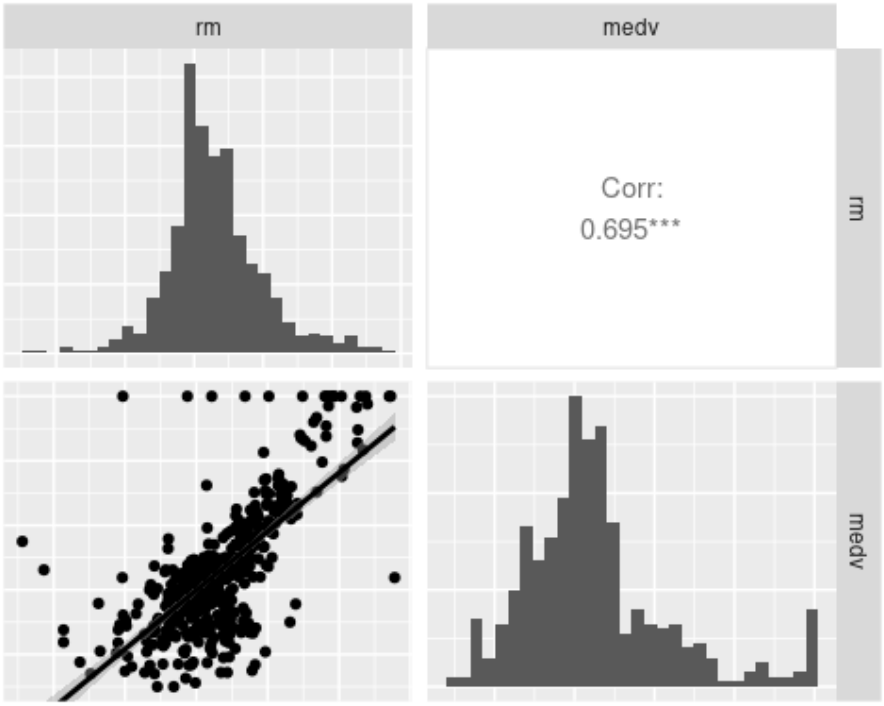
(En python, como en la pantalla del navegador donde se ejecutan los notebooks the JupyterLab hay más espacio, se ve algo mejor).

```
ggpairs(boston[,c(13,14)], lower = list(continuous = "smooth"), diag =
list(continuous = "barDiag"), axisLabels = "none")
```

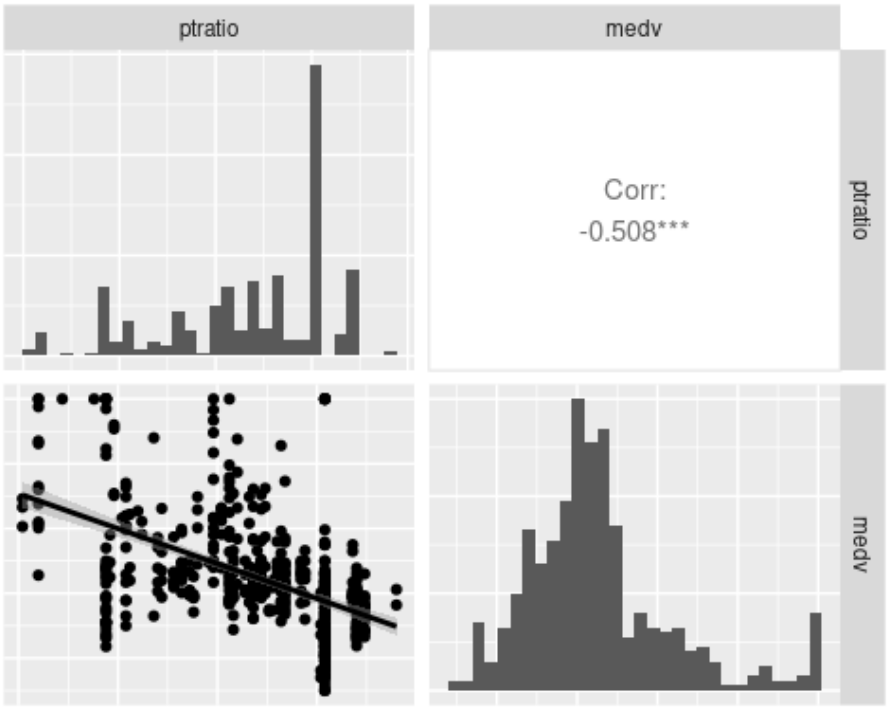


La relación entre lstat y medv puede que no sea exactamente lineal. Quizá pueda ser una mejora del modelo.

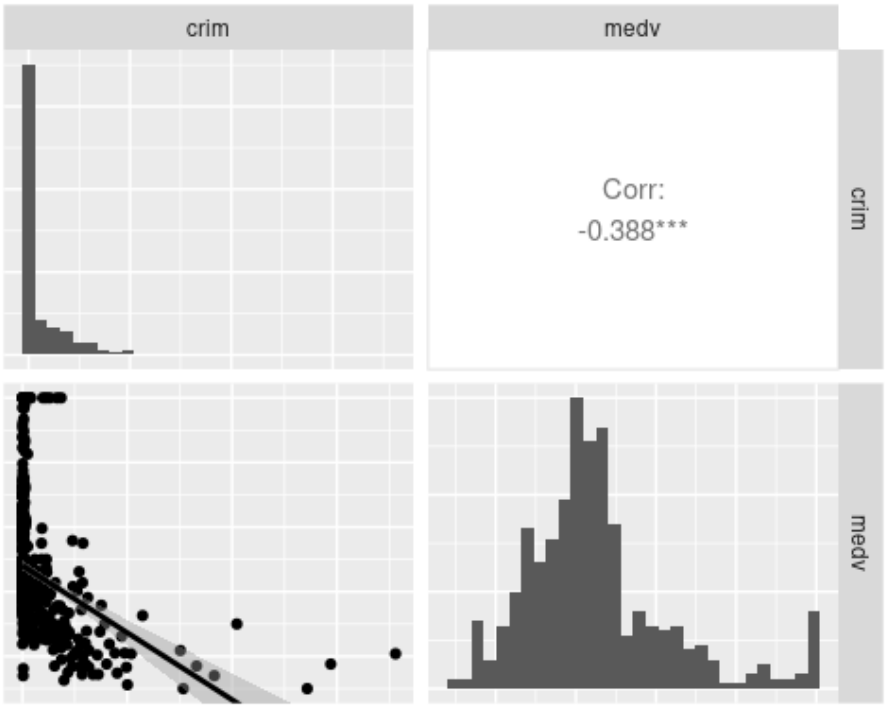
```
ggpairs(boston[,c(6,14)], lower = list(continuous = "smooth"), diag =
list(continuous = "barDiag"), axisLabels = "none")
```



```
ggpairs(boston[,c(11,14)], lower = list(continuous = "smooth"), diag =
list(continuous = "barDiag"), axisLabels = "none")
```




```
ggpairs(boston[,c(1,14)], lower = list(continuous = "smooth"), diag =
list(continuous = "barDiag"), axisLabels = "none")
```



No veo que entre estas dos variables haya alguna relación “razonable” la verdad. El diagrama de dispersion me parece muy extraño también.

Una vez visto todo esto (

- variables que son cualitativas como chas y rad

- variables que están muy muy relacionadas entre sí, como indus y nox, tax y rad

- variables que tienen muchos valores atípicos como crim, zn y black

) estimamos el modelo con todas ellas, y luego utilizaremos el criterio AIC para ver si es conveniente eliminar alguna.

```
model <- lm(medv ~ ., data = boston)

summary(model)
##
## Call:
## lm(formula = medv ~ ., data = boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas         2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## black        9.312e-03  2.686e-03   3.467 0.000573 ***
## lstat       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

El modelo con todos los predictores tiene un **R^2** de **0.7406**, lo que indica que es capaz de explicar un 74% de la variabilidad del precio de las viviendas. El hecho de que el **valor p** del estadístico F sea despreciable (**< 2.2e-16**), y también los valores p de todas las variables menos 2 (‘indus’ y ‘age’), indican que es probable que el modelo sea correcto. (Así que con todo lo extrañas que encuentro esas variables que menciono, parece que las que no aportan son indus y age).

Vamos a comprobar si efectivamente esas dos variables ‘indus’ y ‘age’ no aportan nada al modelo. Para ello utilizaremos el criterio de Akaike (AIC - Akaike Information Criterion):

```
#?step -> Choose a model by AIC in a Stepwise Algorithm
step(object = model, direction = "both", trace = 1)
##
## Start:  AIC=1589.64
## medv ~ crim + zn + indus + chas + nox + rm + age + dis + rad +
##     tax + ptratio + black + lstat
##
##           Df Sum of Sq   RSS    AIC
## - age      1      0.06 11079 1587.7
## - indus    1      2.52 11081 1587.8
## <none>                 11079 1589.6
## - chas     1    218.97 11298 1597.5
## - tax      1    242.26 11321 1598.6
## - crim     1    243.22 11322 1598.6
## - zn       1    257.49 11336 1599.3
## - black    1    270.63 11349 1599.8
## - rad      1    479.15 11558 1609.1
## - nox      1    487.16 11566 1609.4
## - ptratio  1   1194.23 12273 1639.4
## - dis      1   1232.41 12311 1641.0
## - rm       1   1871.32 12950 1666.6
## - lstat    1   2410.84 13490 1687.3
##
## Step:  AIC=1587.65
## medv ~ crim + zn + indus + chas + nox + rm + dis + rad + tax +
##     ptratio + black + lstat
##
##           Df Sum of Sq   RSS    AIC
## - indus    1      2.52 11081 1585.8
## <none>                 11079 1587.7
## + age      1      0.06 11079 1589.6
## - chas     1    219.91 11299 1595.6
## - tax      1    242.24 11321 1596.6
## - crim     1    243.20 11322 1596.6
## - zn       1    260.32 11339 1597.4
## - black    1    272.26 11351 1597.9
## - rad      1    481.09 11560 1607.2
## - nox      1    520.87 11600 1608.9
## - ptratio  1   1200.23 12279 1637.7
## - dis      1   1352.26 12431 1643.9
## - rm       1   1959.55 13038 1668.0
## - lstat    1   2718.88 13798 1696.7
##
## Step:  AIC=1585.76
## medv ~ crim + zn + chas + nox + rm + dis + rad + tax + ptratio +
##     black + lstat
##
##           Df Sum of Sq   RSS    AIC
## <none>                 11081 1585.8
## + indus    1      2.52 11079 1587.7
## + age      1      0.06 11081 1587.8
## - chas     1    227.21 11309 1594.0
## - crim     1    245.37 11327 1594.8
## - zn       1    257.82 11339 1595.4
## - black    1    270.82 11352 1596.0
## - tax      1    273.62 11355 1596.1
## - rad      1    500.92 11582 1606.1
## - nox      1    541.91 11623 1607.9
## - ptratio  1   1206.45 12288 1636.0
## - dis      1   1448.94 12530 1645.9
## - rm       1   1963.66 13045 1666.3
## - lstat    1   2723.48 13805 1695.0
##
## Call:
## lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad +
##     tax + ptratio + black + lstat, data = boston)
##
```

```
##
## Coefficients:
## (Intercept)         crim           zn         chas          nox           rm
##   36.341145    -0.108413     0.045845     2.718716   -17.376023     3.801579
##         dis          rad          tax      ptratio        black        lstat
##   -1.492711     0.299608    -0.011778    -0.946525     0.009291    -0.522553
```

Pues según este criterio (es mejor un valor AIC menor, y que el modelo tenga menos variables), el mejor modelo es

medv ~ crim + zn + chas + nox + rm + dis + rad + tax + ptratio + black + lstat

es decir, el que efectivamente no contiene ni indus ni age.

```
model <- lm(medv ~ crim + zn + chas + nox + rm + dis + rad + tax + ptratio +
black + lstat, data = boston)

summary(model)
##
## Call:
## lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad +
##     tax + ptratio + black + lstat, data = boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.5984  -2.7386  -0.5046   1.7273  26.2373
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  36.341145   5.067492   7.171 2.73e-12 ***
## crim         -0.108413   0.032779  -3.307 0.001010 **
## zn            0.045845   0.013523   3.390 0.000754 ***
## chas          2.718716   0.854240   3.183 0.001551 **
## nox         -17.376023   3.535243  -4.915 1.21e-06 ***
## rm            3.801579   0.406316   9.356  < 2e-16 ***
## dis          -1.492711   0.185731  -8.037 6.84e-15 ***
## rad           0.299608   0.063402   4.726 3.00e-06 ***
## tax          -0.011778   0.003372  -3.493 0.000521 ***
## ptratio      -0.946525   0.129066  -7.334 9.24e-13 ***
## black         0.009291   0.002674   3.475 0.000557 ***
## lstat        -0.522553   0.047424 -11.019  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.736 on 494 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7348
## F-statistic: 128.2 on 11 and 494 DF,  p-value: < 2.2e-16
```

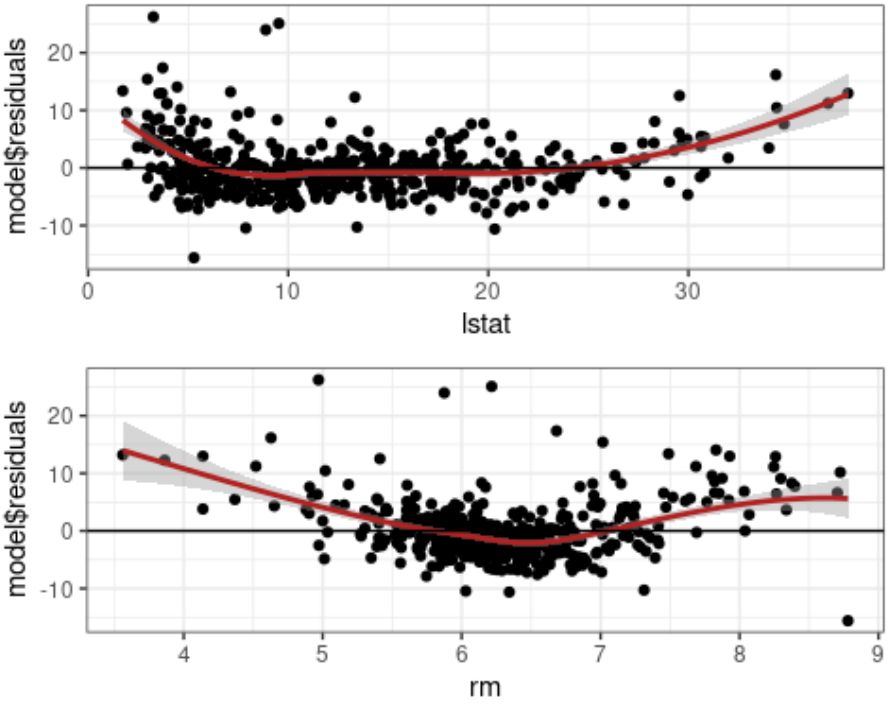
Se comprueba que el valor R^2 es igual que el del modelo completo, pero el valor estadístico F es mejor.

3 Validez del modelo

Pasamos a validar el modelo. Debemos asegurarnos de que cumple los supuestos de un modelo de regresión lineal.

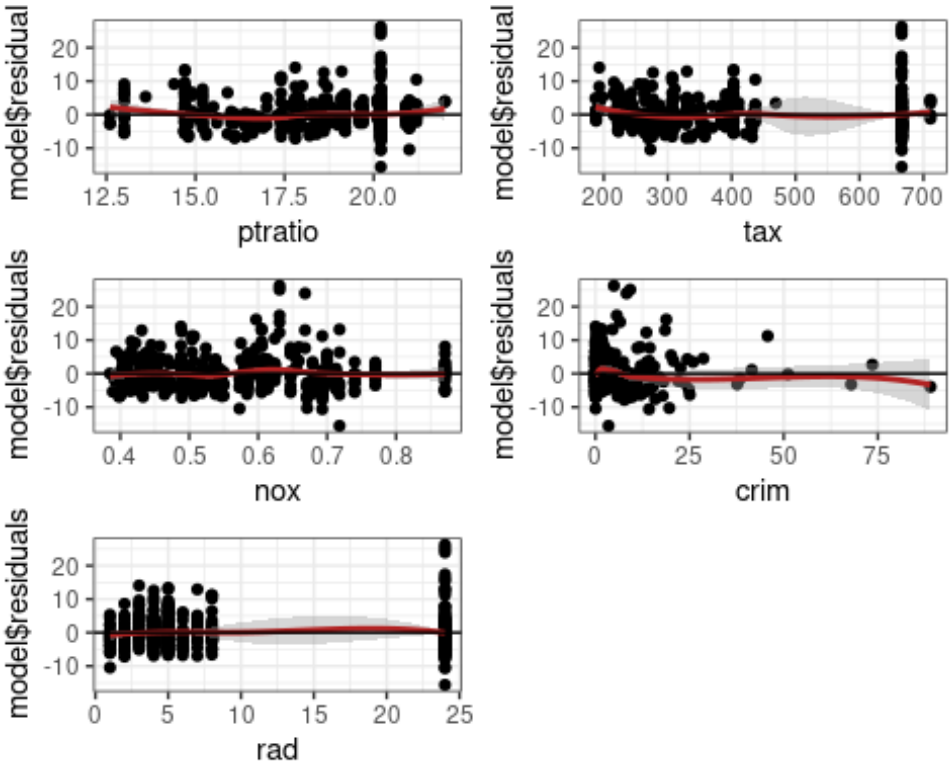
Si para los predictores, los residuos se distribuyen aleatoriamente a lo largo del eje x en un diagrama de dispersión con esos predictores, entonces la relación es lineal.

```
# two first predictors
library(ggplot2)
library(gridExtra)

#lstat
plot_lstat <- ggplot(data = boston, aes(lstat, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()


#rm
plot_rm <- ggplot(data = boston, aes(rm, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

grid.arrange(plot_lstat, plot_rm)
```



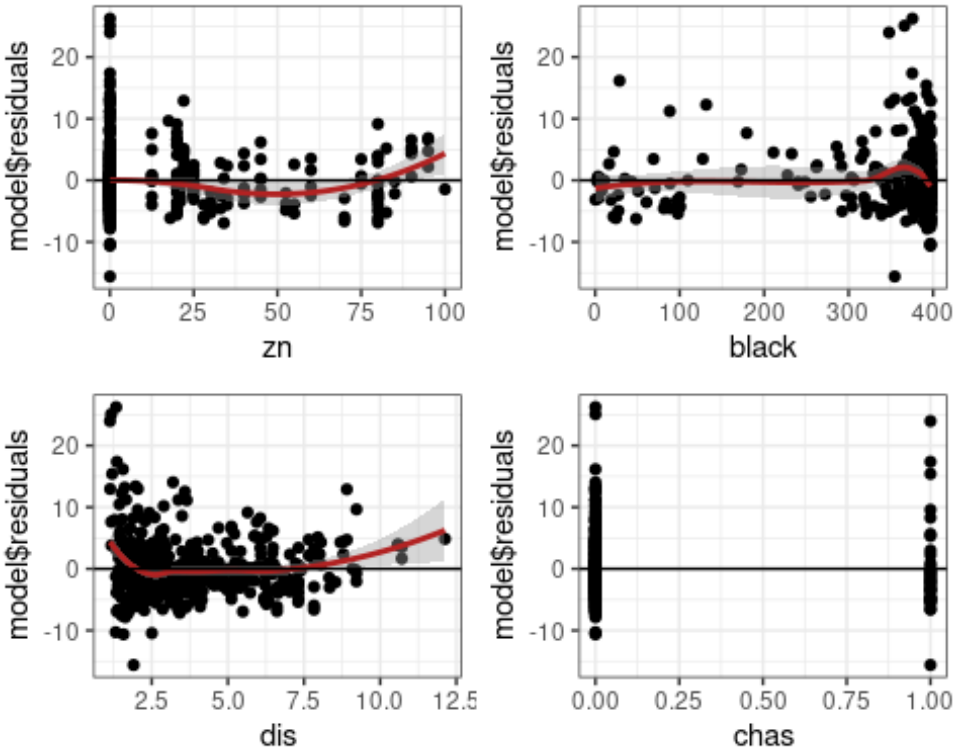
Hay la misma proporción de puntos arriba y abajo del 0. Pero para lstat los puntos se acumulan en la parte izquierda del eje x, y para rm en la parte central. No se reparten a lo largo de todo el eje x de manera aleatoria.

```r
# seven intermediate predictors

#ptratio
plot_ptratio <- ggplot(data = boston, aes(ptratio, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#indus
#plot_indus <- ggplot(data = boston, aes(indus, model$residuals)) +
#geom_point() + geom_smooth(color = "firebrick") +
##geom_hline(yintercept = 0) +
#theme_bw()

#tax
plot_tax <- ggplot(data = boston, aes(tax, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#nox
plot_nox <- ggplot(data = boston, aes(nox, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#crim
plot_crim <- ggplot(data = boston, aes(crim, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#rad
plot_rad <- ggplot(data = boston, aes(rad, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#age
#plot_age <- ggplot(data = boston, aes(age, model$residuals)) +
#geom_point() + geom_smooth(color = "firebrick") +
#geom_hline(yintercept = 0) +
#theme_bw()


#grid.arrange(plot_ptratio, plot_indus, plot_tax, plot_nox, plot_crim, plot_rad,
plot_age)
grid.arrange(plot_ptratio, plot_tax, plot_nox, plot_crim, plot_rad)
```



Para tax, crim y rad, los puntos no se reparten a lo largo del eje x.

```
#last four predictors

#zn
plot_zn <- ggplot(data = boston, aes(zn, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#black
plot_black <- ggplot(data = boston, aes(black, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#dis
plot_dis <- ggplot(data = boston, aes(dis, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()

#chas
plot_chas <- ggplot(data = boston, aes(chas, model$residuals)) +
geom_point() + geom_smooth(color = "firebrick") +
geom_hline(yintercept = 0) +
theme_bw()


grid.arrange(plot_zn, plot_black, plot_dis, plot_chas)
```



zn podría pasar, black y dis no tienen los puntos repartidos a lo largo del eje x. (chas es un caso especial, claro).

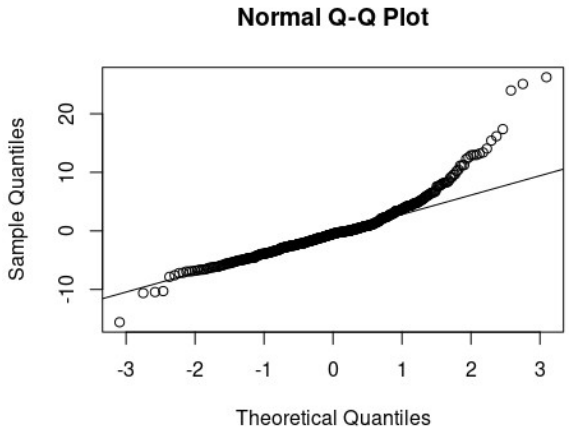
Vamos a examinar ahora los siguientes 3 criterios para los residuos: normalidad, homocedasticidad, y autocorrelación.

1 Normalidad:

Veamos a continuación si los residuos de la regresión se distribuyen según una Normal.

Para ello, realizaremos un gráfico Q-Q y un test Shapiro-Wilk:

```
#Q-Q graph
qqnorm(model$residuals)
qqline(model$residuals)
```



La línea de puntos se separa de la línea recta .. (lo que indica no normalidad).

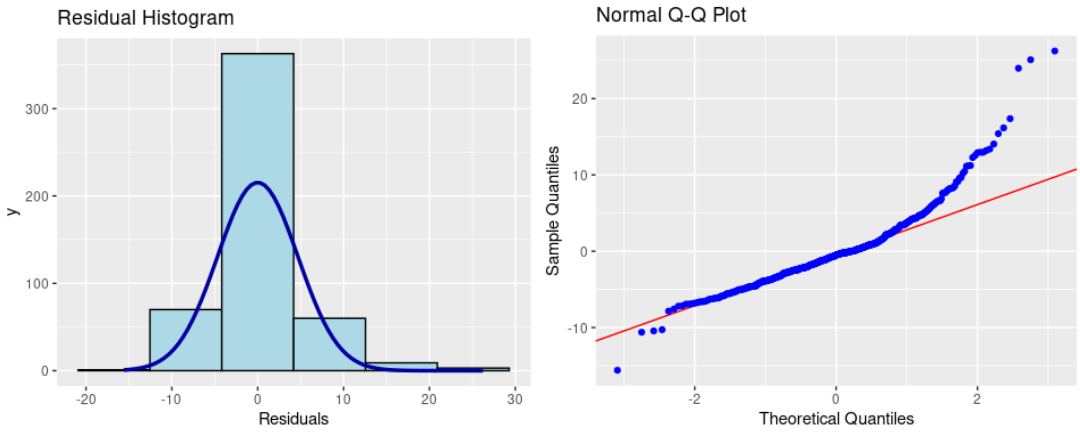
Shapiro-Wilk

```
shapiro.test(model$residuals)
##
##  Shapiro-Wilk normality test
##
## data:  model$residuals
## W = 0.90131, p-value < 2.2e-16
```

Según el test de Shapiro-Wilk, se confirma que los residuos no siguen una distribución normal. (El valor p indica que W es significativo, que al ser superior a 0.05 indica no normalidad).

Otra manera/librería :

```
if (!require(olsrr)) install.packages('olsrr', dependencies = T)
library(olsrr)

ols_plot_resid_hist(model)
ols_plot_resid_qq(model)
```

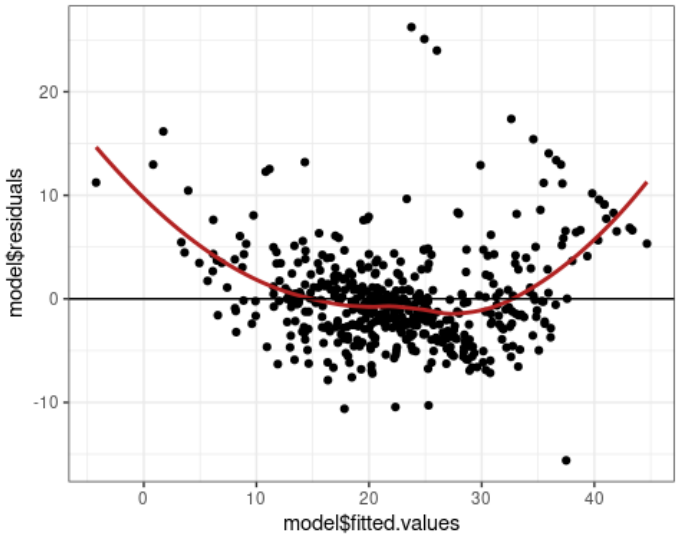


2 Homocedasticidad

Pasamos a revisar la homocedasticidad de los residuos. Para ello, representamos los residuos frente a los valores devueltos por el modelo. Confirmaremos la homocedasticidad si los primeros se distribuyen de forma aleatoria en torno a cero, manteniendo aproximadamente la misma variabilidad a lo largo del eje X.

Si, por el contrario, se observara algún patrón específico significaría que la variabilidad es dependiente del valor ajustado y por lo tanto se violaría el supuesto de homocedasticidad de los residuos:

```
ggplot(data = boston, aes(model$fitted.values, model$residuals)) +
geom_point() +
geom_smooth(color = "firebrick", se = FALSE) +
geom_hline(yintercept = 0) +
theme_bw()
```



Los residuos se acumulan en la zona central, alrededor del 20. No parece que se distribuyan uniformemente a lo largo del eje x. (También me llama mucho la atención una línea recta que va desde las 12 en punto a las 3 en punto).

Breusch-Pagan

```
#library(lmtest)
bptest(model)
##
##  studentized Breusch-Pagan test
##
## data:  model
## BP = 59.907, df = 11, p-value = 9.647e-09
```

El contraste de Breusch-Pagan devuelve un valor muy superior a 0, siendo p muy pequeño, lo que indica la falta de homocedasticidad (y la presencia de heterocedasticidad).

3 Autocorrelación

El análisis de la posible presencia de autocorrelación en los residuos a través del test de **Durbin-Watson**

```
#library(car)
dwt(model, alternative = "two.sided")
##  lag Autocorrelation D-W Statistic p-value
##    1       0.4544515      1.077875       0
##  Alternative hypothesis: rho != 0

# other  library
##library(lmtest)
dwtest(model)
##
##  Durbin-Watson test
##
## data:  model
## DW = 1.0779, p-value < 2.2e-16
## alternative hypothesis: true autocorrelation is greater than 0
```

Pues también se da la presencia de autocorrelación.

Es decir, no tenemos normalidad ni homecedasticidad, y sí autocorrelación … Deduzco que los valores predichos por el modelo no serán de una gran valor.

4 Mejora del modelo

Vamos a ver qué pasa si tenemos en cuenta la posible relación no lineal entre lstat y medv:

```
model_lstat2 <- lm(medv ~ crim + zn + chas + nox + rm + dis + rad + tax +
ptratio + black + lstat + I(lstat^2), data = Boston)

summary(model_lstat2)
##
## Call:
## lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad +
##     tax + ptratio + black + lstat + I(lstat^2), data = Boston)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.5603  -2.6709  -0.3071   1.9469  25.0085
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  42.124187   4.632559   9.093  < 2e-16 ***
## crim         -0.149072   0.030006  -4.968 9.34e-07 ***
## zn            0.021281   0.012500   1.703 0.089291 .
## chas          2.589821   0.775307   3.340 0.000900 ***
## nox         -13.534522   3.229619  -4.191 3.30e-05 ***
## rm            3.233174   0.372802   8.673  < 2e-16 ***
## dis          -1.357892   0.169051  -8.032 7.09e-15 ***
## rad           0.271744   0.057599   4.718 3.11e-06 ***
## tax          -0.009546   0.003068  -3.111 0.001970 **
## ptratio      -0.790820   0.118089  -6.697 5.82e-11 ***
```

```
## black         0.008174   0.002429   3.365 0.000824 ***
## lstat        -1.669567   0.119012 -14.029  < 2e-16 ***
## I(lstat^2)    0.033055   0.003198  10.337  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.298 on 493 degrees of freedom
## Multiple R-squared:  0.7868, Adjusted R-squared:  0.7816
## F-statistic: 151.6 on 12 and 493 DF,  p-value: < 2.2e-16
```

El valor de R-squared pasa del 74% al 78.7%, y el estadístico F pasa de 128 a 151.6 Parece una mejora.

(También parece que zn ha dejado de ser significativo).

A ver qué dice el criterio de Akaike:

```
#?step -> Choose a model by AIC in a Stepwise Algorithm
step(object = model_lstat2, direction = "both", trace = 1)
##
## Start:  AIC=1488.49
## medv ~ crim + zn + chas + nox + rm + dis + rad + tax + ptratio +
##     black + lstat + I(lstat^2)
##
##              Df Sum of Sq     RSS    AIC
## <none>                    9107.3 1488.5
## - zn          1      53.5  9160.8 1489.5
## - tax         1     178.8  9286.1 1496.3
## - chas        1     206.1  9313.4 1497.8
## - black       1     209.2  9316.5 1498.0
## - nox         1     324.4  9431.7 1504.2
## - rad         1     411.2  9518.5 1508.8
## - crim        1     456.0  9563.2 1511.2
## - ptratio     1     828.5  9935.8 1530.5
## - dis         1    1191.9 10299.2 1548.7
## - rm          1    1389.5 10496.7 1558.3
## - I(lstat^2)  1    1974.1 11081.4 1585.8
## - lstat       1    3635.6 12742.8 1656.5
##
##
## Call:
## lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad +
##     tax + ptratio + black + lstat + I(lstat^2), data = Boston)
##
## Coefficients:
## (Intercept)         crim           zn         chas          nox           rm
##   42.124187    -0.149072     0.021281     2.589821   -13.534522     3.233174
##         dis          rad          tax      ptratio        black        lstat
##   -1.357892     0.271744    -0.009546    -0.790820     0.008174    -1.669567
##  I(lstat^2)
##    0.033055
```

Se mantienen todas las variables, incluida zn.

Dejamos como modelo final para hacer el último punto

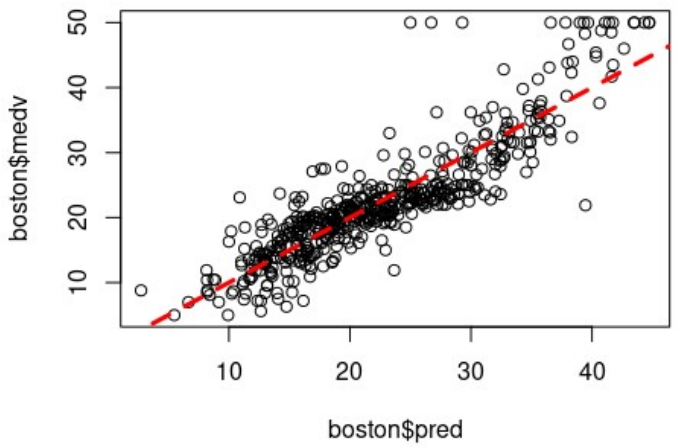
medv ~ crim + zn + chas + nox + rm + dis + rad + tax + ptratio + black + lstat + I(lstat^2)

Vamos a comprobar la correlación entre lo que predice el modelo mejorado, y los datos reales:

```
boston$pred <- predict(model_lstat2, boston)
cor(boston$pred, boston$medv)
## [1] 0.8870152
```

Es un valor bastante alto.

```
plot(boston$pred, boston$medv)
abline(a = 0, b = 1, col = "red", lwd = 3, lty = 2)
```



Los puntos están bastante cerca de la línea roja discontinua que marca donde las predicciones están muy cerca de los valores reales. Quizá el modelo no haga predicciones tan erróneas después de todo.

5 Predicción

En lugar de inventarme 5 registros, he pensado mejor combinar de manera aleatoria parejas de registros existentes. (Como si fueran viviendas que están entre otras 2 viviendas ya evaluadas).

```
boston1And10 <- (boston[1,]+boston[10,])/2

boston15And40 <- (boston[15,]+boston[40,])/2
boston15And40$rad <- 3

boston100And110 <- (boston[100,]+boston[110,])/2
boston100And110$rad <- 5

boston200And220 <- (boston[200,]+boston[220,])/2

boston300And500 <- (boston[300,]+boston[500,])/2
boston300And500$rad <- 5
```

```
#predict(model_lstat2, data.frame(boston1And10))
predict(model_lstat2, data.frame(crim=0.08818 , zn=15.25 , chas=0 , nox=0.531 ,
rm=6.2895 , dis=5.34105 , rad=3 , tax=303.5 , ptratio=15.25 , black=391.805 ,
lstat=11.04 ))
##        1
## 22.98874

predict(model_lstat2, data.frame(boston15And40))
## 25.51835

#predict(model_lstat2, data.frame(boston[100,]))  # -> 33.17257
#predict(model_lstat2, data.frame(boston[110,]))  # -> 18.12436
predict(model_lstat2, data.frame(boston100And110))
## 25.3321

#predict(model_lstat2, data.frame(boston[200,]))  # -> 30.69801
#predict(model_lstat2, data.frame(boston[220,]))  # -> 28.37799
predict(model_lstat2, data.frame(boston200And220))
## 29.24643

#predict(model_lstat2, data.frame(boston[300,]))  # -> 32.43072
#predict(model_lstat2, data.frame(boston[500,]))  # -> 17.2025
predict(model_lstat2, data.frame(boston300And500))
## 23.79381
```

Conclusión final:

Los resultados me parecen “razonables”, a pesar de lo comentado (predictores con muchos valores atípicos, variables categóricas(?) .. predictor racista* ...).

*Al investigar sobre el tema, he encontrado críticas sobre este data set en internet.

Parece que este dataset también estaba en sckit-learn, y en la versión 1.2 lo quitaron. Al menos al hacer esta tarea con Python, me sale un mensaje avisando de esta circunstancia, como se puede ver en el repositorio de GitHub).

Nota sobre el código en Python:

Es la tarea hasta ahora en la que encuentro los resultados más parecidos. En este caso no es que sean muy muy parecidos, es que son idénticos quitando la precisión decimal (los valores de R^2, los coeficientes .. la correlación de los residuos, las predicciones para los 5 valores del final).

Por eso no incluyo ninguna comparativa de resultados aquí como en los casos anteriores. Sí incluyo el enlace al repositorio en GitHub en el anexo.

(Ha sido un poco más complicado que en R, porque he tenido que recurrir a otras librerías, no solo a scikit-learn).

EVALUACIÓN TEMA 7 - MÉTODOS BLACK BOX: REDES NEURONALES Y MÁQUINAS DE VECTOR SOPORTE

Nota: El alumno deberá presentar todo el código utilizado para resolver el ejercicio.

1. La base de datos incluida en el archivo Bank.csv (dentro de Bank.zip) recoge información de 4.521 clientes a los que se les ofreció contratar un depósito a plazo en una entidad bancaria portuguesa (el zip también contiene un fichero de texto denominado Bank-names.txt con el detalle completo de todas las variables incluidas)

   Utilizando dicha base de datos, elabore una red neuronal que permita pronosticar si, en base a sus características, el cliente contratará el depósito o no.

   De cara a la realización de este ejercicio, debe tener en cuenta que:

   - La variable objetivo de nuestro modelo es “y”, la cual tiene el valor “yes” si el cliente ha contratado el depósito y “no” en caso contrario.
   - Observe que hay múltiples variable de tipo cualitativo que deberá transformar antes de estimar el modelo.
   - No olvide normalizar los datos antes de introducirlos en el modelo.
   - Recuerde especificar el número de capas ocultas y neuronas utilizadas, así como el umbral de error permitido y el algoritmo de cálculo elegidos. Se permite realizar y presentar variaciones del modelo a fin de obtener un ajuste óptimo.
   - Deberá dejar un porcentaje del *dataset* para validar los resultados de la red neuronal estimada.

2. Repita el ejercicio anterior utilizando ahora una Máquina de Vector Soporte para estimar el modelo. No olvide dejar un porcentaje del *dataset* para validar los resultados.

3. Realice una comparación detallada de los resultados obtenidos por ambos modelos.

1 Redes de neuronas

1 Paso 1: Carga de los datos

```
# import the CSV file
bank_raw <- read.csv(file.path("Chapter07/Bank", "bank.csv"), sep = ";",
stringsAsFactors = TRUE)
```

2 Paso 2: Explorar y preparar los datos

Carga de paquetes que son necesarios para diversas funciones.

```
if (library=="neuralnet") {
  print("Choosing neuralnet")

  if (!require(neuralnet)) install.packages('neuralnet', dependencies = T)
  library(neuralnet)

} else if (library=="RSNNS") {
  print("Choosing RSNNS")

  # Downloading packages ---------------------------------------------------
  if (!require(RSNNS)) install.packages('RSNNS', dependencies = T)
  library(RSNNS)

} else {
  print("Choosing Keras")

  if (!require(keras3)) install.packages('keras3', dependencies = T)
  library(keras3)
  #install_keras()

  if (!require(tidyverse)) install.packages('tidyverse', dependencies = T)
  library(tidyverse)

  if (!require(jsonlite)) install.packages('jsonlite', dependencies = T)
  library(jsonlite)
}

if (!require(caret)) install.packages('caret', dependencies = T)
library(caret)

if (!require(ggplot2)) install.packages('ggplot2', dependencies = T)
library(ggplot2)
```

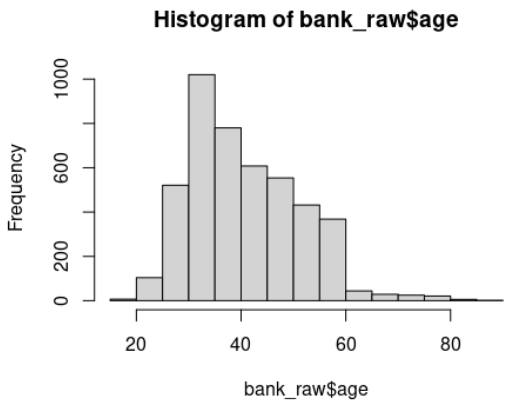
Examinamos la estructura y el aspecto del fichero importado:

```
#See the structure
str(bank_raw)

## 'data.frame':    4521 obs. of  17 variables:
##  $ age       : int  30 33 35 30 59 35 36 39 41 43 ...
##  $ job       : Factor w/ 12 levels "admin.","blue-collar",..: 11 8 5 5 2 5 7 10 3 8 ...
##  $ marital   : Factor w/ 3 levels "divorced","married",..: 2 2 3 2 2 3 2 2 2 2 ...
##  $ education: Factor w/ 4 levels "primary","secondary",..: 1 2 3 3 2 3 3 2 3 1 ...
##  $ default   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ balance   : int  1787 4789 1350 1476 0 747 307 147 221 -88 ...
##  $ housing   : Factor w/ 2 levels "no","yes": 1 2 2 2 2 1 2 2 2 2 ...
##  $ loan      : Factor w/ 2 levels "no","yes": 1 2 1 2 1 1 1 1 1 2 ...
##  $ contact   : Factor w/ 3 levels "cellular","telephone",..: 1 1 1 3 3 1 1 1 3 1 ...
##  $ day       : int  19 11 16 3 5 23 14 6 14 17 ...
##  $ month     : Factor w/ 12 levels "apr","aug","dec",..: 11 9 1 7 9 4 9 9 9 1 ...
```

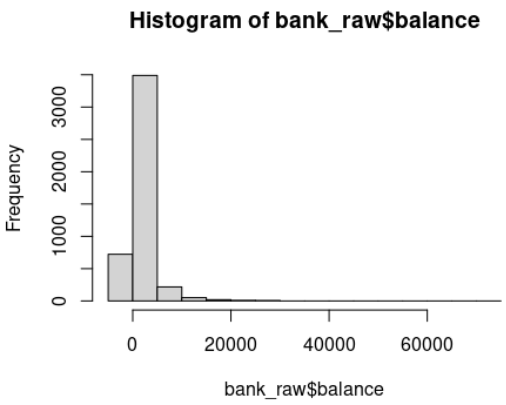
```
##  $ duration : int  79 220 185 199 226 141 341 151 57 313 ...
##  $ campaign : int  1 1 1 4 1 2 1 2 2 1 ...
##  $ pdays    : int  -1 339 330 -1 -1 176 330 -1 -1 147 ...
##  $ previous : int  0 4 1 0 0 3 2 0 0 2 ...
##  $ poutcome : Factor w/ 4 levels "failure","other",..: 4 1 1 4 4 1 2 4 4 1 ...
##  $ y        : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...

#Summary
summary(bank_raw)

##       age                job          marital         education    default
##  Min.   :19.00   management :969   divorced: 528   primary  : 678   no :4445
##  1st Qu.:33.00   blue-collar:946   married :2797   secondary:2306   yes:  76
##  Median :39.00   technician :768   single  :1196   tertiary :1350
##  Mean   :41.17   admin.     :478                   unknown  : 187
##  3rd Qu.:49.00   services   :417
##  Max.   :87.00   retired    :230
##                  (Other)    :713
##     balance      housing     loan           contact          day
##  Min.   :-3313   no :1962   no :3830   cellular :2896   Min.   : 1.00
##  1st Qu.:   69   yes:2559   yes: 691   telephone: 301   1st Qu.: 9.00
##  Median :  444                         unknown  :1324   Median :16.00
##  Mean   : 1423                                          Mean   :15.92
##  3rd Qu.: 1480                                          3rd Qu.:21.00
##  Max.   :71188                                          Max.   :31.00
##
##      month         duration       campaign          pdays
##  may    :1398   Min.   :   4   Min.   : 1.000   Min.   : -1.00
##  jul    : 706   1st Qu.: 104   1st Qu.: 1.000   1st Qu.: -1.00
##  aug    : 633   Median : 185   Median : 2.000   Median : -1.00
##  jun    : 531   Mean   : 264   Mean   : 2.794   Mean   : 39.77
##  nov    : 389   3rd Qu.: 329   3rd Qu.: 3.000   3rd Qu.: -1.00
##  apr    : 293   Max.   :3025   Max.   :50.000   Max.   :871.00
##  (Other): 571
##     previous          poutcome      y
##  Min.   : 0.0000   failure: 490   no :4000
##  1st Qu.: 0.0000   other  : 197   yes: 521
##  Median : 0.0000   success: 129
##  Mean   : 0.5426   unknown:3705
##  3rd Qu.: 0.0000
##  Max.   :25.0000
##

#see some records
head(bank_raw,5)

##   age         job marital education default balance housing loan  contact day
## 1  30  unemployed married   primary      no    1787      no   no cellular  19
## 2  33    services married secondary      no    4789     yes  yes cellular  11
## 3  35  management  single  tertiary      no    1350     yes   no cellular  16
## 4  30  management married  tertiary      no    1476     yes  yes  unknown   3
## 5  59 blue-collar married secondary      no       0     yes   no  unknown   5
##   month duration campaign pdays previous poutcome  y
## 1   oct       79        1    -1        0  unknown no
## 2   may      220        1   339        4  failure no
## 3   apr      185        1   330        1  failure no
## 4   jun      199        4    -1        0  unknown no
## 5   may      226        1    -1        0  unknown no
```

Examinamos ahora las variables numéricas, para ver qué conviene más, si una normalización, o una estandarización.

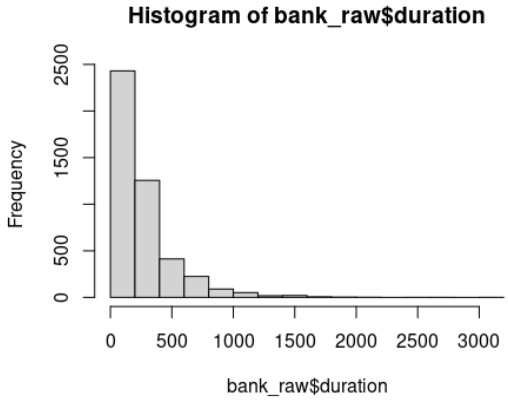
```r
hist(bank_raw$age)
```



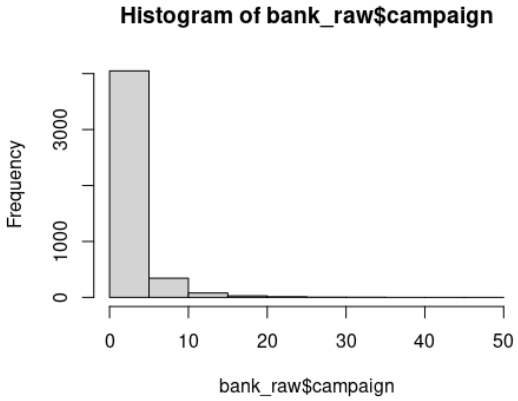

```r
hist(bank_raw$balance)
```



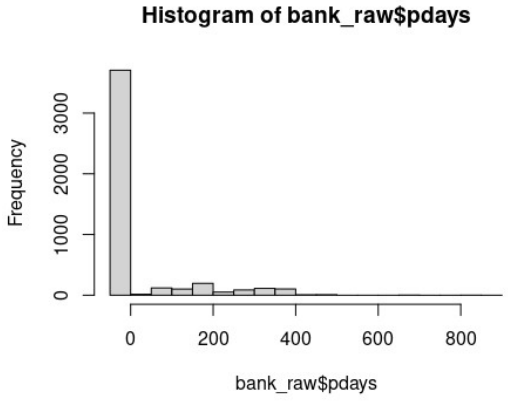

```r
hist(bank_raw$duration)
```



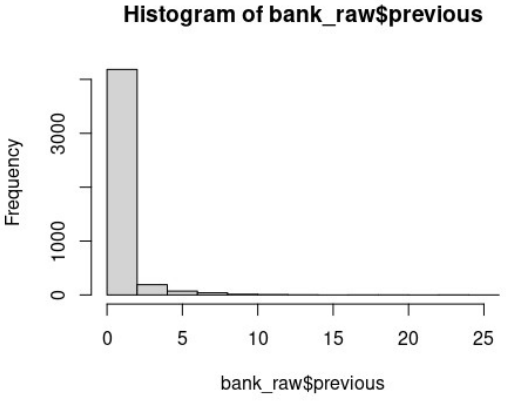

```r
hist(bank_raw$campaign)
```




```r
hist(bank_raw$pdays)
```




```r
hist(bank_raw$previous)
```



La única variable que se aproxima a la distribución normal es la edad. Ninguna se aproxima a la uniforme.

Así que normalizamos las variables numéricas de 0 a 1 con la ayuda de la función scale. (No se normalizan ni los días ni los meses).

```
#scale numeric variables
maxs <- apply(bank_raw[c(1,6,12,13,14,15)], 2, max)
mins <- apply(bank_raw[c(1,6,12,13,14,15)], 2, min)

bank_norm <- data.frame(scale(bank_raw[c(1,6,12,13,14,15)], center = mins, scale = maxs - mins))

#Summary
summary(bank_norm)

##       age            balance          duration         campaign
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.2059   1st Qu.:0.04540   1st Qu.:0.03310   1st Qu.:0.00000
##  Median :0.2941   Median :0.05043   Median :0.05991   Median :0.02041
##  Mean   :0.3260   Mean   :0.06356   Mean   :0.08605   Mean   :0.03660
##  3rd Qu.:0.4412   3rd Qu.:0.06433   3rd Qu.:0.10758   3rd Qu.:0.04082
##  Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000

##      pdays            previous
##  Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000
##  Median :0.00000   Median :0.0000
##  Mean   :0.04675   Mean   :0.0217
##  3rd Qu.:0.00000   3rd Qu.:0.0000
##  Max.   :1.00000   Max.   :1.0000
```

Ahora debemos transformar las variables categóricas en numéricas (“hot encoding”). La variable “month” he pensado transformarla en una sola variable: Enero -> 1, Febrero -> 2 … Utilizar “hot enconding” con esta variable me parece que es añadir demasiadas variables sin necesidad. (He leído que a las redes neuronales no les van demasiado bien las matrices dispersas. Añadir 12 columnas para los meses, llenas de 0s, no me parece buena idea).

```
#hot encoding of categorical features
dummies <- dummyVars(" ~ job + marital + education + default + housing + loan + contact + poutcome +
y", data = bank_raw)  # y for neuralnet and RSNNS

bank_hot_encoded_feat <- data.frame(predict(dummies, newdata = bank_raw))

head(bank_hot_encoded_feat,5)
##
##   job.admin. job.blue.collar job.entrepreneur job.housemaid job.management
## 1          0               0                0             0              0
## 2          0               0                0             0              0
## 3          0               0                0             0              1
## 4          0               0                0             0              1
## 5          0               1                0             0              0
##   job.retired job.self.employed job.services job.student job.technician
## 1           0                 0            0           0              0
## 2           0                 0            1           0              0
## 3           0                 0            0           0              0
## 4           0                 0            0           0              0
## 5           0                 0            0           0              0
##   job.unemployed job.unknown marital.divorced marital.married marital.single
## 1              1           0                0               1              0
## 2              0           0                0               1              0
## 3              0           0                0               0              1
## 4              0           0                0               1              0
## 5              0           0                0               1              0
##   education.primary education.secondary education.tertiary education.unknown
## 1                 1                   0                  0                 0
## 2                 0                   1                  0                 0
## 3                 0                   0                  1                 0
## 4                 0                   0                  1                 0
## 5                 0                   1                  0                 0
##   default.no default.yes housing.no housing.yes loan.no loan.yes
```

```
## 1          1           0          1           0       1        0
## 2          1           0          0           1       0        1
## 3          1           0          0           1       1        0
## 4          1           0          0           1       0        1
## 5          1           0          0           1       1        0
##   contact.cellular contact.telephone contact.unknown poutcome.failure
## 1                1                 0               0                 0
## 2                1                 0               0                 1
## 3                1                 0               0                 1
## 4                0                 0               1                 0
## 5                0                 0               1                 0
##   poutcome.other poutcome.success poutcome.unknown y.no y.yes
## 1              0                 0                 1    1     0
## 2              0                 0                 0    1     0
## 3              0                 0                 0    1     0
## 4              0                 0                 1    1     0
## 5              0                 0                 1    1     0
```

Transformamos los meses en una variable numérica.

```
#encoding month (name to number)

#unique(bank_raw$month)  -> Levels: apr aug dec feb jan jul jun mar may nov oct sep

month_to_number <- function(month_name) {
  month_and_number <- c("jan"=1,"feb"=2,  "mar"=3, "apr"=4, "may"=5, "jun"=6, "jul"=7, "aug"=8,
"sep"=9, "oct"=10, "nov"=11, "dec"=12)
  return(month_and_number[as.character(month_name)])
}

#tests
month_to_number("oct")
## oct
##  10
month_to_number("may")
## may
##   5
test <- bank_raw$month[1:5]
test
## [1] oct may apr jun may
## Levels: apr aug dec feb jan jul jun mar may nov oct sep
result <-sapply(test, month_to_number)
result
## oct may apr jun may
##  10   5   4   6   5

bank_raw$month_num <- sapply(bank_raw$month, month_to_number)


head(bank_raw,5)
...
##   month duration campaign pdays previous poutcome  y month_num
## 1   oct       79        1    -1        0  unknown no        10
## 2   may      220        1   339        4  failure no         5
## 3   apr      185        1   330        1  failure no         4
## 4   jun      199        4    -1        0  unknown no         6
## 5   may      226        1    -1        0  unknown no         5

#transform target categorical feature (keras)
dummy_y <- fastDummies::dummy_cols(bank_raw$y,remove_first_dummy = TRUE)

head(dummy_y)
##   .data .data_yes
## 1    no         0
## 2    no         0
## 3    no         0
## 4    no         0
## 5    no         0
## 6    no         0
```

Juntamos todas las variables en un mismo dataframe.

```
bank_processed <- cbind(bank_norm,as.numeric(bank_raw$day), bank_raw$month_num,
bank_hot_encoded_feat, dummy_y$.data_yes)
names(bank_processed)[7:8] <- c("day","month")
names(bank_processed)[43] <- c("y")
head(bank_processed,5)
# output omitted for brevity
```

Finalmente, creamos los conjuntos de entrenamiento y validación:

```
#Set seed to make the process reproducible
set.seed(9)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(bank_processed$y, times=1, p=.75,
list=FALSE)

#create training set
bank_processed_train <- bank_processed[train_indices, ]

#create testing set
bank_processed_test  <- bank_processed[-train_indices, ]


if (library == "keras") {
  X_train_bank <- bank_processed_train %>%
    select(-y,-y.yes,-y.no) %>%
    keras3::as_tensor(dtype = "float32")

  y_train_bank <- keras3::to_categorical(bank_processed_train$y)

  X_test_bank <- bank_processed_test %>%
    select(-y,-y.yes,-y.no) %>%
    keras3::as_tensor(dtype = "float32")

  y_test_bank <- keras3::to_categorical(bank_processed_test$y)

} else {
  X_train_bank <- bank_processed_train[ , -c(41,42,43)]
  y_train_bank <- bank_processed_train[ , c(41,42)]

  X_test_bank <- bank_processed_test[ , -c(41,42,43)]
  y_test_bank <- bank_processed_test[ , c(41,42)]
}


#view number of rows in each set
#nrow(X_train_bank)  # 3391
#nrow(X_test_bank)   # 1130
#nrow(y_train_bank)  # 3391
#nrow(y_test_bank)   # 1130
```

3 Paso 3: Entrenamiento del modelo

Además de las librerías mencionadas en la literatura, he descubierto que también es posible utilizar en R las librerías Keras y TensorFlow (con PyTorch, creo que son las librerías más utilizadas hoy en día para deep learning). Por eso he modificado el código para ir añadiendo además de neuralnet, RSNNS, y luego Keras. Puede que quede así un poco más farragoso, pero he querido hacerlo de esta manera para aprovechar y aprender alguna cosa más de .Rmd, como los parámetros:

```
---
title: "Tema7_Ejercicio_redes_neuronas"
author: "Fran Camacho"
date: "2025-02-24"
output: word_document
params:
  library: "keras"  #options: "neuralnet", "RSNNS", "keras"
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)

library <- params$library
print(library)
```
```

```
# softplus function
softplus <- function(x) { log(1 + exp(x)) }
set.seed(9)


if (library=="neuralnet") {
  print("Choosing neuralnet")

  system.time({
    model <- neuralnet(y.yes+y.no ~ .,
                data = bank_processed_train[ , -which(names(bank_processed_train) %in% c("y"))],
                hidden = 40, threshold = 0.5, lifesign="full")
                #act.fct = softplus, threshold = 0.01, algorithm = "backprop", learningrate=0.05
  })

} else if  (library=="RSNNS") {
  print("Choosing RSNNS")

  system.time({
      model <- mlp(X_train_bank, y_train_bank, size = c(40,10,4,2),
                       learnFuncParams = c(0.05), maxit = 20)
                      # if hiddenActFunc=softplus, it never ends ...
  })

} else { #Keras
  print("Choosing Keras")

  model <- keras_model_sequential(name = "keras_mid_complex", input_shape = ncol(X_train_bank))


  model %>%
    layer_dense(name = "dense_1",units = 40, activation = 'relu') %>%
    layer_dropout(name = "droput_1", rate = 0.8) %>%
    layer_dense(name = "dense_2",units = 10, activation = 'relu') %>%
    layer_dropout(name = "droput_2", rate = 0.4) %>%
    layer_dense(name = "output_layer", units = 2, activation = 'sigmoid')
```

```
  model %>% compile(
    optimizer = "adam",
    loss = "binary_crossentropy",
    metrics = 'accuracy'
  )

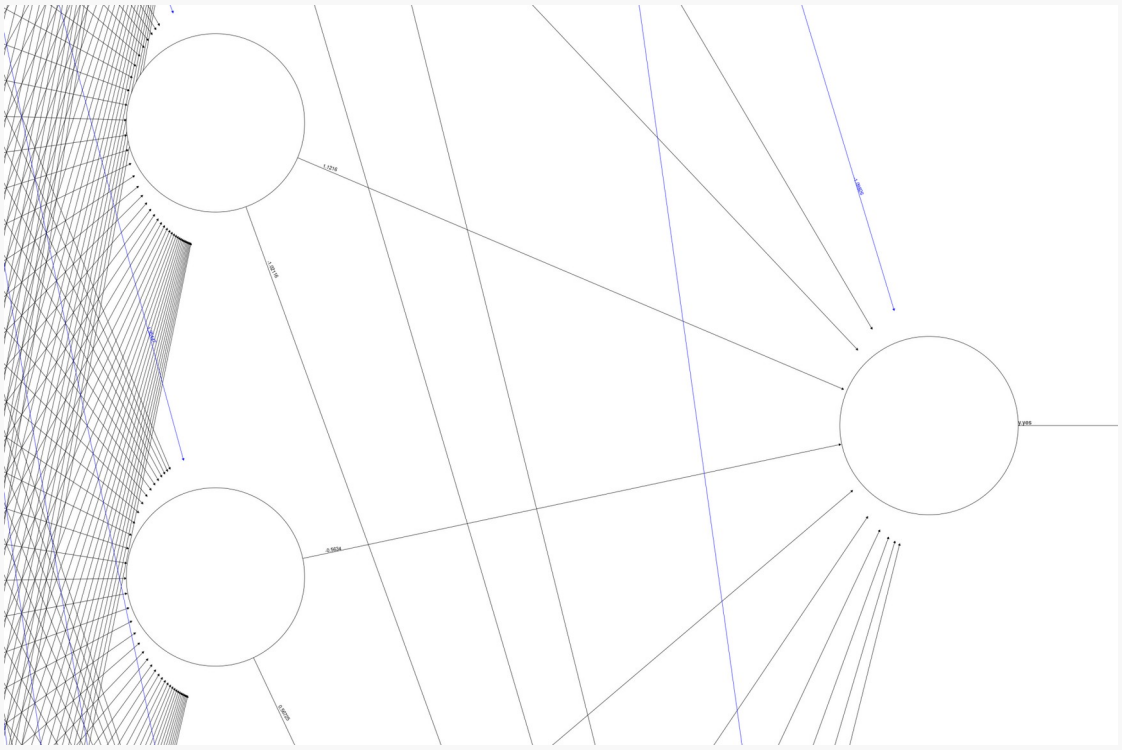
  #Training
  system.time({
    history <- model %>% fit(
      X_train_bank, y_train_bank,
      epochs = 1000,
      batch_size = 40,
      validation_split = 0.2
    )
  })

}
```

Visualizamos la arquitectura de la red entrenada y sus pesos:

```
# neuralnet
if (library=="neuralnet") {
  plot(model)   #saved in file "Chapter07/neuralnet_10_neurons_model.png"
}

# Detail of 3 neurons. To see the weights, you must zoom really a lot
```



```
if (library=="keras") {
  model
}
```

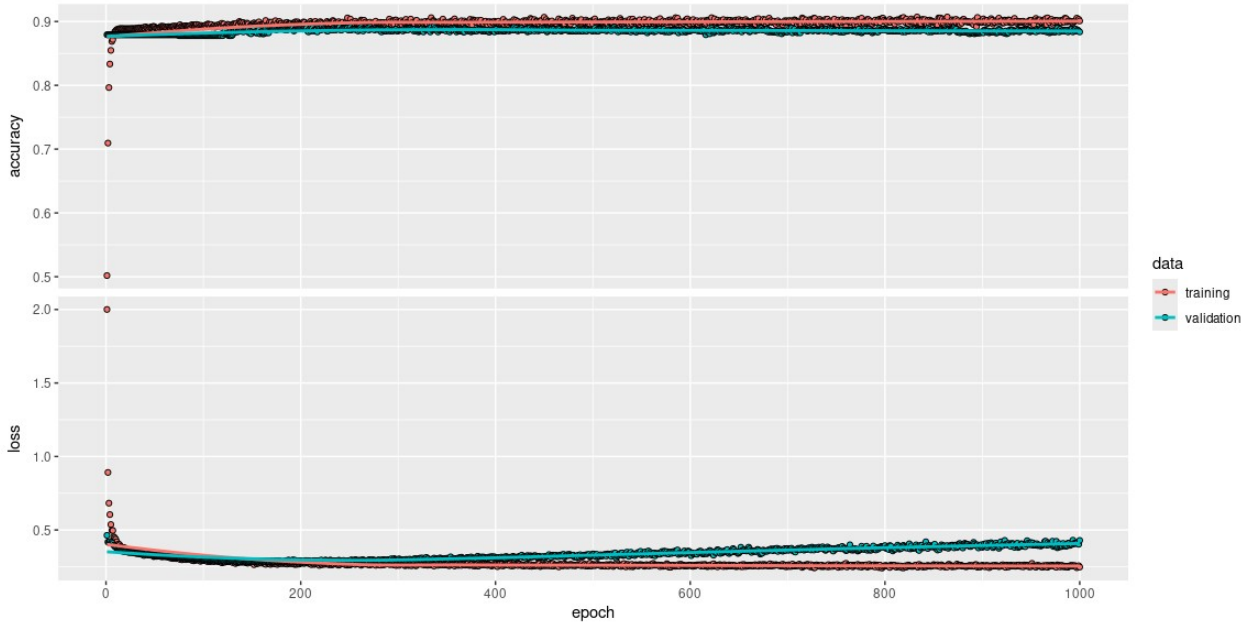
```
## Model: "keras_mid_complex"
## ┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
## ┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
## ┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
## │ dense_1 (Dense)                 │ (None, 40)             │         1,640 │
## ├─────────────────────────────────┼────────────────────────┼───────────────┤
## │ droput_1 (Dropout)              │ (None, 40)             │             0 │
## ├─────────────────────────────────┼────────────────────────┼───────────────┤
## │ dense_2 (Dense)                 │ (None, 10)             │           410 │
## ├─────────────────────────────────┼────────────────────────┼───────────────┤
## │ droput_2 (Dropout)              │ (None, 10)             │             0 │
## ├─────────────────────────────────┼────────────────────────┼───────────────┤
## │ output_layer (Dense)            │ (None, 2)              │            22 │
## └─────────────────────────────────┴────────────────────────┴───────────────┘
##  Total params: 6,218 (24.29 KB)
##  Trainable params: 2,072 (8.09 KB)
##  Non-trainable params: 0 (0.00 B)
##  Optimizer params: 4,146 (16.20 KB)
```

Haciendo un curso de redes de neuronas en Coursera, he visto suele ser conveniente añadir capas “dropout“ (abandono) para intentar reducir el sobreajuste al que tienden las redes de neuronas. Las neuronas de estas capas ponen a 0 aleatoriamente algunas salidas de las neuronas de la capa anterior (como si las apagaran), y de esta manera se reduce la co-adaptación (problema que consiste en que muchas neuronas, al tener pesos parecidos, obtienen de los datos las mismas características ocultas), y por tanto el sobreajuste.

En Keras, al entrenar un modelo, se puede guardar el histórico del entrenamiento (`history <- model %>% fit(..)`). Es decir, los valores de exactitud, pérdida .. para cada epoch.
Esto puede dar información sobre la velocidad de convergencia según avanzan las “epochs“, cuándo ha convergido el modelo ..

```
if (library=="keras") {
  plot(history)
}
```

(Página siguiente)



(Ahora veo que no hace faltan 1000 epochs para que el modelo converga. Voy aprendiendo sobre la marcha, claro).

También en el caso de utilizar las librerías Keras&TensorFlow, se puede obtener un JSON con información del modelo:

(Por brevedad se muestran solamente las 3 primeras capas).

```
if (library=="keras") {
  #We can print a json with the info of the model:
  prettify(keras::model_to_json(model))
}
```

```
## {
##     "module": "keras",
##     "class_name": "Sequential",
##     "config": {
##         "name": "keras_mid_complex",
##         "trainable": true,
##         "dtype": {
##             "module": "keras",
##             "class_name": "DTypePolicy",
##             "config": {
##                 "name": "float32"
##             },
##             "registered_name": null
##         },
##         "layers": [
##             {
##                 "module": "keras.layers",
##                 "class_name": "InputLayer",
##                 "config": {
##                     "batch_shape": [
##                         null,
##                         40
##                     ],
##                     "dtype": "float32",
##                     "sparse": false,
##                     "name": "input_layer"
##                 },
##                 "registered_name": null
##             },
##             {
```

```
##                 "module": "keras.layers",
##                 "class_name": "Dense",
##                 "config": {
##                     "name": "dense_1",
##                     "trainable": true,
##                     "dtype": {
##                         "module": "keras",
##                         "class_name": "DTypePolicy",
##                         "config": {
##                             "name": "float32"
##                         },
##                         "registered_name": null
##                     },
##                     "units": 40,
##                     "activation": "relu",
##                     "use_bias": true,
##                     "kernel_initializer": {
##                         "module": "keras.initializers",
##                         "class_name": "GlorotUniform",
##                         "config": {
##                             "seed": null
##                         },
##                         "registered_name": null
##                     },
##                     "bias_initializer": {
##                         "module": "keras.initializers",
##                         "class_name": "Zeros",
##                         "config": {
##
##                         },
##                         "registered_name": null
##                     },
##                     "kernel_regularizer": null,
##                     "bias_regularizer": null,
##                     "kernel_constraint": null,
##                     "bias_constraint": null
##                 },
##                 "registered_name": null,
##                 "build_config": {
##                     "input_shape": [
##                         null,
##                         40
##                     ]
##                 }
##             },
##             {
##                 "module": "keras.layers",
##                 "class_name": "Dropout",
##                 "config": {
##                     "name": "droput_1",
##                     "trainable": true,
##                     "dtype": {
##                         "module": "keras",
##                         "class_name": "DTypePolicy",
##                         "config": {
##                             "name": "float32"
##                         },
##                         "registered_name": null
##                     },
##                     "rate": 0.8,
##                     "seed": null,
##                     "noise_shape": null
##                 },
##                 "registered_name": null
##             },
…
```

4 Paso 4: Evaluación de los modelos

Una vez entrenados los modelos, pasamos a analizar su capacidad predictiva.

```
if (library=="neuralnet") {
  #prediction <- compute(model, bank_processed_test[ , -
which(names(bank_processed_test) %in% c("y"))])  #compute is deprecated, so we
use predict
  predictions <- predict(model,
                 bank_processed_test[ , -which(names(bank_processed_test)
                       %in% c("y", "y.yes", "y.no"))])

} else if (library=="RSNNS") {

  predictions <- predict(model, X_test_bank)

} else {  #Keras

  predictions <- model %>% predict(X_test_bank)

}
```

Matriz de confusión:

Tranformamos tanto las predicciones como los valores reales (test) en vectores de “yes/no“ para poder pasarlos a la función confusionMatrix:

```
# neuralnet, RSNNS, keras

#find which of the two columns in predictions has the highest value
prediction <- apply(predictions,1,which.max)

#and translate that to one of the two possible values
prediction[prediction==1] <- "no"
prediction[prediction==2] <- "yes"

y_test_bank_real <- apply(y_test_bank,1,which.max)

y_test_bank_real[y_test_bank_real==1] <- "no"
y_test_bank_real[y_test_bank_real==2] <- "yes"


# Confussion matrix

caret::confusionMatrix(as.factor(y_test_bank_real), as.factor(prediction),
positive="yes", mode = "everything")
```

He realizado pruebas con las 3 librerías.

Con neuralnet, aunque se supone que era la más sencilla, es con la que más problemas tuve. Quizá valga para un ejemplo como el del libro con un conjunto de datos tan pequeño, pero con fichero .csv de este ejercicio, he tenido como digo muchos problemas. Mensajes por ejemplo diciendo que el algoritmo no convergía .. otras veces la ejecución parecía que iba a tardar horas (sobre todo al intentar utilizar la función ReLu/Softmax). Y cuando obtenía algún resultado .. la exactitud era del 15% (). Pensé que estaba haciendo algo mal.

En cuanto pasé a utilizar la librería RSNNS, obtuve resultados mucho mejores. Resultados “normales“. Enseguida obtuve valores para “accuracy“ que se acercaban al 90%, sin tener que investigar demasiado.

Fuí haciendo pruebas entonces con RSNNS con diferentes configuraciones: una capa, 2 capas .. y más.

Luego probé también con Keras, y obtuve prácticamente los mismos resultados que con RSNNS.

En ambos casos, he obtenido valores de precisión que llegan fácilmente al 88 u 89% .. pero no he conseguido pasar de ese muro*.

Es como si la información contenida en el conjunto de datos no permitiera alcanzar un mejor resultado.

* Al hacer posteriormente la versión de este ejercicio con SVM en R, y luego también en Python con Keras y SVM, me he encontrado con el mismo muro. Solo en 2 ocasiones -con Keras en Python- he llegado a obtener una exactitud del 90%, y otra del 90.19%.

Son diferencias muy pequeñas, pero me ha llamado la atención que haya sido tan fácil llegar a una exactitud entre el 88 y el 89 %, que me haya sido tan difícil llegar al 90%, por eso hablo de “muro“.

2 SVM

Se muestra a continuación el código de la solución utilizando “support vector machines“.
He utilizado la librería “kernlab“, y también he probado con la librería LIBSVM. (He obtenido resultados prácticamente iguales).

```
#https://www.jstatsoft.org/article/view/v011i09
if (!require(kernlab)) install.packages('kernlab', dependencies = T)
library(kernlab)

if (!require(caret)) install.packages('caret', dependencies = T)
library(caret)

# LIBSVM   https://www.csie.ntu.edu.tw/~cjlin/libsvm/
if (!require(e1071)) install.packages('e1071', dependencies = T)
library(e1071)
```

Se omiten el paso 1 y el paso 2, ya que se trata del mismo código que para las redes de neuronas.

1 Paso 3: Entrenamiento del modelo

Vamos a comparar dos kernels de la librería kernlab.

```
#train the model vanilladot
model_vanilladot <- ksvm(y ~ ., data=bank_processed_train, kernel="vanilladot")
model_vanilladot
```

```
##  Setting default kernel parameters model_vanilladot
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 983
##
## Objective Function Value : -718.0508
## Training error : 0.105868
```

```
#train the model rbfdot
model_rbfdot <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot")
model_rbfdot
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0177605577375513
##
## Number of Support Vectors : 949
##
## Objective Function Value : -672.7457
## Training error : 0.093483
```

El error es ligeramente más pequeño con el kernel rbf (“radial basis function”).

(Como digo también he probado con el paquete e1071. Como no mejora los resultados de kernlab, no incluyo el código ni los resultados en este documento).

2 Paso 4 – Evaluación del modelo

Con el modelo entrenado podemos realizar predicciones usando el dataset de validación y obtener así la pertinente matriz de confusión:

```
#Confusion matrix vanilladot
prediction_vanilladot <- predict(model_vanilladot, bank_processed_test)

confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_vanilladot), positive="yes",
mode = "everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  987  13
##        yes 112  18
##
##                Accuracy : 0.8894
##                  95% CI : (0.8696, 0.9071)
##     No Information Rate : 0.9726
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1876
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.58065
##             Specificity : 0.89809
##          Pos Pred Value : 0.13846
##          Neg Pred Value : 0.98700
##               Precision : 0.13846
##                  Recall : 0.58065
##                      F1 : 0.22360
##              Prevalence : 0.02743
##          Detection Rate : 0.01593
##    Detection Prevalence : 0.11504
##       Balanced Accuracy : 0.73937
##
##        'Positive' Class : yes
##

#Confusion matrix rbfdot
prediction_rbfdot <- predict(model_rbfdot, bank_processed_test)

confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_rbfdot), positive="yes", mode
= "everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  988  12
##        yes 110  20
##
##                Accuracy : 0.892
##                  95% CI : (0.8725, 0.9095)
##     No Information Rate : 0.9717
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2111
```

```
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.62500
##             Specificity : 0.89982
##          Pos Pred Value : 0.15385
##          Neg Pred Value : 0.98800
##               Precision : 0.15385
##                  Recall : 0.62500
##                      F1 : 0.24691
##              Prevalence : 0.02832
##          Detection Rate : 0.01770
##    Detection Prevalence : 0.11504
##       Balanced Accuracy : 0.76241
##
##        'Positive' Class : yes
##
```

Al ser el resultado del kernel rfb mejor, aunque sea por milésimas, lo elegimos para intentar mejorar el modelo.

3 Paso 5 – Mejora del modelo

Como se explica en el libro, vamos a intentar averiguar si con algún valor del parámetro coste (parámetro C en la función ksvm), se puede obtener una exactitud mejor que con el valor por defecto (C=1):

```
set.seed(12345)

cost_values <- c(1, seq(from = 5, to = 50, by = 5))

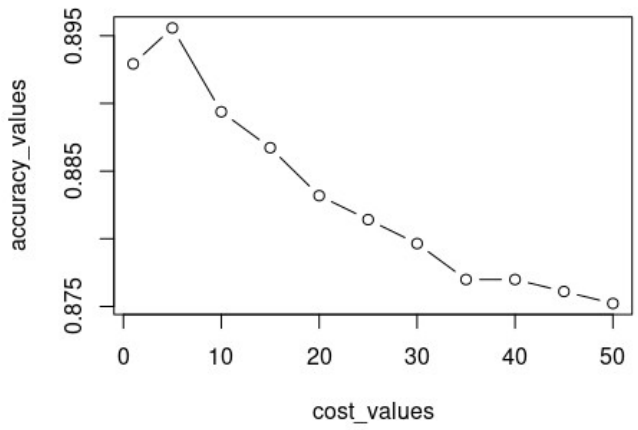
accuracy_values <- sapply(cost_values, function(x) {

  m <-ksvm (y ~ ., data=bank_processed_train, kernel="rbfdot", C = x)
  pred <- predict(m, bank_processed_test)

  agree <- ifelse(pred == bank_processed_test$y, 1, 0)
  accuracy <- sum(agree) / nrow(bank_processed_test)

  return (accuracy)
})

plot(cost_values, accuracy_values, type = "b")
```



El mejor resultado parece que se obtiene para C=5.

Examinamos con más detalle los valores alrededor de 5:

```
set.seed(12345)

cost_values <- c(seq(from = 2, to = 8, by = 1))

accuracy_values <- sapply(cost_values, function(x) {

  m <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot", C = x)
  pred <- predict(m, bank_processed_test)

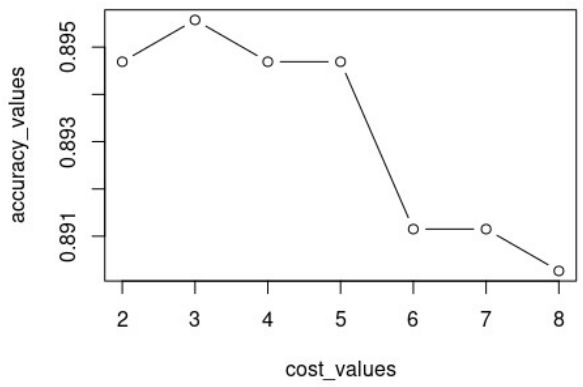
  agree <- ifelse(pred == bank_processed_test$y, 1, 0)
  accuracy <- sum(agree) / nrow(bank_processed_test)

  print(sprintf("C: %f - acc: %f", x, accuracy))

  return (accuracy)
})
## [1] "C: 2.000000 - acc: 0.894690"
## [1] "C: 3.000000 - acc: 0.895575"
## [1] "C: 4.000000 - acc: 0.894690"
## [1] "C: 5.000000 - acc: 0.894690"
## [1] "C: 6.000000 - acc: 0.891150"
## [1] "C: 7.000000 - acc: 0.891150"
## [1] "C: 8.000000 - acc: 0.890265"
```

La gráfica:

```
plot(cost_values, accuracy_values, type = "b")
```



Aunque por muy poco, el mejor valor de la exactitud se da en realidad para C=3. Entonces entrenamos el modelo y hacemos la predicción con ese valor.

```
#train the model
model_rbfdot_C <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot", C = 3)
model_rbfdot_C

## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 3
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0175374530882441
##
## Number of Support Vectors : 959
##
## Objective Function Value : -1758.389
## Training error : 0.070186
```

```
#Confusion matrix
prediction_rbfdot <- predict(model_rbfdot_C, bank_processed_test)
confusionMatrix(as.factor(bank_processed_test$y), as.factor(prediction_rbfdot))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  982  18
##        yes  99  31
##
##                Accuracy : 0.8965
##                  95% CI : (0.8772, 0.9136)
##     No Information Rate : 0.9566
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3024
##
##  Mcnemar's Test P-Value : 1.403e-13
##
##             Sensitivity : 0.9084
##             Specificity : 0.6327
##          Pos Pred Value : 0.9820
##          Neg Pred Value : 0.2385
##              Prevalence : 0.9566
##          Detection Rate : 0.8690
##    Detection Prevalence : 0.8850
##       Balanced Accuracy : 0.7705
##
##        'Positive' Class : no
##
```

Se obtiene una exactitud del 89.65%, solo algunas decimas mejor que con el parámetro C=1 (89.2%).

Al igual que con las redes de neuronas, parece como si hubiera un muro en el 88-89%, y no consigo pasar de este valor.

[

Nota sobre la versión en Python de este ejercicio.

En esta ocasión voy a mostrar aquí la parte del código equivalente:

https://github.com/fcamadi/masterMLpython/blob/main/Tema07/Tema7_SVM.ipynb

```
# We try to find the best C and gamma conbination
param_grid=[
    {'C': [1,10,100,1000],
     'gamma': [0.1, 0.01, 0.001, 0.0001],
     'kernel': ['rbf']},
]
optimal_params = GridSearchCV(
    SVC(),
    param_grid,
    cv=20,
    verbose=1
)
optimal_params.fit(X_train_scaled, y_train)
optimal_params.best_params_
```

En Python he obtenido una exactitud del 89.57%, es decir, solo 4 centésimas menos que en R.

Añadir que durante la redacción de este documento, he buscado el equivalente en R de GridSearchCV. Lo he encontrado en la librería caret, en mlr .. Se puede ver en el código en GitHub. Aquí no lo incluyo porque entiendo que usa el algoritmo SVM de la librería caret (en el código he elegido “svmRadial“), no el de kernlab. El mejor resultado es nuevamente .. 89.65%.

]

3 Comparación de resultados y conclusiones.

Se muestran los mejores resultados para cada modelo y lenguaje de programación.

Redes de neuronas	R RSNNS	R Keras	Python & Keras&TensorFlow
Matriz de confusión	Reference Pred. No Yes No **934** 59 Yes 75 **62**	Reference Pred. No Yes No **953** 40 Yes 99 **38**	Reference Pred. No Yes No **973** 26 Yes 86 **46**
Exactitud	0.8814	0.877	0.901
SVM	R kernlab (rbf)		Python & Scikit-learn SVM (rbf)
Matriz de confusión	Reference Pred. No Yes No **982** 18 Yes 99 **31**		Reference Pred. No Yes No **973** 26 Yes 92 **40**
Exactitud	0.8965		0.8957

Los resultados vuelven a ser muy parecidos en R y en Python.
También son prácticamente idénticos los resultados entre ambos algoritmos.
Tengo la sensación de que SVM es algo más rápido (con el tamaño de este conjunto de datos, y con las configuraciones de redes de neuronas que he usado).

Dada la variedad de casos, variantes, parámetros (especialmente en el caso de redes de neuronas: ¿Cómo elegir el número de capas? ¿El número de neuronas en cada una? ¿Añadir capas de abandono? ¿Qué porcentaje de abandono?), no me ha sido fácil elegir qué modelos comparar. En el caso de redes de neuronas he pensado ni elegir un modelo demasiado sencillo (una capa solo), ni demasiado complicado (he llegado a hacer alguna prueba con 4 capas, pero he usado sobre todo 2 y 3).

La conclusión final a la que llego en todo caso, como ya he mencionado, es que el mejor resultado conseguido está alrededor del 88-89% para la exactitud. Me ha sido muy fácil llegar a esa cifra, pero imposible pasar del 90%.

También quisiera añadir, que entiendo perfectamente que se llame a estas técnicas “de caja negra“. Hay una gran diferencia por ejemplo con los árboles de decisión, que lo muestran todo.

EVALUACIÓN TEMA 8 - BÚSQUEDA DE PATRONES MEDIANTE REGLAS DE ASOCIACIÓN

Utilizando el dataset IncomeESL incluido con la librería *arules*, se pide generar reglas de asociación.

Para ello, previamente deberá depurar el *dataset*. En particular:

- Revisar que no haya valores omitidos.
- Transformar los factores en valores numéricos. ← no es necesario!!!
- Una vez depurado el dataset, crear la matriz de transacciones usando la función *transactions*.

A la hora de ejecutar el algoritmo para obtener las reglas, no olvide establecer los valores de los parámetros de la función *apriori*, justificando el motivo de su elección.

Por último, elabore un breve informe resumiendo las reglas obtenidas y analizando su significado.

1 Paso 1: Carga de los datos

```
if (!require(arules)) install.packages('arules', dependencies = T)
library(arules)

# To plot rules we use package arulesViz
if (!require(arulesViz)) {
  # package graph -and other packages too- must be installed first (and it is not available anymore
in CRAN)
  if (!require("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
    BiocManager::install("Rgraphviz")
    BiocManager::install("graph")
  }
  install.packages('arulesViz', dependencies = T)
}

data("IncomeESL")
#data("Income")
```

2 Paso 2: Explorar y preparar los datos

```
income_raw <- IncomeESL

#structure
str(income_raw)
```

```
## 'data.frame':    8993 obs. of  14 variables:
##  $ income               : Ord.factor w/ 9 levels "[0,10)"<"[10,15)"<..: 9 9 9 1 1 8 1 6 2 4 ...
##  $ sex                  : Factor w/ 2 levels "male","female": 2 1 2 2 2 1 1 1 1 1 ...
```

```
##  $ marital status       : Factor w/ 5 levels "married","cohabitation",..: 1 1 1 5 5 1 5 3 1 1 ...
##  $ age                  : Ord.factor w/ 7 levels "14-17"<"18-24"<..: 5 5 3 1 1 6 2 3 6 7 ...
##  $ education            : Ord.factor w/ 6 levels "grade <9"<"grades 9-11"<..: 4 5 5 2 2 4 3 4 ...
##  $ occupation           : Factor w/ 9 levels "professional/managerial",..: 5 5 1 6 6 8 9 3 8 8 ...
##  $ years in bay area    : Ord.factor w/ 5 levels "<1"<"1-3"<"4-6"<..: 5 5 5 5 3 5 4 5 5 4 ...
##  $ dual incomes         : Factor w/ 3 levels "not married",..: 3 3 2 1 1 3 1 1 3 3 ...
##  $ number in household  : Ord.factor w/ 9 levels "1"<"2"<"3"<"4"<..: 3 5 3 4 4 2 3 1 3 2 ...
##  $ number of children   : Ord.factor w/ 10 levels "0"<"1"<"2"<"3"<..: 1 3 2 3 3 1 2 1 1 1 ...
##  $ householder status   : Factor w/ 3 levels "own","rent","live with parents/family": 1 1 2 3 3...
##  $ type of home         : Factor w/ 5 levels "house","condominium",..: 1 1 3 1 1 1 3 3 3 3 ...
##  $ ethnic classification: Factor w/ 8 levels "american indian",..: 7 7 7 7 7 7 7 7 7 7 ...
##  $ language in home     : Factor w/ 3 levels "english","spanish",..: NA 1 1 1 1 1 1 1 1 1 ...
```

Nos quedamos con las observaciones que estén completas:

```
# only complete observations
income_complete <- income_raw[complete.cases(income_raw), ]
```

Y obtenemos las transacciones con ellas:

```
# get transactions
income_transac <- transactions(income_complete)
```

Estructura y sumario de las transacciones:

```
#Structure
str(income_transac)
## Formal class 'transactions' [package "arules"] with 3 slots
##   ..@ data       :Formal class 'ngCMatrix' [package "Matrix"] with 5 slots
##   .. .. ..@ i       : int [1:96264] 8 9 11 20 27 33 42 45 50 57 ...
##   .. .. ..@ p       : int [1:6877] 0 14 28 42 56 70 84 98 112 126 ...
##   .. .. ..@ Dim     : int [1:2] 84 6876
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ factors : list()
##   ..@ itemInfo   :'data.frame':  84 obs. of  3 variables:
##   .. ..$ labels   : chr [1:84] "income=[0,10)" "income=[10,15)" "income=[15,20)" "income=[20,25)"
...
##   .. ..$ variables: Factor w/ 14 levels "age","dual incomes",..: 6 6 6 6 6 6 6 6 6 12 ...
##   .. ..$ levels   : Factor w/ 73 levels "[0,10)","[10,15)",..: 1 2 3 4 5 6 7 8 29 54 ...
##   ..@ itemsetInfo:'data.frame':  6876 obs. of  1 variable:
##   .. ..$ transactionID: chr [1:6876] "2" "3" "4" "5" ...
```

```
#Summary
summary(income_transac)
## transactions as itemMatrix in sparse format with
##  6876 rows (elements/itemsets/transactions) and
##  84 columns (items) and a density of 0.1666667
##
## most frequent items:
##    language in home=english ethnic classification=white
##                        6277                        4605
##       years in bay area=>10        number of children=0
##                        4446                        4276
##    dual incomes=not married                     (Other)
##                        4114                       72546
##
## element (itemset/transaction) length distribution:
## sizes
##   14
## 6876
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      14      14      14      14      14      14
##
## includes extended item information - examples:
```

```
##           labels variables  levels
## 1  income=[0,10)    income  [0,10)
## 2 income=[10,15)    income [10,15)
## 3 income=[15,20)    income [15,20)
##
## includes extended transaction information - examples:
##   transactionID
## 1             2
## 2             3
## 3             4
```

La matriz dispersa tiene 6876 filas y 84 columnas. Al no tratarse de un problema como el de la cesta de la compra, y al tener las filas originales todas 14 valores por haber eliminado las filas que tenían valores nulos, todas las transacciones tienen 14 valores. La densidad es el 16.67%.
(No soy sociólogo, pero los items más frecuentes, dan una idea bastante clara de la población de la muestra).

Examinamos un par de transacciones con “inspect”:

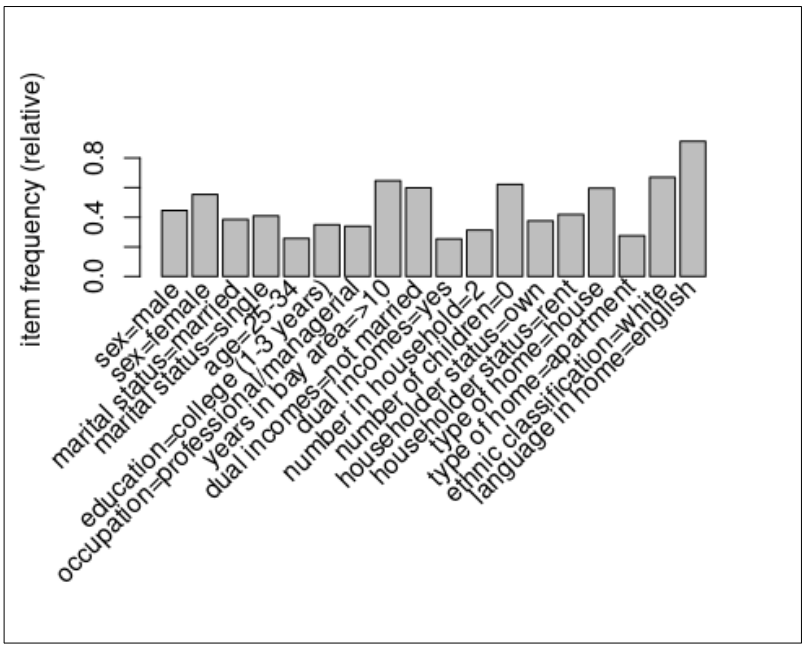
```
inspect(income_transac[1:2])
```

```
##     items                                  transactionID
## [1] {income=75+,
##      sex=male,
##      marital status=married,
##      age=45-54,
##      education=college graduate,
##      occupation=homemaker,
##      years in bay area=>10,
##      dual incomes=no,
##      number in household=5,
##      number of children=2,
##      householder status=own,
##      type of home=house,
##      ethnic classification=white,
##      language in home=english}                       2
## [2] {income=75+,
##      sex=female,
##      marital status=married,
##      age=25-34,
##      education=college graduate,
##      occupation=professional/managerial,
##      years in bay area=>10,
##      dual incomes=yes,
##      number in household=3,
##      number of children=1,
##      householder status=rent,
##      type of home=apartment,
##      ethnic classification=white,
##      language in home=english}                       3
```

1 Visualización de los datos

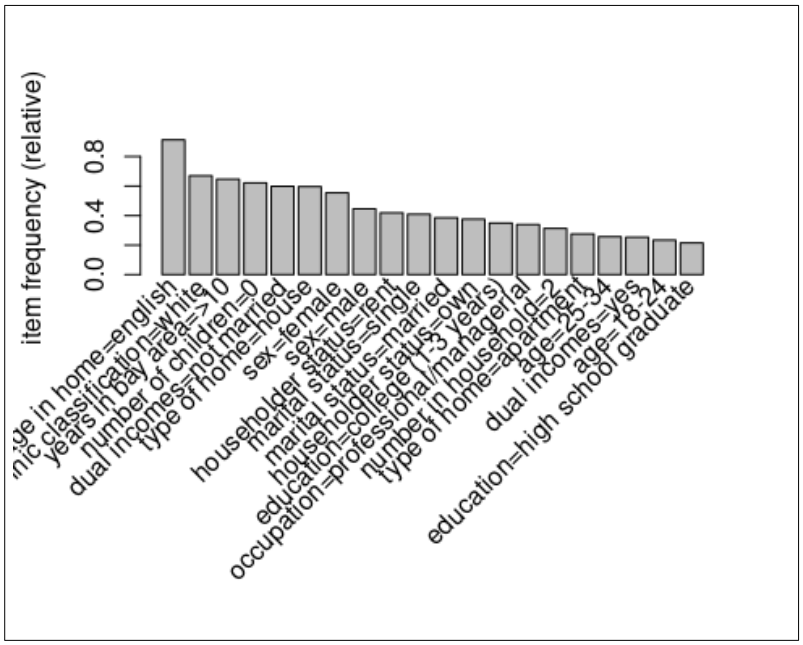
- Items con al menos un 25% de frecuencia

```
itemFrequencyPlot(income_transac, support = 0.25)   # at least 25 percent support
```



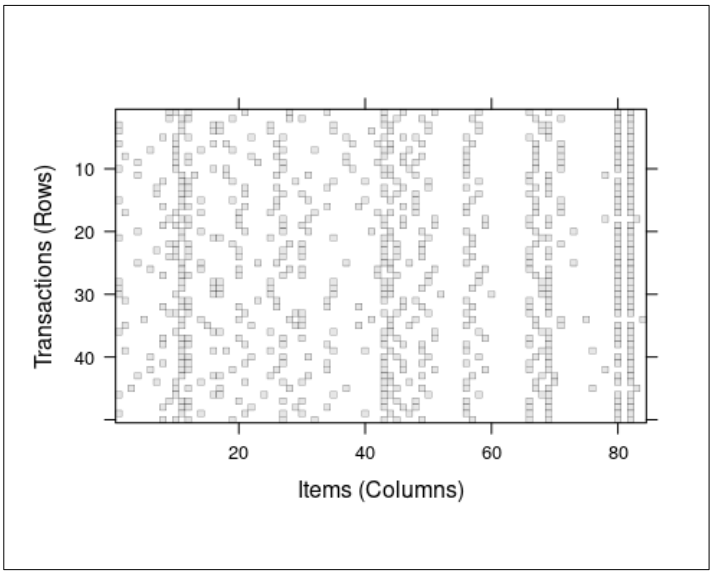
- Los 20 items más frecuentes

```
itemFrequencyPlot(income_transac, topN = 20) # top 20 items
```

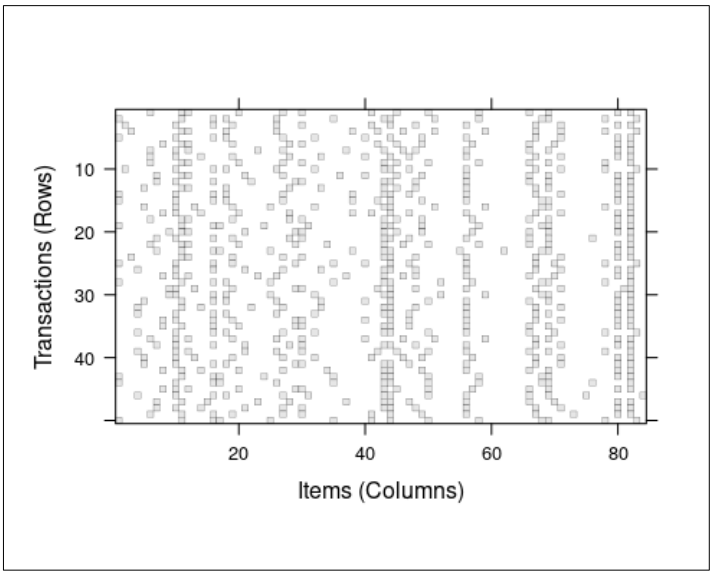


- Visualizamos también las matrices dispersas para algunos tramos de datos (por si vemos algo extraño).

```
image(income_transac[1:50])
#image(income_transac[1000:1050])
#image(income_transac[2000:2050])
#image(income_transac[5000:5050])
```




```
image(sample(income_transac, 50))
```



Se ve cómo predomina el idioma inglés (últimas columnas). No veo nada más que sea digno de mención.

3 Paso 3: Entrenamiento del modelo

Vamos a probar primero con los valores por defecto para soporte y confianza:

```
system.time({
  income_rules <- apriori(income_transac)
})

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 687
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[84 item(s), 6876 transaction(s)] done [0.01s].
## sorting and recoding items ... [42 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.02s].
## writing ... [1111 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

##    user  system elapsed
##   0.055   0.000   0.055
```

Con los parámetros por defecto (soporte 10%, confianza 80%), se obtienen 1111 reglas.

Vamos a aumentar el soporte y a disminuir la confianza:

```
system.time({
income_rules <- apriori(income_transac, parameter = list(support = 0.5, confidence = 0.5,
minlen = 2))
})

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5     0.5      2
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 3438
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[84 item(s), 6876 transaction(s)] done [0.01s].
## sorting and recoding items ... [7 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [12 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

##    user  system elapsed
##   0.017   0.000   0.017
```

Ahora obtenemos solo 12 reglas:

```
inspect(income_rules)
##      lhs                                rhs                                support confidence
coverage     lift count
## [1]  {sex=female}                    => {language in home=english}    0.5122164  0.9246521
0.5539558 1.012890  3522
## [2]  {language in home=english}     => {sex=female}                   0.5122164  0.5610961
0.9128854 1.012890  3522
## [3]  {type of home=house}           => {language in home=english}    0.5446481  0.9129693
0.5965678 1.000092  3745
## [4]  {language in home=english}     => {type of home=house}           0.5446481  0.5966226
0.9128854 1.000092  3745
## [5]  {dual incomes=not married}     => {language in home=english}    0.5426120  0.9069033
0.5983130 0.993447  3731
## [6]  {language in home=english}     => {dual incomes=not married}     0.5426120  0.5943922
0.9128854 0.993447  3731
## [7]  {number of children=0}         => {language in home=english}    0.5801338  0.9328812
0.6218732 1.021904  3989
## [8]  {language in home=english}     => {number of children=0}         0.5801338  0.6354947
0.9128854 1.021904  3989
## [9]  {years in bay area=>10}        => {language in home=english}    0.6013671  0.9300495
0.6465969 1.018802  4135
## [10] {language in home=english}     => {years in bay area=>10}        0.6013671  0.6587542
0.9128854 1.018802  4135
## [11] {ethnic classification=white} => {language in home=english}    0.6595404  0.9847991
0.6697208 1.078776  4535
## [12] {language in home=english}     => {ethnic classification=white} 0.6595404  0.7224789
0.9128854 1.078776  4535
```

Considero todas estas reglas triviales. No aportan información relevante. Si se trata de un dataset de una población de los EEUU … lógico que la gran mayoría hable inglés (ya sean blancos, o lleven más de 10 años en esa zona).

Vamos a buscar algo intermedio:

```
system.time({
  income_rules <- apriori(income_transac, parameter = list(support = 0.25,
confidence = 0.5, minlen = 2))
})
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.25      2
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 1719
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[84 item(s), 6876 transaction(s)] done [0.01s].
## sorting and recoding items ... [18 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [186 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

##    user  system elapsed
##   0.016   0.000   0.016
```

```
inspect(sort(income_rules, by = "lift")[1:20])
```

```
##      lhs                             rhs                          support confidence  coverage     lift count
## [1]  {type of home=apartment}     => {householder status=rent}  0.2507272  0.9097625 0.2755963 2.170551  1724
## [2]  {householder status=rent}    => {type of home=apartment}   0.2507272  0.5981957 0.4191390 2.170551  1724
## [3]  {householder status=own}     => {marital status=married}   0.2614892  0.6958204 0.3757999 1.804096  1798
## [4]  {marital status=married}     => {householder status=own}   0.2614892  0.6779789 0.3856894 1.804096  1798
## [5]  {marital status=single}      => {dual incomes=not married} 0.4091041  1.0000000 0.4091041 1.671366  2813
## [6]  {dual incomes=not married}   => {marital status=single}    0.4091041  0.6837628 0.5983130 1.671366  2813
## [7]  {marital status=single,
##       ethnic classification=white} => {dual incomes=not married} 0.2524724  1.0000000 0.2524724 1.671366  1736
## [8]  {marital status=single,
##       language in home=english}    => {dual incomes=not married} 0.3647469  1.0000000 0.3647469 1.671366  2508
## [9]  {dual incomes=not married,
##       language in home=english}    => {marital status=single}    0.3647469  0.6722058 0.5426120 1.643117  2508
## [10] {dual incomes=not married,
##       ethnic classification=white} => {marital status=single}    0.2524724  0.6608298 0.3820535 1.615310  1736
## [11] {number in household=2,
##       language in home=english}    => {number of children=0}     0.2725422  0.9186275 0.2966841 1.477194  1874
## [12] {number in household=2}      => {number of children=0}     0.2878127  0.9179035 0.3135544 1.476030  1979
## [13] {type of home=house,
##       language in home=english}    => {householder status=own}   0.2987202  0.5484646 0.5446481 1.459459  2054
## [14] {householder status=own,
##       language in home=english}    => {type of home=house}       0.2987202  0.8421484 0.3547120 1.411656  2054
## [15] {householder status=own}     => {type of home=house}       0.3161722  0.8413313 0.3757999 1.410286  2174
## [16] {type of home=house}         => {householder status=own}   0.3161722  0.5299854 0.5965678 1.410286  2174
## [17] {marital status=married,
##       language in home=english}    => {type of home=house}       0.2617801  0.7371007 0.3551483 1.235569  1800
## [18] {householder status=rent,
##       language in home=english}    => {number of children=0}     0.2920303  0.7608943 0.3837987 1.223552  2008
## [19] {marital status=married}     => {type of home=house}       0.2814136  0.7296380 0.3856894 1.223060  1935
## [20] {number of children=0,
##       language in home=english}    => {householder status=rent}  0.2920303  0.5033843 0.5801338 1.200996  2008
```

Creo que de estas reglas sí se podrían sacar conclusiones. Pero de todas maneras, considero que son demasiadas (186), y no veo los ingresos por ningún lado (aparecen muy pocas veces en el total de reglas).

Voy a examinar con más detalle los ingresos. Voy a dividirlos en 3 niveles (bajo, medio, alto), y voy a intentar obtener reglas en las que estos 3 niveles aparezcan en el consecuente.

```
# only complete observations
table(income_complete$income)
```

```
##
##  [0,10) [10,15) [15,20) [20,25) [25,30) [30,40) [40,50) [50,75)     75+
##    1255     529     505     618     527     846     784    1069     743
```

```
levels(income_complete[["income"]])
## [1] "[0,10)"  "[10,15)" "[15,20)" "[20,25)" "[25,30)" "[30,40)" "[40,50)"
## [8] "[50,75)" "75+"

income_complete["income3levels"] <- income_complete["income"]
levels(income_complete[["income3levels"]]) <- c("0-20k$","0-20k$","0-20k$","20k-50k$","20k-
50k$","20k-50k$","20k-50k$","50k+","50k+")

#tests
income_complete[0:5,c("income","income3levels")]
##    income income3levels
## 2     75+          50k+
## 3     75+          50k+
## 4  [0,10)        0-20k$
## 5  [0,10)        0-20k$
## 6 [50,75)          50k+

# remove old income variable with 9 levels
income3L_complete <- income_complete[,-1]
```

Obtenemos las transacciones con este nuevo dataframe.

```
# get transactions
income3L_transac <- transactions(income3L_complete)
```

Ahora aplicamos el algoritmo Apriori a estas transacciones:

```
system.time({
      income3L_rules <- apriori(income3L_transac, parameter = list(supp = 0.25,
conf = 0.5, minlen = 2))
})
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.25      2
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 1719
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[78 item(s), 6876 transaction(s)] done [0.00s].
## sorting and recoding items ... [21 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [199 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

##    user  system elapsed
##   0.018   0.000   0.018
```

Con el mismo nivel de soporte y confianza, obtenemos ahora 199 reglas, en lugar de 186. Presupongo que son las mismas o muy parecidas:

```
inspect(sort(income3L_rules, by = "lift")[1:20])
```

```
##      lhs                                rhs                          support confidence  coverage     lift count
## [1]  {type of home=apartment}      => {householder status=rent}  0.2507272  0.9097625 0.2755963 2.170551  1724
## [2]  {householder status=rent}     => {type of home=apartment}   0.2507272  0.5981957 0.4191390 2.170551  1724
## [3]  {householder status=own}      => {marital status=married}   0.2614892  0.6958204 0.3757999 1.804096  1798
## [4]  {marital status=married}      => {householder status=own}   0.2614892  0.6779789 0.3856894 1.804096  1798
## [5]  {marital status=single}       => {dual incomes=not married} 0.4091041  1.0000000 0.4091041 1.671366  2813
## [6]  {dual incomes=not married}    => {marital status=single}    0.4091041  0.6837628 0.5983130 1.671366  2813
## [7]  {marital status=single,
##       ethnic classification=white} => {dual incomes=not married} 0.2524724  1.0000000 0.2524724 1.671366  1736
## [8]  {marital status=single,
##       language in home=english}    => {dual incomes=not married} 0.3647469  1.0000000 0.3647469 1.671366  2508
## [9]  {dual incomes=not married,
##       language in home=english}    => {marital status=single}    0.3647469  0.6722058 0.5426120 1.643117  2508
...
```

Lo que se pretende es encontrar las reglas que en el lado derecho tengan alguno de los 3 niveles de ingresos.

Probamos a especificar los niveles de ingreso en el lado derecho, y ajustamos los valores de soporte y confianza:

```
system.time({
                                                    # supp = 0.33, conf = 0.5 -> 0 rules
                                                    # supp = 0.2, conf = 0.8 -> 0
                                                    # supp = 0.1, conf = 0.5 -> 101
                                                    # supp = 0.1, conf = 0.6 -> 29
    income3L_rules_RHS <-
        apriori(income3L_transac, parameter = list(supp = 0.1, conf = 0.55, minlen = 2),
                appearance = list(rhs=c("income3levels=0-20k$", "income3levels=20k-50k$",
                                                                "income3levels=50k+")))
})

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.55    0.1    1 none FALSE            TRUE       5     0.1      2
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 687
##
## set item appearances ...[3 item(s)] done [0.00s].
## set transactions ...[78 item(s), 6876 transaction(s)] done [0.00s].
## sorting and recoding items ... [40 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.02s].
## writing ... [57 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

##    user  system elapsed
##   0.039   0.000   0.039
```

Vamos a examinar las 57 reglas obtenidas ahora:
(Estas son las reglas de asociación que estaba buscando).

```
inspect(sort(income3L_rules_RHS, by = "lift"))
##      lhs                                                rhs                        support confidence  coverage     lift count
## [1]  {dual incomes=yes,
##       householder status=own}                        => {income3levels=50k+}     0.1023851  0.6622766 0.1545957 2.513142   704
## [2]  {marital status=married,
##       occupation=professional/managerial,
##       language in home=english}                      => {income3levels=50k+}     0.1012216  0.6590909 0.1535777 2.501054   696
## [3]  {occupation=professional/managerial,
##       householder status=own,
##       language in home=english}                      => {income3levels=50k+}     0.1090750  0.6590510 0.1655032 2.500902   750
## [4]  {occupation=professional/managerial,
##       householder status=own}                        => {income3levels=50k+}     0.1122746  0.6547922 0.1714660 2.484741   772
## [5]  {marital status=married,
##       occupation=professional/managerial}            => {income3levels=50k+}     0.1048575  0.6501353 0.1612856 2.467070   721
## [6]  {occupation=student,
##       dual incomes=not married,
##       language in home=english}                      => {income3levels=0-20k$}   0.1025305  0.7730263 0.1326353 2.322118   705
## [7]  {marital status=single,
##       occupation=student}                            => {income3levels=0-20k$}   0.1118383  0.7713139 0.1449971 2.316975   769
## [8]  {marital status=single,
##       occupation=student,
##       dual incomes=not married}                      => {income3levels=0-20k$}   0.1118383  0.7713139 0.1449971 2.316975   769
## [9]  {occupation=student,
##       dual incomes=not married}                      => {income3levels=0-20k$}   0.1182373  0.7698864 0.1535777 2.312686   813
## [10] {marital status=married,
##       householder status=own,
##       type of home=house,
##       ethnic classification=white,
##       language in home=english}                      => {income3levels=50k+}     0.1103839  0.6081731 0.1815009 2.307836   759
## [11] {marital status=married,
##       householder status=own,
##       type of home=house,
##       ethnic classification=white}                   => {income3levels=50k+}     0.1111111  0.6068308 0.1831006 2.302742   764
## [12] {marital status=married,
##       householder status=own,
##       ethnic classification=white,
##       language in home=english}                      => {income3levels=50k+}     0.1215823  0.5992832 0.2028796 2.274101   836
## [13] {marital status=married,
```

```
##       householder status=own,
##       ethnic classification=white}                  => {income3levels=50k+}     0.1224549  0.5975869 0.2049156 2.267664   842
## [14] {occupation=student,
##       language in home=english}                     => {income3levels=0-20k$}   0.1061664  0.7510288 0.1413613 2.256039   730
## [15] {occupation=student}                           => {income3levels=0-20k$}   0.1231821  0.7508865 0.1640489 2.255612   847
## [16] {marital status=married,
##       householder status=own,
##       type of home=house,
##       language in home=english}                     => {income3levels=50k+}     0.1307446  0.5922266 0.2207679 2.247324   899
## [17] {marital status=married,
##       householder status=own,
##       language in home=english}                     => {income3levels=50k+}     0.1445608  0.5847059 0.2472368 2.218785   994
## [18] {dual incomes=yes,
##       type of home=house}                           => {income3levels=50k+}     0.1009308  0.5827036 0.1732112 2.211187   694
## [19] {marital status=married,
##       householder status=own,
##       type of home=house}                           => {income3levels=50k+}     0.1351076  0.5799001 0.2329843 2.200548   929
## [20] {marital status=married,
##       householder status=own}                       => {income3levels=50k+}     0.1496510  0.5723026 0.2614892 2.171718  1029
## [21] {occupation=professional/managerial,
##       type of home=house,
##       language in home=english}                     => {income3levels=50k+}     0.1090750  0.5660377 0.1926992 2.147945   750
## [22] {marital status=married,
##       type of home=house,
##       ethnic classification=white,
##       language in home=english}                     => {income3levels=50k+}     0.1185282  0.5659722 0.2094241 2.147696   815
## [23] {marital status=married,
##       type of home=house,
##       ethnic classification=white}                  => {income3levels=50k+}     0.1194008  0.5654270 0.2111693 2.145627   821
## [24] {marital status=married,
##       years in bay area=>10,
##       householder status=own,
##       type of home=house}                           => {income3levels=50k+}     0.1022397  0.5597134 0.1826643 2.123945   703
## [25] {marital status=married,
##       years in bay area=>10,
##       householder status=own,
##       language in home=english}                     => {income3levels=50k+}     0.1070390  0.5579985 0.1918266 2.117438   736
## [26] {occupation=professional/managerial,
##       type of home=house}                           => {income3levels=50k+}     0.1128563  0.5578720 0.2022978 2.116958   776
## [27] {marital status=single,
##       householder status=live with parents/family}  => {income3levels=0-20k$}   0.1316172  0.6983025 0.1884817 2.097653   905
## [28] {marital status=single,
##       dual incomes=not married,
##       householder status=live with parents/family}  => {income3levels=0-20k$}   0.1316172  0.6983025 0.1884817 2.097653   905
## [29] {dual incomes=not married,
##       householder status=live with parents/family}  => {income3levels=0-20k$}   0.1394706  0.6979622 0.1998255 2.096631   959
## [30] {marital status=single,
##       householder status=live with parents/family,
##       language in home=english}                     => {income3levels=0-20k$}   0.1122746  0.6961226 0.1612856 2.091105   772
## [31] {marital status=single,
##       dual incomes=not married,
##       householder status=live with parents/family,
##       language in home=english}                     => {income3levels=0-20k$}   0.1122746  0.6961226 0.1612856 2.091105   772
## [32] {householder status=live with parents/family}  => {income3levels=0-20k$}   0.1423793  0.6943262 0.2050611 2.085709   979
## [33] {dual incomes=not married,
##       householder status=live with parents/family,
##       language in home=english}                     => {income3levels=0-20k$}   0.1185282  0.6942078 0.1707388 2.085353   815
## [34] {householder status=live with parents/family,
##       language in home=english}                     => {income3levels=0-20k$}   0.1201280  0.6889074 0.1743746 2.069431   826
## [35] {marital status=single,
##       householder status=live with parents/family,
##       type of home=house}                           => {income3levels=0-20k$}   0.1060209  0.6800373 0.1559046 2.042786   729
## [36] {marital status=single,
##       dual incomes=not married,
##       householder status=live with parents/family,
##       type of home=house}                           => {income3levels=0-20k$}   0.1060209  0.6800373 0.1559046 2.042786   729
## [37] {dual incomes=not married,
##       householder status=live with parents/family,
##       type of home=house}                           => {income3levels=0-20k$}   0.1119837  0.6790123 0.1649215 2.039707   770
## [38] {householder status=live with parents/family,
##       type of home=house}                           => {income3levels=0-20k$}   0.1140198  0.6746988 0.1689936 2.026749   784
## [39] {sex=female,
##       marital status=single}                        => {income3levels=0-20k$}   0.1237638  0.6184593 0.2001163 1.857810   851
## [40] {sex=female,
##       marital status=single,
##       dual incomes=not married}                     => {income3levels=0-20k$}   0.1237638  0.6184593 0.2001163 1.857810   851
## [41] {sex=female,
##       marital status=single,
##       language in home=english}                     => {income3levels=0-20k$}   0.1099476  0.6106624 0.1800465 1.834388   756
## [42] {sex=female,
##       marital status=single,
##       dual incomes=not married,
##       language in home=english}                     => {income3levels=0-20k$}   0.1099476  0.6106624 0.1800465 1.834388   756
## [43] {marital status=single,
##       type of home=house}                           => {income3levels=0-20k$}   0.1327807  0.5830140 0.2277487 1.751334   913
## [44] {marital status=single,
##       dual incomes=not married,
##       type of home=house}                           => {income3levels=0-20k$}   0.1327807  0.5830140 0.2277487 1.751334   913
## [45] {marital status=single,
##       type of home=house,
##       language in home=english}                     => {income3levels=0-20k$}   0.1163467  0.5813953 0.2001163 1.746472   800
## [46] {marital status=single,
##       dual incomes=not married,
##       type of home=house,
##       language in home=english}                     => {income3levels=0-20k$}   0.1163467  0.5813953 0.2001163 1.746472   800
## [47] {age=18-24,
##       dual incomes=not married,
##       language in home=english}                     => {income3levels=0-20k$}   0.1022397  0.5738776 0.1781559 1.723889   703
## [48] {age=18-24,
```

```
##       dual incomes=not married}                    => {income3levels=0-20k$}   0.1156195  0.5731795 0.2017161 1.721792   795
## [49] {marital status=single,
##       age=18-24}                                   => {income3levels=0-20k$}   0.1015125  0.5670187 0.1790285 1.703285   698
## [50] {marital status=single,
##       age=18-24,
##       dual incomes=not married}                    => {income3levels=0-20k$}   0.1015125  0.5670187 0.1790285 1.703285   698
## [51] {marital status=single}                       => {income3levels=0-20k$}   0.2268761  0.5545681 0.4091041 1.665885  1560
## [52] {marital status=single,
##       dual incomes=not married}                    => {income3levels=0-20k$}   0.2268761  0.5545681 0.4091041 1.665885  1560
## [53] {age=18-24}                                   => {income3levels=0-20k$}   0.1295812  0.5534161 0.2341478 1.662424   891
## [54] {age=18-24,
##       language in home=english}                    => {income3levels=0-20k$}   0.1135835  0.5503876 0.2063700 1.653327   781
## [55] {occupation=professional/managerial,
##       dual incomes=not married}                    => {income3levels=20k-50k$} 0.1016579  0.5964164 0.1704479 1.477823   699
## [56] {age=25-34,
##       householder status=rent}                     => {income3levels=20k-50k$} 0.1009308  0.5866441 0.1720477 1.453609   694
## [57] {sex=male,
##       householder status=rent,
##       language in home=english}                    => {income3levels=20k-50k$} 0.1018034  0.5524862 0.1842641 1.368971   700
```

Vamos a intentar encontrar ahora las reglas redundantes (con la función **is.subset()**), eliminarlas y simplicar así el conjunto de reglas:

```
income3L_rules_RHS_sortedByLift <- sort(income3L_rules_RHS, by = "lift")

#inspect(income3L_rules_RHS_sortedByLift)
subset_matrix <- is.subset(income3L_rules_RHS_sortedByLift,income3L_rules_RHS_sortedByLift)
#subset_matrix
```

Eliminamos las ocurrencias que se producen en la diagonal principal y por debajo de ella:

```
subset_matrix[lower.tri(subset_matrix, diag = T)] <- F
#subset_matrix
```

Ahora podemos extraer las reglas de asociación redundantes, que serán aquellas que tengan una columna cuya suma sea igual o superior a 1:

```
redundant <- colSums(subset_matrix, na.rm = T) >= 1
redundant
```

[

output:

```
{dual incomes=yes,householder status=own,income3levels=50k+}
FALSE
{marital status=married,occupation=professional/managerial,language in
home=english,income3levels=50k+}
FALSE
{occupation=professional/managerial,householder status=own,language in
home=english,income3levels=50k+}
FALSE
...
{marital status=single,occupation=student,dual incomes=not married,income3levels=0-20k$}
TRUE
{occupation=student,dual incomes=not married,income3levels=0-20k$}
FALSE
...
```

]

Las reglas marcadas como TRUE son redundantes por lo que podemos eliminarlas:

```
rules_pruned <- income3L_rules_RHS_sortedByLift[!redundant]
inspect(rules_pruned)
##      lhs                                                rhs                        support confidence  coverage     lift count
## [1]  {dual incomes=yes,
##       householder status=own}                        => {income3levels=50k+}     0.1023851  0.6622766 0.1545957 2.513142   704
## [2]  {marital status=married,
##       occupation=professional/managerial,
##       language in home=english}                      => {income3levels=50k+}     0.1012216  0.6590909 0.1535777 2.501054   696
## [3]  {occupation=professional/managerial,
##       householder status=own,
##       language in home=english}                      => {income3levels=50k+}     0.1090750  0.6590510 0.1655032 2.500902   750
## [4]  {occupation=professional/managerial,
##       householder status=own}                        => {income3levels=50k+}     0.1122746  0.6547922 0.1714660 2.484741   772
## [5]  {marital status=married,
##       occupation=professional/managerial}            => {income3levels=50k+}     0.1048575  0.6501353 0.1612856 2.467070   721
## [6]  {occupation=student,
##       dual incomes=not married,
##       language in home=english}                      => {income3levels=0-20k$}   0.1025305  0.7730263 0.1326353 2.322118   705
## [7]  {marital status=single,
##       occupation=student}                            => {income3levels=0-20k$}   0.1118383  0.7713139 0.1449971 2.316975   769
## [8]  {occupation=student,
##       dual incomes=not married}                      => {income3levels=0-20k$}   0.1182373  0.7698864 0.1535777 2.312686   813
## [9]  {marital status=married,
##       householder status=own,
##       type of home=house,
##       ethnic classification=white,
##       language in home=english}                      => {income3levels=50k+}     0.1103839  0.6081731 0.1815009 2.307836   759
## [10] {marital status=married,
##       householder status=own,
##       type of home=house,
##       ethnic classification=white}                   => {income3levels=50k+}     0.1111111  0.6068308 0.1831006 2.302742   764
## [11] {marital status=married,
##       householder status=own,
##       ethnic classification=white,
##       language in home=english}                      => {income3levels=50k+}     0.1215823  0.5992832 0.2028796 2.274101   836
## [12] {marital status=married,
##       householder status=own,
##       ethnic classification=white}                   => {income3levels=50k+}     0.1224549  0.5975869 0.2049156 2.267664   842
## [13] {occupation=student,
##       language in home=english}                      => {income3levels=0-20k$}   0.1061664  0.7510288 0.1413613 2.256039   730
## [14] {occupation=student}                            => {income3levels=0-20k$}   0.1231821  0.7508865 0.1640489 2.255612   847
## [15] {marital status=married,
##       householder status=own,
##       type of home=house,
##       language in home=english}                      => {income3levels=50k+}     0.1307446  0.5922266 0.2207679 2.247324   899
## [16] {marital status=married,
##       householder status=own,
##       language in home=english}                      => {income3levels=50k+}     0.1445608  0.5847059 0.2472368 2.218785   994
## [17] {dual incomes=yes,
##       type of home=house}                            => {income3levels=50k+}     0.1009308  0.5827036 0.1732112 2.211187   694
## [18] {marital status=married,
##       householder status=own,
##       type of home=house}                            => {income3levels=50k+}     0.1351076  0.5799001 0.2329843 2.200548   929
## [19] {marital status=married,
##       householder status=own}                        => {income3levels=50k+}     0.1496510  0.5723026 0.2614892 2.171718  1029
## [20] {occupation=professional/managerial,
##       type of home=house,
##       language in home=english}                      => {income3levels=50k+}     0.1090750  0.5660377 0.1926992 2.147945   750
## [21] {marital status=married,
##       type of home=house,
##       ethnic classification=white,
##       language in home=english}                      => {income3levels=50k+}     0.1185282  0.5659722 0.2094241 2.147696   815
## [22] {marital status=married,
##       type of home=house,
##       ethnic classification=white}                   => {income3levels=50k+}     0.1194008  0.5654270 0.2111693 2.145627   821
## [23] {occupation=professional/managerial,
##       type of home=house}                            => {income3levels=50k+}     0.1128563  0.5578720 0.2022978 2.116958   776
## [24] {marital status=single,
##       householder status=live with parents/family}   => {income3levels=0-20k$}   0.1316172  0.6983025 0.1884817 2.097653   905
## [25] {dual incomes=not married,
##       householder status=live with parents/family}   => {income3levels=0-20k$}   0.1394706  0.6979622 0.1998255 2.096631   959
## [26] {householder status=live with parents/family}   => {income3levels=0-20k$}   0.1423793  0.6943262 0.2050611 2.085709   979
## [27] {sex=female,
##       marital status=single}                         => {income3levels=0-20k$}   0.1237638  0.6184593 0.2001163 1.857810   851
## [28] {marital status=single,
##       type of home=house}                            => {income3levels=0-20k$}   0.1327807  0.5830140 0.2277487 1.751334   913
## [29] {age=18-24,
##       dual incomes=not married,
##       language in home=english}                      => {income3levels=0-20k$}   0.1022397  0.5738776 0.1781559 1.723889   703
## [30] {age=18-24,
##       dual incomes=not married}                      => {income3levels=0-20k$}   0.1156195  0.5731795 0.2017161 1.721792   795
## [31] {marital status=single,
##       age=18-24}                                     => {income3levels=0-20k$}   0.1015125  0.5670187 0.1790285 1.703285   698
## [32] {marital status=single}                         => {income3levels=0-20k$}   0.2268761  0.5545681 0.4091041 1.665885  1560
## [33] {age=18-24}                                     => {income3levels=0-20k$}   0.1295812  0.5534161 0.2341478 1.662424   891
## [34] {occupation=professional/managerial,
##       dual incomes=not married}                      => {income3levels=20k-50k$} 0.1016579  0.5964164 0.1704479 1.477823   699
## [35] {age=25-34,
##       householder status=rent}                       => {income3levels=20k-50k$} 0.1009308  0.5866441 0.1720477 1.453609   694
## [36] {sex=male,
##       householder status=rent,
##       language in home=english}                      => {income3levels=20k-50k$} 0.1018034  0.5524862 0.1842641 1.368971   700
```

Nos quedamos finalmente con 36 reglas relativas a los niveles de ingresos.

Las reglas con mayor elevación son las que incluyen en los consecuentes los sueldos altos. Las que menos, las que incluyen los sueldos intermedios.

Las reglas de asociación de este grupo de población con sueldos intermedios son las 3 últimas (34,35 y 36). La 35 por ejemplo indica que teniendo una edad entre 25 y 34, si se vive de alquiler, es porque probablemente tendrá un sueldo que le permite vivir de manera independiente (no en el domicilio de los padres), pero tampoco adquirir una vivienda en propiedad.

La regla 31 indica que las personas solteras, y con una edad entre 18 y 24, tendrán probablemente un sueldo inferior a 20k $. (Igual que la regla 29 y 30 ..) Si la ocupación es estudiante, entonces la regla de asociación con un sueldo bajo tiene una elevación y una confianza todavía mayores (regla 14). (Y mayores todavía si la regla incluye en la parte izquierda que se está soltero y se es estudiante, regla 7).

En cuanto a los sueldos más altos: Si hay dos sueldos en la unidad familiar, y se tiene la propiedad de la vivienda (regla 1), eso indica que es bastante probable que se trate de sueldos altos. (También si la categoría profesional es alta, y se tiene en propiedad la casa -y más si se habla inglés-, reglas 2 y 3).

Métricas:

Se pueden generar diferentes métricas con la función interestMeasure() disponible en la librería arules.

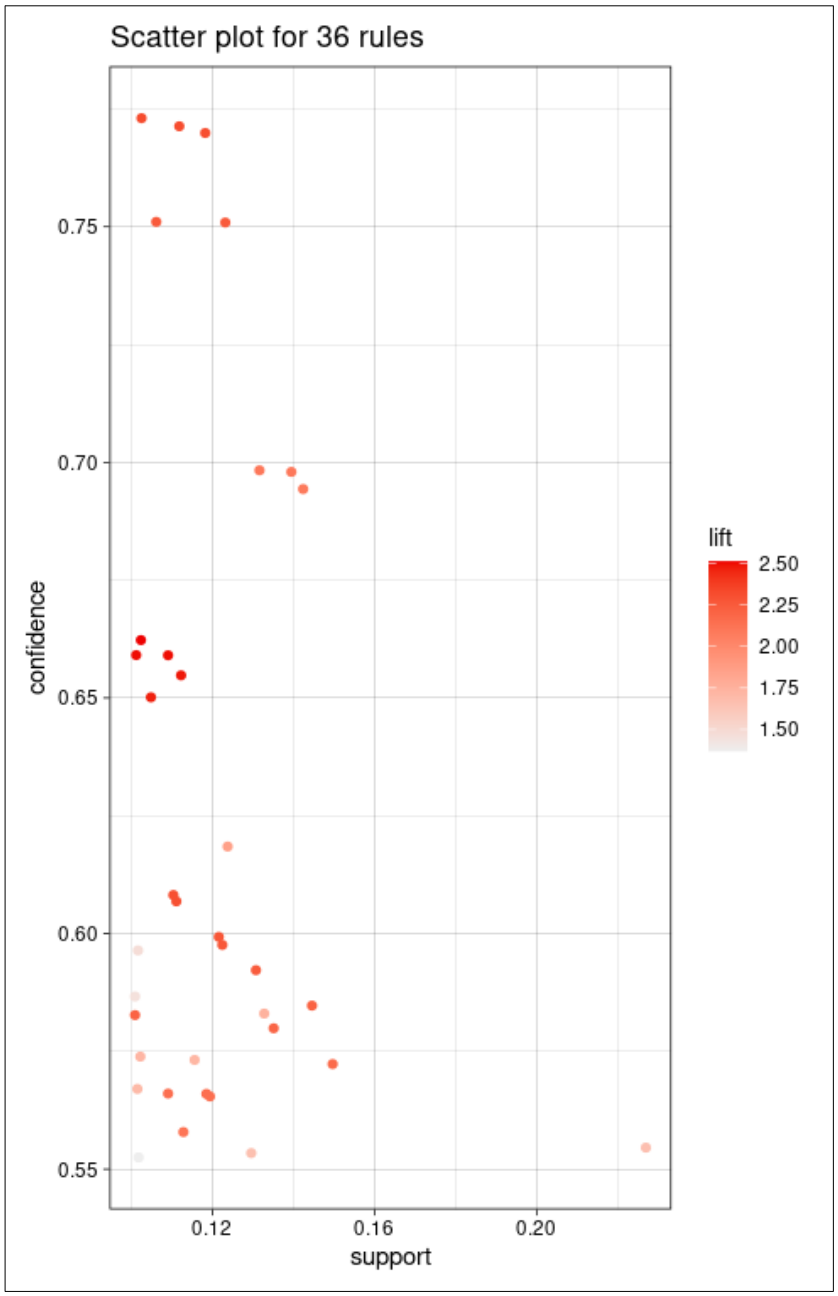
```
#interestMeasure(rules_pruned, c("support", "chiSquare", "confidence", "conviction", "cosine",
"coverage", "leverage", "lift", "oddsRatio"), income3L_complete)
interestMeasure(rules_pruned[1:10], c("support",  "confidence", "lift", "chiSquare",  "coverage"),
income3L_complete)
##      support confidence     lift chiSquared  coverage
## 1  0.1023851  0.6622766 2.513142  1030.1318 0.1545957
## 2  0.1012216  0.6590909 2.501054  1005.8511 0.1535777
## 3  0.1090750  0.6590510 2.500902  1099.2254 0.1655032
## 4  0.1122746  0.6547922 2.484741  1122.4564 0.1714660
## 5  0.1048575  0.6501353 2.467070  1018.3179 0.1612856
## 6  0.1025305  0.7730263 2.322118   917.1720 0.1326353
## 7  0.1118383  0.7713139 2.316975  1009.2514 0.1449971
## 8  0.1182373  0.7698864 2.312686  1072.7924 0.1535777
## 9  0.1103839  0.6081731 2.307836   933.1837 0.1815009
## 10 0.1111111  0.6068308 2.302742   935.9197 0.1831006
```

Los valores por ejemplo de “chi squared” para las 26 primeras reglas son muy altos (también son bastante altos para las últimas 10). Esto indica una relación muy fuerte entre la parte izquierda y derecha de las reglas. Entiendo que por tanto las reglas de asociación obtenidas son validas, no son triviales.

1 Visualización de las reglas

Con la ayuda de la librería arulesViz, vamos a visualizar las reglas obtenidas, para ver si podemos deducir algo más:

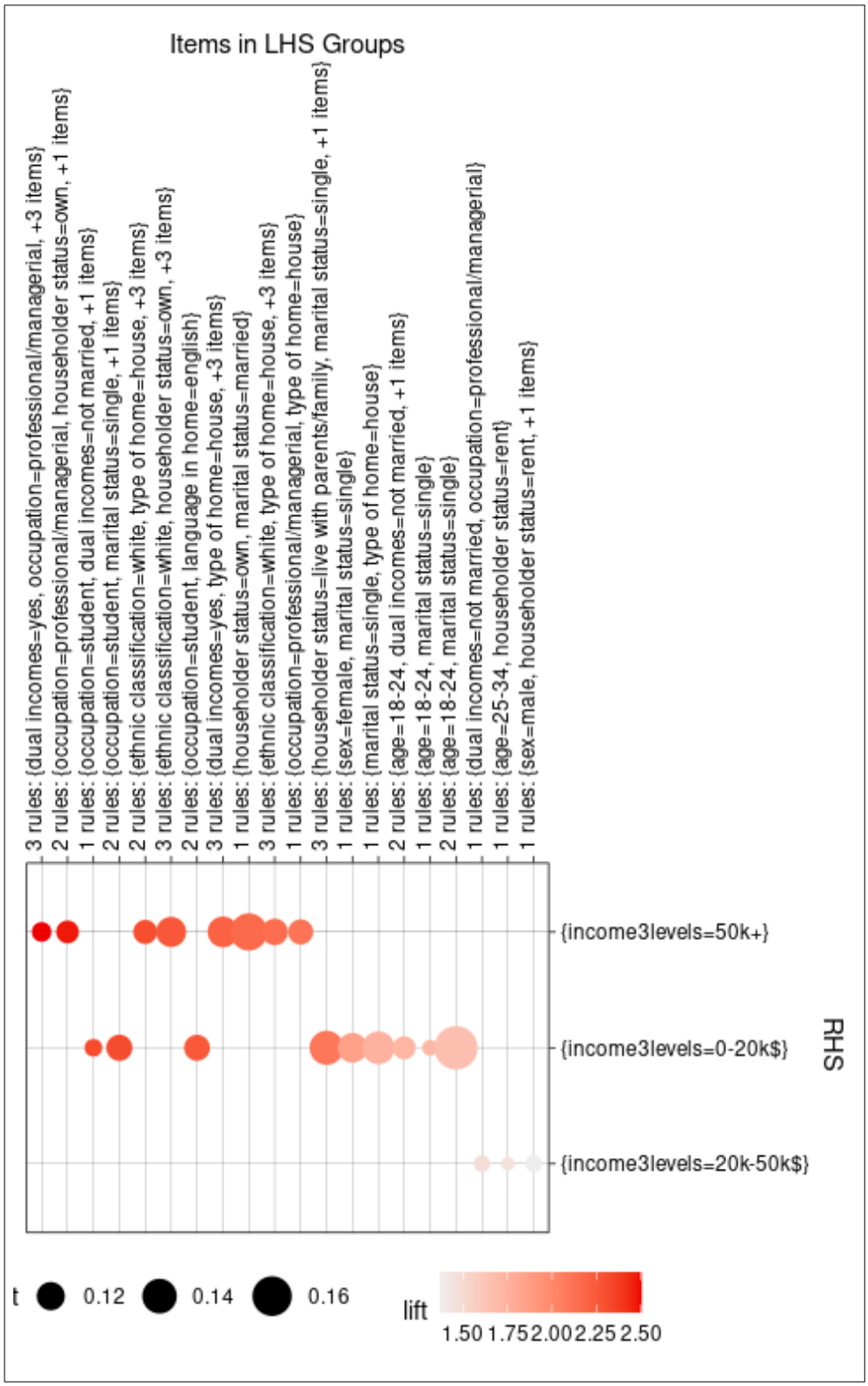
```
plot(rules_pruned)
```



Se puede ver gráficamente que el grado de confianza supera el 55% (lógico, ese fue el parámetro que elegí), y el soporte no es muy alto. Tuve que disminuirlo hasta ese nivel para poder encontrar reglas de asociación relacionadas con los niveles de ingreso.

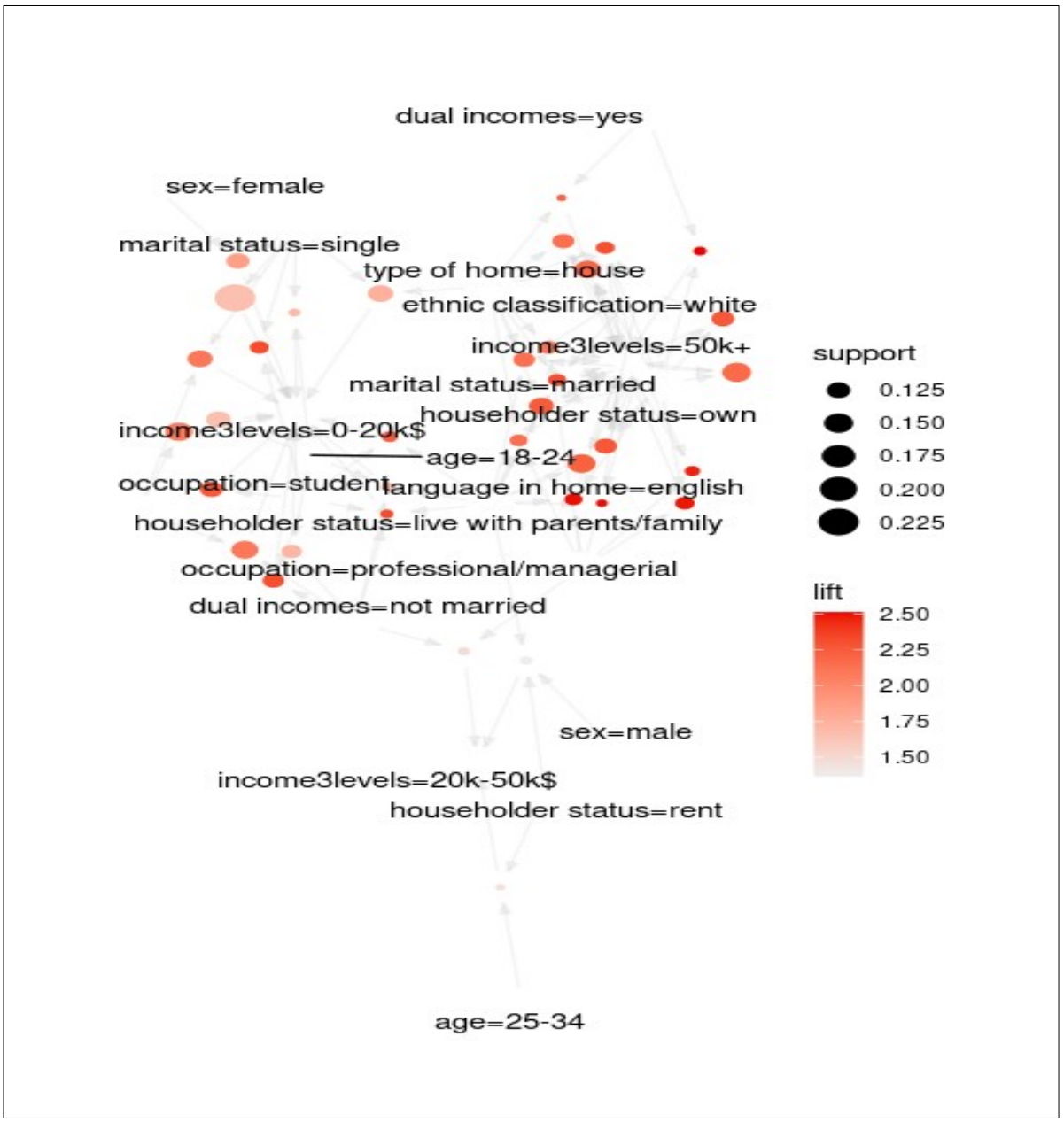
- Gráfico de grupos de reglas:

```
plot(rules_pruned, method="grouped")
```



- Gráfico de conexiones entre reglas:

```
plot(rules_pruned, method="graph")
```



En este gráfico se pueden apreciar* 2 “clusters”, y algunos puntos sueltos en la parte inferior. En el cluster de la parte izquierda se puede ver “estudiante”, “sueldo inferior a 20k $”, “edad 18 a 24”, “vive con los padres” .. En el cluster de la parte derecha “casado”, “sueldo superior a 50k $”, “vivienda en propiedad” … Y en la parte inferior, lo también comentado previamente: las personas con edades entre 25 y 34, que viven de alquiler, tienen un sueldo intermedio de entre 20k y 50k $.

* En el Rstudio, en el monitor, que tiene mucha más anchura y definición, esto se aprecia mucho mejor.

EVALUACIÓN TEMA 9 - AGRUPACIÓN DE DATOS. EL ALGORITMO K-MEANS

La base de datos Acidosis.txt contiene datos de 40 pacientes de un hospital con 6 medidas relativas a la acidez de diferentes fluidos corporales. En particular:

- La columna 1 es el pH del líquido cefalorraquídeo (nanomoles/litro).
- La columna 2 es el pH de la sangre (nanomoles/litro).
- La columna 3 es la cantidad de HCO_3 en el líquido cefalorraquídeo (milimoles/litro).
- La columna 4 contiene los valores de HCO_3 en sangre (milimoles/litro).
- La columna 5 es la presión del CO_2 en el líquido cefalorraquídeo (mmHg).
- La columna 4 es la presión del CO_2 en sangre (mmHg).

Considerando este *dataset*, obtenga los coeficientes de silueta y obtenga el valor de *k* previamente a la ejecución del algoritmo K-Means.

Con los clústeres estimados, realice una representación gráfica de los mismos usando *fviz_cluster()* e interprete los resultados.

1 Paso 1: Carga de los datos

```
# import the CSV file
acidosis_raw <- read.csv(file.path("Chapter09", "Acidosis.txt"), sep = "")
```

2 Paso 2: Exploración y preparación de los datos

Carga de librerías necesarias.

```
if (!require(cluster)) install.packages('cluster', dependencies = T)
library(cluster)

if (!require(factoextra)) install.packages('factoextra', dependencies = T)
library(factoextra)

if (!require(hms)) install.packages('hms', dependencies = T)
library(hms)
```

Estructura y resumen estadístico de los datos:

```
# structure
str(acidosis_raw)

## 'data.frame':    40 obs. of  6 variables:
##  $ X1: num  39.8 53.7 47.3 41.7 44.7 47.9 48.4 48.4 48.4 41.7 ...
##  $ X2: num  38 37.2 39.8 37.6 38.5 39.8 36.7 35.1 45.7 81.3 ...
##  $ X3: num  22.2 18.7 23.3 22.8 24.8 22 21 23.9 18.6 9.8 ...
##  $ X4: num  23.2 18.5 22.1 22.3 24.4 23.3 21.3 24 14.9 4.2 ...
##  $ X5: num  38.8 45.1 48.2 41.6 48.5 46.2 44.5 50.6 39.4 17.8 ...
##  $ X6: num  36.5 28.3 36.4 34.6 38.8 38.5 32.6 35 28.8 12.9 ...
```

```
# summary
summary(acidosis_raw)
```

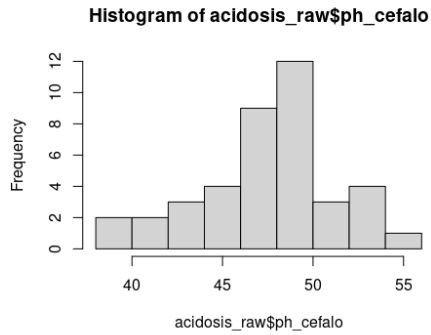
```
##        X1              X2              X3              X4
##  Min.   :38.50   Min.   :32.80   Min.   : 9.80   Min.   : 4.20
##  1st Qu.:45.67   1st Qu.:37.20   1st Qu.:19.43   1st Qu.:19.00
##  Median :48.05   Median :39.55   Median :22.65   Median :23.30
##  Mean   :47.52   Mean   :41.55   Mean   :21.91   Mean   :22.72
##  3rd Qu.:49.45   3rd Qu.:41.98   3rd Qu.:24.52   3rd Qu.:26.93
##  Max.   :54.90   Max.   :81.30   Max.   :30.40   Max.   :34.80
##        X5              X6
##  Min.   :17.80   Min.   :12.90
##  1st Qu.:40.38   1st Qu.:30.25
##  Median :45.25   Median :35.70
##  Mean   :45.57   Mean   :36.44
##  3rd Qu.:52.58   3rd Qu.:42.30
##  Max.   :69.00   Max.   :61.40
```

Creo que las variables no difieren demasiado en sus valores (todos están entre 0 y 100). De todas maneras vamos a cambiar los nombres de las variables por unos que sean descriptivos y a normalizar:

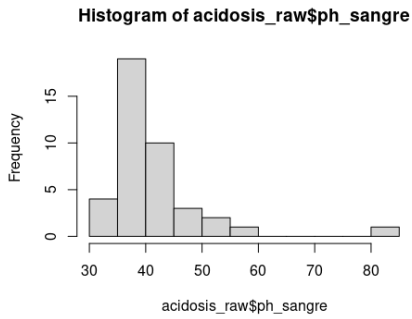
```
# rename columns
colnames(acidosis_raw) <- c("ph_cefalo", "ph_sangre", "hco3_cefalo",
"hco3_sangre", "pres_co2_cefalo", "pres_co2_sangre")
```

Comprobamos las distribuciones de las variables:

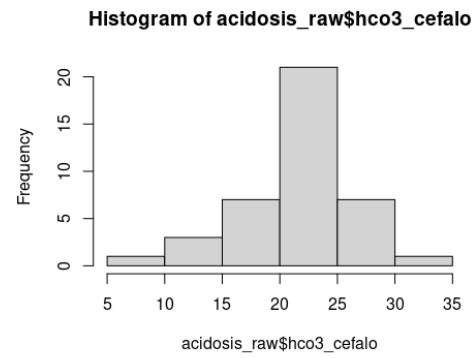
```
hist(acidosis_raw$ph_cefalo)
```



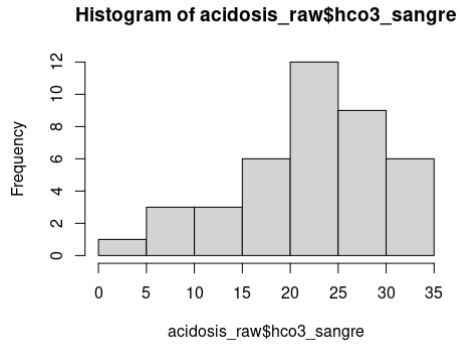

```
hist(acidosis_raw$ph_sangre)
```



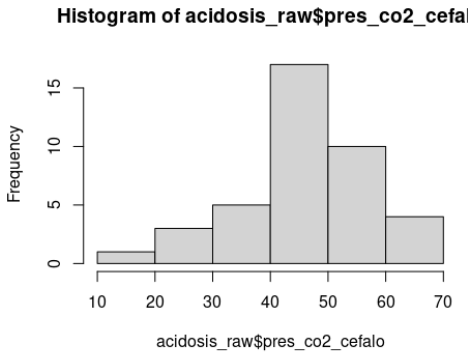

```
hist(acidosis_raw$hco3_cefalo)
```



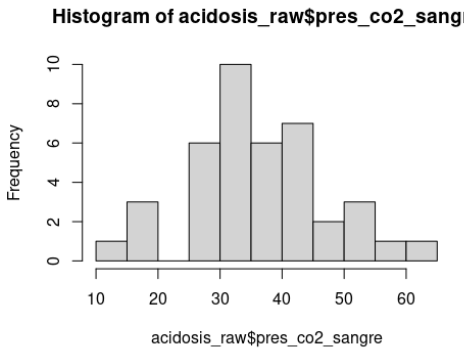

```
hist(acidosis_raw$hco3_sangre)
```



```
hist(acidosis_raw$pres_co2_cefalo)
```




```
hist(acidosis_raw$pres_co2_sangre)
```



Y aplicamos la función scale():

```
# scale the dataset
datos_scaled <- as.data.frame(scale(acidosis_raw))

summary(datos_scaled)

##    ph_cefalo         ph_sangre         hco3_cefalo       hco3_sangre
##  Min.   :-2.4062   Min.   :-1.01403   Min.   :-2.5299   Min.   :-2.49793
##  1st Qu.:-0.4932   1st Qu.:-0.50441   1st Qu.:-0.5200   1st Qu.:-0.50147
##  Median : 0.1400   Median :-0.23223   Median : 0.1535   Median : 0.07858
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.5132   3rd Qu.: 0.04865   3rd Qu.: 0.5450   3rd Qu.: 0.56757
##  Max.   : 1.9663   Max.   : 4.60340   Max.   : 1.7718   Max.   : 1.62987


##  pres_co2_cefalo    pres_co2_sangre
##  Min.   :-2.41696   Min.   :-2.19918
##  1st Qu.:-0.45215   1st Qu.:-0.57829
##  Median :-0.02785   Median :-0.06913
##  Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.60968   3rd Qu.: 0.54746
##  Max.   : 2.03923   Max.   : 2.33184
```

3 Paso 3: Ejecución del algoritmo k-means

Vamos a de determinar el número óptimo de clústeres, utilizando como se explica en el material complementario la métrica denominada coeficiente de silueta, la cual permite evaluar la calidad del agrupamiento obtenido.
El valor de este coeficiente es un valor comprendido entre -1 y +1. Cuanto más cercano sea su valor a +1, mejor será la calidad del agrupamiento.

Creamos un bucle variando el parámetro k y obtenemos los coeficientes de silueta.

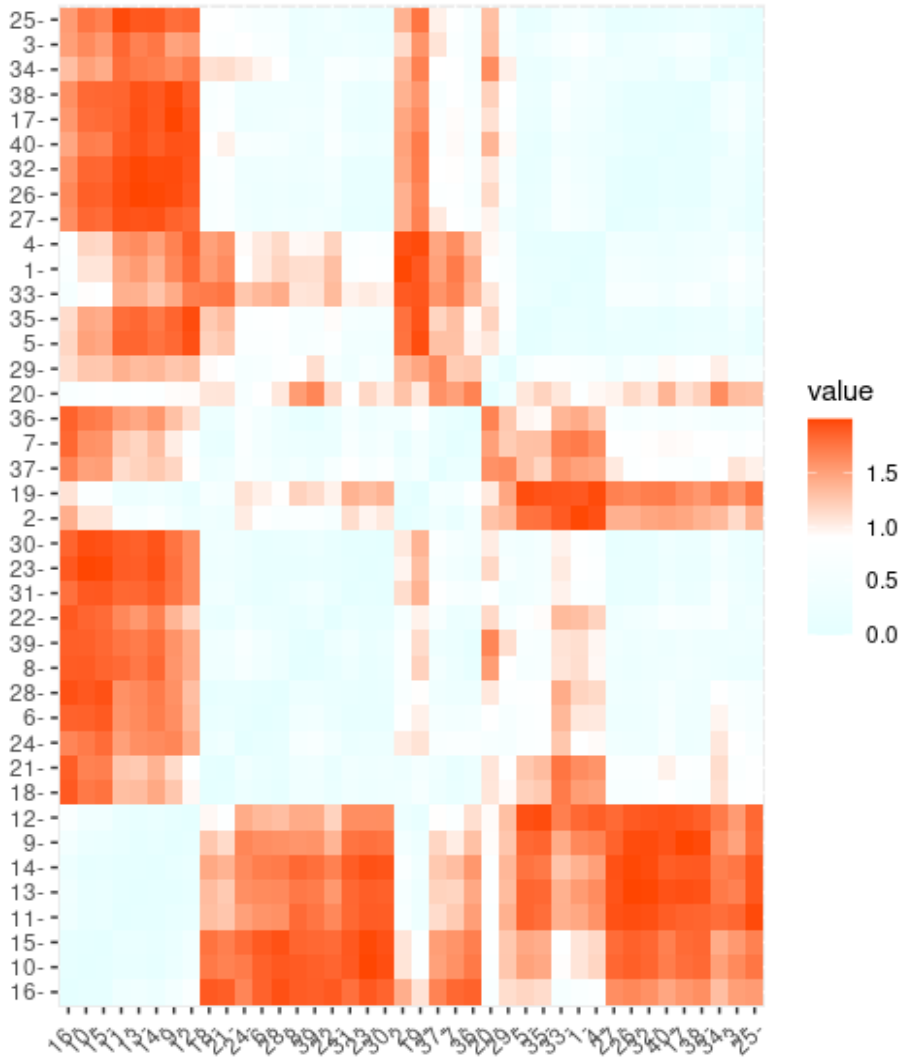
Primero se necesita calcular la matriz de distancias:

```
d <- dist(datos_scaled) # This function computes and returns the distance matrix computed
                        # by using the specified distance measure to compute the distances between
the rows of a data matrix.
#d
```

```
[
   Output:
        1          2          3          4          5          6          7          8     ...
2 3.9453991
3 2.1877121 2.1923064
4 0.6165141 3.4165874 1.6321668
5 1.6703134 3.0143345 0.8651709 1.1915932
6 2.2713136 2.0720159 0.4407694 1.7710169 1.0748526
7 2.4056835 1.5929585 0.8288452 1.8600744 1.5092490 0.7673272
8 2.5655752 2.0991379 0.7242077 1.9987717 1.1514652 0.8567462 0.9305192
…
]
```

Otra manera* de calcular la matriz de distancias:

```r
res.dist <- get_dist(datos_scaled, stand = TRUE, method = "pearson")  # Compared to the standard dist() function, it supports correlation-based distance measures including "pearson", "kendall" and "spearman" methods.

fviz_dist(res.dist, gradient = list(low = "#E0FFFF", mid = "white", high = "#FF4500"))
```



*Otra función, get_dist(), encontrada aquí: https://warin.ca/posts/rcourse-clustering-with-r/

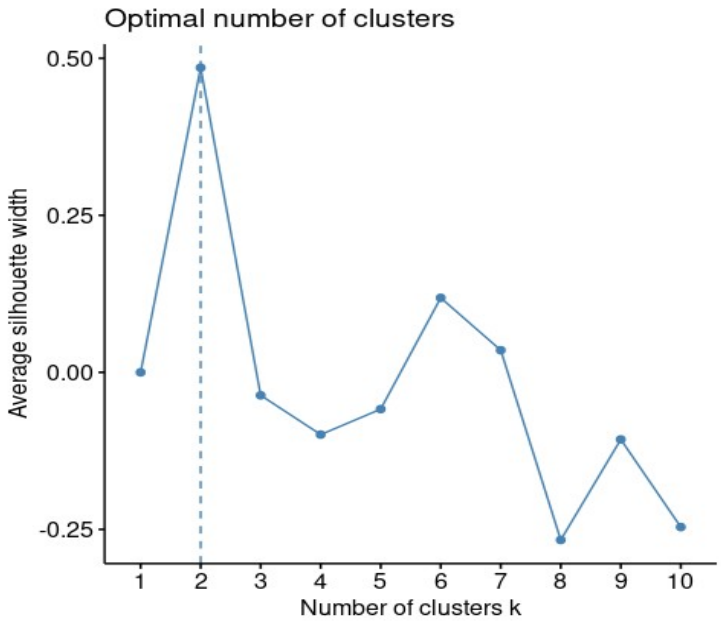
```
# silhouette score
set.seed(911)

avgS <- c()

for(k in 2:9) {
  cl <- kmeans(datos_scaled, centers = k, iter.max = 200)
  s <- silhouette(cl$cluster, d)
  avgS <- c(avgS, mean(s[,3]))
}

data.frame(nClus = 2:9, Silh = avgS)

##   nClus      Silh
## 1     2 0.5171241
## 2     3 0.3890987
## 3     4 0.2849339
## 4     5 0.3201198
## 5     6 0.3288259
## 6     7 0.3010176
## 7     8 0.3088594
## 8     9 0.2922981
```

El resultado obtenido muestra que la mejor opción se da para k=2, por ser esta opción la que presenta un coeficiente de silueta más elevado. (Es positivo, y está a medio camino entre el 0 y el 1 …)

Habiendo determinado el número óptimo de clústeres, pasamos a ejecutar el algoritmo K-Means, especificando centers=2 para establecer el valor del número de clústeres.

Otra manera para obtener el número óptimo de clústeres:

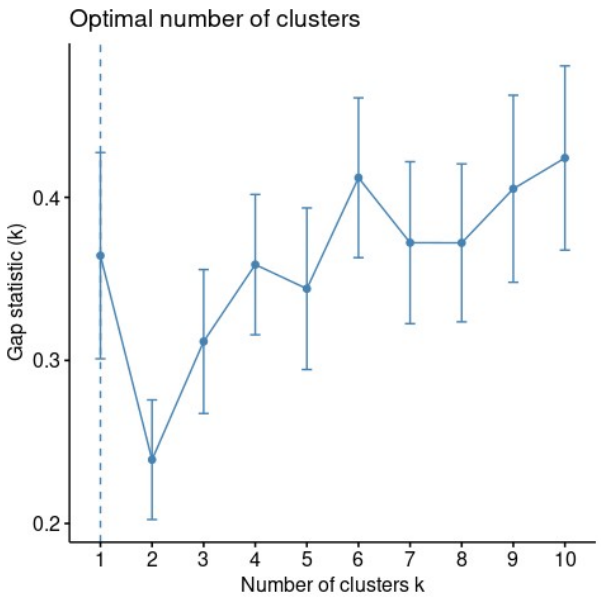
```
fviz_nbclust(datos_scaled, diss=res.dist, kmeans, method = "silhouette")  #using dist obtained by get_dist -if not, we get 3 clusters-
```

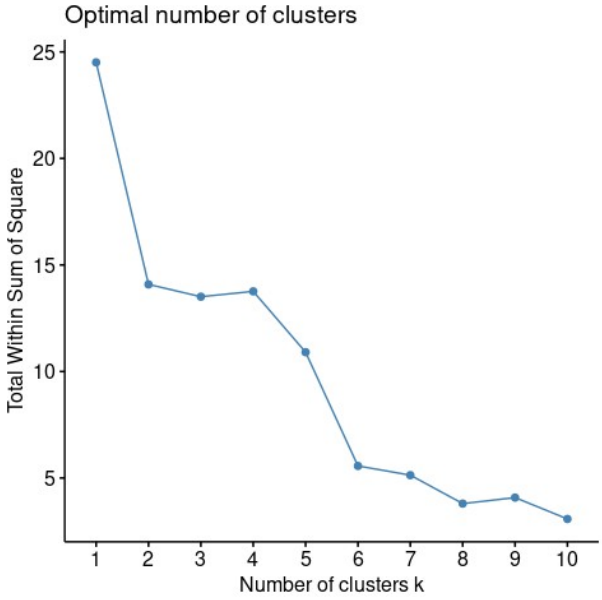


Con esta otra función, usando la misma métrica de los coeficientes de silueta, también se obtiene el mismo número de clústeres.

-Usando las otras dos métricas disponibles (gap_stat y wss):

```
fviz_nbclust(datos_scaled, diss=res.dist, kmeans, method = "gap_stat")
fviz_nbclust(datos_scaled, diss=res.dist, kmeans, method = "wss")
```





Estas otras 2 gráficas, me parecen indicar que a mayor número de clústeres, mejor será la agrupación.

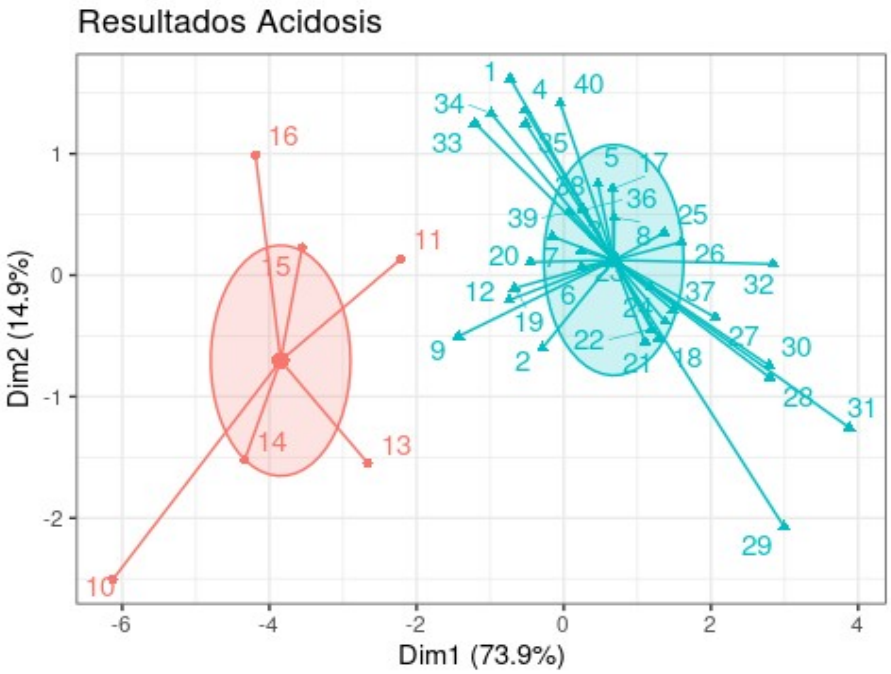
Probamos con 2 clústeres:

```
clusters <- kmeans(datos_scaled, centers = 2, nstart = 50)

clusters
## K-means clustering with 2 clusters of sizes 6, 34
##
## Cluster means:
##    ph_cefalo  ph_sangre hco3_cefalo hco3_sangre pres_co2_cefalo pres_co2_sangre
## 1 -0.8509424  1.6595567  -1.8163916  -1.7852282      -1.7569480      -1.5094034
## 2  0.1501663 -0.2928629   0.3205397   0.3150403       0.3100497       0.2663653
##
## Clustering vector:
##  [1] 2 2 2 2 2 2 2 2 2 1 1 2 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [39] 2 2
##
## Within cluster sum of squares by cluster:
## [1]  23.52067 102.26958
##  (between_SS / total_SS =  46.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

1 Visualización de los clústeres

Pasamos a representar gráficamente los clústeres obtenidos. Para ello, usaremos la función fviz_cluster() incluida en la librería factoextra:

```
fviz_cluster(object = clusters, data = datos_scaled, show.clust.cent = TRUE, ellipse.type =
"euclid", star.plot = TRUE, repel = TRUE) +
  labs(title = "Resultados Acidosis") +
  theme_bw() +
  theme(legend.position = "none")
```



Se ve que hay dos grupos de pacientes, uno pequeño con solo 6 miembros, para el que todas las variables a excepción del ph en sangre tienen valores inferiores a los del otro grupo más numeroso.

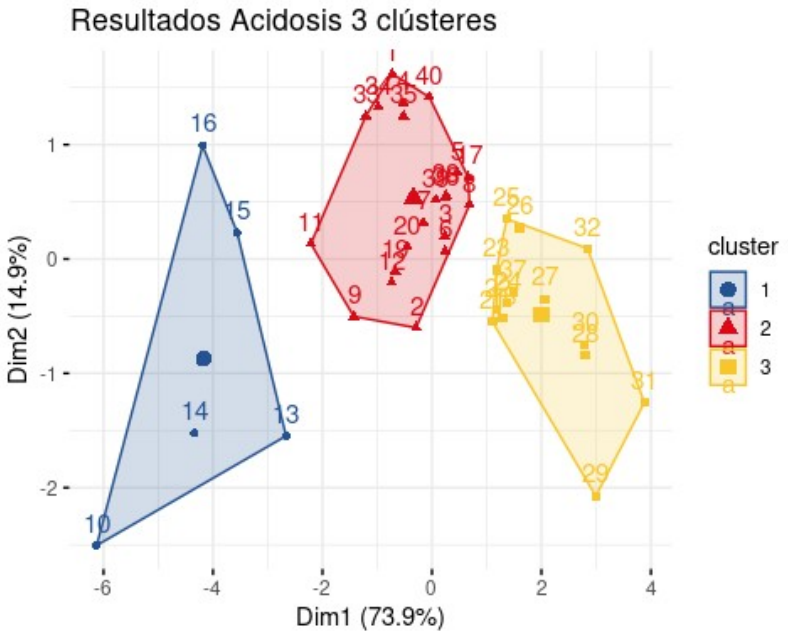
De todas maneras, aunque el coeficiente de silueta es claro (con las dos funciones utilizadas), me parece que en esta agrupación hay muchos elementos muy alejados del centroide de su clúster. Vamos a probar también a examinar cómo sería resultado con 3 (y al final con 5) clústeres:

```
clusters3 <- kmeans(datos_scaled, centers = 3, nstart = 25)

clusters3
## K-means clustering with 3 clusters of sizes 5, 21, 14
##
## Cluster means:
##    ph_cefalo   ph_sangre hco3_cefalo hco3_sangre pres_co2_cefalo  pres_co2_sangre
## 1 -0.9504782  1.96494447  -1.9117531  -1.9340627      -1.8599395       -1.6311646
## 2 -0.3050189 -0.41230383  -0.1343910  -0.1713011      -0.2012997       -0.2759981
## 3  0.7969848 -0.08331014   0.8843555   0.9476883       0.9662137        0.9965559
##
## Clustering vector:
##  [1] 2 2 2 2 2 2 2 2 2 1 2 2 1 1 1 1 2 3 2 2 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 3 2 2 2
##
## Within cluster sum of squares by cluster:
```

```
## [1] 18.72524 29.97298 25.44634
##  (between_SS / total_SS =  68.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Seguramente sea solo una impresión mía, pero los resultados de las variables me hacen pensar que están mejor separadas con 3 clústeres que con 2.

```
palette <- c("#255293", "#db0a16", "#f8c72d")

fviz_cluster(clusters3, data = datos_scaled, show.clust.cent = TRUE, ellipse.type = "convex",
palette = palette, ggtheme = theme_minimal(), main = "Resultados Acidosis 3 clústeres")
```



Con 3 clústeres, me parece que los datos están repartidos de manera más compacta, más homogénea. Los elementos no están tan alejados del centroide del clúster al que pertenecen como pasa con 2 clústeres.

El grupo más pequeño que había con dos clústeres ha perdido solo un elemento (el paciente 11), mientras que el anterior grupo grande se ha dividido en 2 grupos más parecidos.

Podemos añadir la variable cluster al dataset original:

```
acidosis_raw_cluster <- acidosis_raw
acidosis_raw_cluster['cluster'] <- clusters3$cluster
```

Y así examinarlos por separado fácilmente:

```
# list elements of a group
x=1
groupX <- acidosis_raw_cluster[acidosis_raw_cluster$cluster==x, ]
groupX
##    ph_cefalo ph_sangre hco3_cefalo hco3_sangre pres_co2_cefalo pres_co2_sangre  cluster
## 10      41.7      81.3         9.8         4.2             17.8            12.9        1
```

```
## 13      49.6      55.0        14.6        11.3            31.8            25.7        1
## 14      47.3      59.4        10.4         7.5            21.5            18.6        1
## 15      42.7      49.0        15.3         9.5            26.9            19.0        1
## 16      38.5      47.9        13.7         9.4            23.0            18.7        1
```

```
# summary of a group
summary(groupX)
##
##  ph_cefalo       ph_sangre       hco3_cefalo     hco3_sangre
##  Min.   :38.50   Min.   :47.90   Min.   : 9.80   Min.   : 4.20
##  1st Qu.:41.70   1st Qu.:49.00   1st Qu.:10.40   1st Qu.: 7.50
##  Median :42.70   Median :55.00   Median :13.70   Median : 9.40
##  Mean   :43.96   Mean   :58.52   Mean   :12.76   Mean   : 8.38
##  3rd Qu.:47.30   3rd Qu.:59.40   3rd Qu.:14.60   3rd Qu.: 9.50
##  Max.   :49.60   Max.   :81.30   Max.   :15.30   Max.   :11.30

##  pres_co2_cefalo pres_co2_sangre  cluster
##  Min.   :17.8    Min.   :12.90    Min.   :1
##  1st Qu.:21.5    1st Qu.:18.60    1st Qu.:1
##  Median :23.0    Median :18.70    Median :1
##  Mean   :24.2    Mean   :18.98    Mean   :1
##  3rd Qu.:26.9    3rd Qu.:19.00    3rd Qu.:1
##  Max.   :31.8    Max.   :25.70    Max.   :1
```

Ahora ya sólo quedaría preguntarle a un médico qué hacer con estos pacientes.

NOTA:

He investigado un poco sobre el ph en sangre. Quería convertir los valores de esa columna en valores de 0 a 14. He visto que los valores entre 20 y 80 nanomoles/l se corresponden con un ph entre 7.1 y 7.7, que es un nivel normal. (https://acid-base.com/ph-playground).

```
ph <- -log10((acidosis_raw$ph_sangre)*1e-9)
ph
##  [1] 7.420216 7.429457 7.400117 7.424812 7.414539 7.400117 7.435334 7.454693
##  [9] 7.340084 7.089909 7.369572 7.374688 7.259637 7.226214 7.309804 7.319664
## [17] 7.442493 7.402305 7.396856 7.383000 7.389340 7.383000 7.403403 7.388277
## [25] 7.414539 7.420216 7.377786 7.404504 7.269218 7.379864 7.358526 7.413413
## [33] 7.429457 7.469800 7.449772 7.469800 7.471083 7.428291 7.438899 7.484126
```

Entiendo por tanto que todos estos pacientes tienen un ph en sangre correcto. Todos menos uno, aunque sea por muy poco:

```
#ph<7.1
acidosis_raw[ph<7.1, ]
##    ph_cefalo ph_sangre hco3_cefalo hco3_sangre pres_co2_cefalo pres_co2_sangre
## 10      41.7      81.3         9.8         4.2            17.8            12.9
```

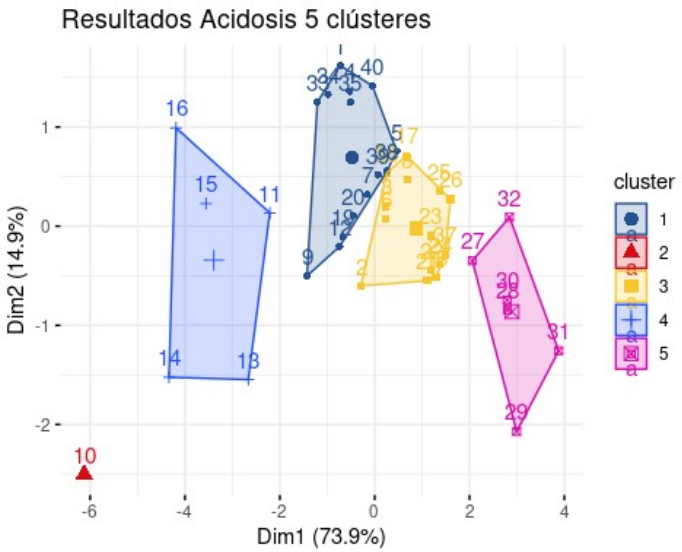
```
patient_10 <- acidosis_raw[10, ]
patient_10['ph14'] <- ph[10]
patient_10
##    ph_cefalo ph_sangre hco3_cefalo hco3_sangre pres_co2_cefalo pres_co2_sangre       ph14
## 10      41.7      81.3         9.8         4.2            17.8            12.9   7.089909
```

El paciente número 10 es precisamente el que sale en el primer gráfico en la esquina inferior izquierda, muy separado de todos los demás. Incluso de los otros miembros de su clúster.

Si elegimos 5 clústeres, ese paciente es el primero que efectivamente es separado del resto:

```
set.seed(9)

clusters5 <- kmeans(datos_scaled, centers = 5, nstart = 50)
clusters5

## K-means clustering with 5 clusters of sizes 14, 1, 14, 5, 6
##
## Cluster means:
##    ph_cefalo  ph_sangre hco3_cefalo hco3_sangre pres_co2_cefalo pres_co2_sangre
## 1 -0.5722675 -0.4009973  -0.1358826  -0.2037402      -0.2783879      -0.2826711
## 2 -1.5530254  4.6034022  -2.5298622  -2.4979255      -2.4169640      -2.1991762
## 3  0.5246579 -0.3918968   0.3190481   0.4167786       0.3917822       0.2231474
## 4 -0.7105258  1.0707876  -1.6736975  -1.6426887      -1.6249448      -1.3714489
## 5  0.9620315  0.1905295   1.3890054   1.2881386       1.4923614       1.6482922
##
## Clustering vector:
##  [1] 1 3 3 1 1 3 1 3 1 2 4 1 4 4 4 4 3 3 1 1 3 3 3 3 3 3 5 5 5 5 5 5 1 1 1 3 3 1
## [39] 1 1
##
## Within cluster sum of squares by cluster:
## [1] 15.700970  0.000000 17.305599 10.215633  5.941569
##  (between_SS / total_SS =  79.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
palette <- c("#255293", "#db0a16", "#f8c72d", "#2552FF", "#db0aAA")

fviz_cluster(clusters5, data = datos_scaled, show.clust.cent = TRUE, ellipse.type = "convex",
palette = palette, ggtheme = theme_minimal(), main = "Resultados Acidosis 5 clústeres")
```



Yo no soy médico, pero sería al primer paciente al que le dedicaría mi atención si lo fuera.

El ratio "between_SS/total_SS" (SS=‘sum of squares’) ha pasado de 46.2% con 2 clústeres a 68.3% con 3, y a 79.0% con 5.
Este ratio indica cómo de buena es la agrupación: cómo de cohesionados están los clústeres, y cómo de separados entre ellos.
Se ha mejorado por tanto la agrupación inicial de 2 clústeres.

EVALUACIÓN TEMA 10 - EVALUACIÓN DE MODELOS DE MACHINE LEARNING

Utilizando las medidas de rendimiento analizadas en el capítulo 10 del libro, tales como:

- Prediction Accuracy
- Error Rate
- Kappa Statistic
- Sensitivity
- Specifity
- Precision
- Recall
- F-Measure
- ROC Curves

Realice un análisis detallado de cualquiera de los modelos obtenidos en las pruebas de evaluación de los temas 3, 4, 5 o 7. Comente los resultados obtenidos.

1 Paso 1 – Carga de los datos

Vamos a realizar el análisis sobre los resultados del tema 7 (SVM)

```
# import the CSV file
bank_raw <- read.csv(file.path("Chapter07/Bank", "bank.csv"), sep = ";",
stringsAsFactors = TRUE)
```

2 Pasos 2,3 y 4 (Exploración y preparación de los datos, entrenamiento, evaluación)

Importar librerías necesarias:

```
#https://www.jstatsoft.org/article/view/v011i09
if (!require(kernlab)) install.packages('kernlab', dependencies = T)
library(kernlab)

if (!require(gmodels)) install.packages('gmodels', dependencies = T)  # cross tables
library(gmodels)

if (!require(caret)) install.packages('caret', dependencies = T)
library(caret)

if (!require(pROC)) install.packages('pROC', dependencies = T)
library(pROC)

if (!require(ggplot2)) install.packages('ggplot2', dependencies = T)
library(ggplot2)
```

Preparamos el dataset:

```
set.seed(12345)

#scale numeric variables (neither day nor month)
maxs <- apply(bank_raw[c(1,6,12,13,14,15)], 2, max)
mins <- apply(bank_raw[c(1,6,12,13,14,15)], 2, min)

bank_norm <- data.frame(scale(bank_raw[c(1,6,12,13,14,15)], center = mins, scale = maxs - mins))

#hot encoding of categorical features
dummies <- dummyVars(" ~ job + marital + education + default + housing + loan + contact + poutcome",
data = bank_raw)
bank_hot_encoded_feat <-  data.frame(predict(dummies, newdata = bank_raw))

#encoding month (name to number)
month_to_number <- function(month_name) {
  month_and_number <-
c
("jan"=1,"feb"=2,"mar"=3,"apr"=4,"may"=5,"jun"=6,"jul"=7,"aug"=8,"sep"=9,"oct"=10,"nov"=11,"dec"=12)
  return(month_and_number[as.character(month_name)])
}
bank_raw$month_num <- sapply(bank_raw$month, month_to_number)


#put all features in the same dataframe
bank_processed <-
cbind(bank_norm,as.numeric(bank_raw$day),bank_raw$month_num,bank_hot_encoded_feat,bank_raw$y)
names(bank_processed)[7:8] <- c("day","month")
names(bank_processed)[41] <- "y"
#head(bank_processed,5)
```

Finalmente, creamos los conjuntos de entrenamiento y validación:

```
#Set seed to make the process reproducible
set.seed(12345)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(bank_processed$y, times=1, p=.75, list=FALSE)

#create training set
bank_processed_train <- bank_processed[train_indices, ]

#create testing set
bank_processed_test  <- bank_processed[-train_indices, ]

#view number of rows in each set
#nrow(bank_processed_train)  # 3391
#nrow(bank_processed_test)   # 1130
```

Entrenamiento del modelo:

```
#train rbfdot
set.seed(12345)
model_rbfdot <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot")
```

Predicción del modelo:

```
set.seed(12345)

#Prediction for SVM rbfdot
model_rbfdot <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot", prob.model = TRUE)
prediction_rbfdot <- predict(model_rbfdot, bank_processed_test)
prediction_rbfdot_prob <- predict(model_rbfdot, bank_processed_test,
                                          type= "probabilities")

CrossTable(x = bank_processed_test$y, y = prediction_rbfdot, prop.chisq = FALSE)
##
##
```

```
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  1130
##
##
##                         | prediction_rbfdot
## bank_processed_test$y |        no |       yes | Row Total |
## ----------------------|-----------|-----------|-----------|
##                    no |       987 |        13 |      1000 |
##                       |     0.987 |     0.013 |     0.885 |
##                       |     0.902 |     0.361 |           |
##                       |     0.873 |     0.012 |           |
## ----------------------|-----------|-----------|-----------|
##                   yes |       107 |        23 |       130 |
##                       |     0.823 |     0.177 |     0.115 |
##                       |     0.098 |     0.639 |           |
##                       |     0.095 |     0.020 |           |
## ----------------------|-----------|-----------|-----------|
##          Column Total |      1094 |        36 |      1130 |
##                       |     0.968 |     0.032 |           |
## ----------------------|-----------|-----------|-----------|
##
##
```

3 Evaluación detallada de los resultados

El propósito en el tema 7 era analizar (predecir) si un cliente, en base a sus características, contratará un determinado producto bancario. Por esta razón, a la hora de obtener la matriz de confusión y todas las demás variables estadísticas, consideramos el “sí” como la clase positiva”.

Otro factor a tener en cuenta, es que se trataba de un típico caso de dataset no balanceado:

```
#Summary
summary(bank_raw$y)
##   no  yes
## 4000  521

round(prop.table(table(bank_raw$y))*100, digits = 2)
##
##    no   yes
## 88.48 11.52
```

Hay muchísimas observaciones de una clase (no: 88.48%), y muy pocas de la otra (sí: 11.52%). Esto tiene implicaciones que iremos detallando.

De la matriz de confusión:

```
TP <-  23
TN <- 987
FP <-  13  # type I error
FN <- 107  # type II error
Total <- TP + TN + FP + FN
```

- Exactitud (**“accuracy”**)

Es la proporción de clasificados correctamente sobre el total:

```
acccuracy <- (TP + TN) / (TP + FP + TN + FN)
round(acccuracy,3)
## [1] 0.894
```

La exactitud no tiene en cuenta si un dataset está o no balanceado. Aquí el porcentaje de acierto para la clase negativa mayoritaria compensa el de la clase positiva y hace que la exactitud esté cerca del 90%.

- Error

Es el “complementario” de la exactitud. Lógicamente se calcula:

```
error <- 1 - acccuracy
round(error,3)
## [1] 0.106
```

- Precisión

La precisión es la proporción de predicciones de una clase que son realmente de esa clase:

```
precision <- TP / (TP + FP)
round(precision,3)
## [1] 0.639
```

Al tratarse de un dataset descompensado, y ser la clase positiva mucho menos numerosa, se obtiene una precisión bastante baja.

- Recall (Recuperación, Tasa de verdaderos positivos (TPR))

La recuperación es la proporción de elementos de una clase que fueron clasificados correctamente como de esa clase.

```
recall <- TP / (TP + FN)
round(recall,3)
## [1] 0.177
```

Y un recall todavía más bajo.

- F1 score (Puntuación F1)

La puntuación F1 da una medida de la exactitud de un modelo de clasificación binaria combinando la precisión y la recuperación: es la media armónica de la precisión y la recuperación. Por tanto se calcula de la siguiente manera:

```
#F1 = 2 x (precision x recall) / (precision + recall) = 2TP / (2TP + FP + FN )
f1_score <- 2 * (precision * recall) / (precision + recall)
round(f1_score,3)
## [1] 0.277
```

Vemos que también tiene un valor muy bajo.

- Sensibilidad (“Sensitivity”, “True positive rate”)

También llamada “fracción de verdaderos positivos”, pone el foco en los casos positivos. Su formulación matemática coincide con la del recall/recuperación:

```
sensitivity <- TP / (TP + FN)
round(sensitivity,3)
## [1] 0.177
```

- Especificidad (“Especifity”, “True negative rate”)

La especifidad en cambio pone el foco en los casos negativos. Es la “fracción de verdaderos negativos”.

```
specifity <- TN / (TN + FP)
round(specifity,3)
## [1] 0.987
```

Tenemos un dataset no balanceado, con muchos más casos negativos que positivos. El modelo aprende mejor por tanto a detectar estos casos negativos que los positivos.

- Kappa

Esta variable sirve para ajustar el valor de la exactitud, dado que se puede dar el caso de que parte de esa exactitud se deba simplemente al azar. Es considerada una medida más robusta y fiable que el simple valor de la exactitud.

La Kappa de Cohen es una medida de la fiabilidad con la que dos ‘evaluadores’ miden lo mismo.

Su fórmula es la siguiente:

k = (Pr(a) - Pr(e)) / (1 - Pr(e))

Donde:

Pr(a) es el acuerdo observado relativo entre los observadores.

Pr(e) es la probabilidad hipotética de acuerdo por azar, utilizando los datos observados para calcular las probabilidades de que cada observador clasifique aleatoriamente cada categoría.

Si los evaluadores están completamente de acuerdo, entonces será κ = 1. Si no hay acuerdo ninguno entre los evaluadores distinto al que cabría esperar por azar (según lo definido por Pr(e)), será κ = 0.

En nuestro caso teníamos:

bank_processed_test$y	prediction_rbfdot no	prediction_rbfdot yes	Row Total
no	987	13	1000
	0.987	0.013	**0.885**
	0.902	0.361	
	0.873	0.012	
yes	107	23	130
	0.823	0.177	**0.115**
	0.098	0.639	
	0.095	**0.020**	
Column Total	1094	36	1130
	0.968	**0.032**	

```
pr_a <- 0.873 + 0.020   # this is accuracy = 0.892
pr_a
## [1] 0.893
pr_e = 0.115*0.032 + 0.885*0.968 # probability of both positive + probability of both negative:
# (130/1130)*(36/1130) + (1000/130)*(1094/1130)
pr_e
## [1] 0.86036

k = (pr_a - pr_e) / (1 - pr_e)
k
## [1] 0.2337439
```

Se obtiene un valor de k muy pequeño.

(Coincide con el valor devuelto por la función confusionMatrix del paquete caret. Ver más abajo en el anexo de esta tarea).

Otra manera:

El valor devuelto por la función Kappa del paquete **vcd** (Visualizing Categorical Data) es prácticamente igual:

```
if (!require(vcd)) install.packages('vcd', dependencies = T)
library(vcd)

Kappa(table(bank_processed_test$y, prediction_rbfdot))
##             value     ASE     z  Pr(>|z|)
## Unweighted 0.2391 0.04475 5.344 9.079e-08
## Weighted   0.2391 0.04475 5.344 9.079e-08
```

- Coeficiente de correlación de Matthews

El coeficiente de correlación de Matthews coincide con el coeficiente phi φ o rφ en Estadística. Es una medida de la asociación entre dos variables binarias.

En machine learning es considerado una medida de la calidad de una clasificación binaria.

(También es similar su interpretación a la del coeficiente de correlación de Pearson).

Su fórmula es:

$$MCC = [TP \times TN - FP \times FN] \ / \ \sqrt{[(TP + FP)(TP + FN)(TN + FP)(TN + FN)]}$$

El numerador premia un valor alto de verdaderos positivos y verdaderos negativos, y penaliza los falsos positivos y negativos.

El denominador normaliza el resultado entre -1 y +1.

Valores cercanos a +1 indican una calidad buena de los resultados.

Valores cercanos a 0 indican que la predicción es tan buena como una predicción hecha al azar (es decir, nada buena).

Valores cercanos a -1 indican un gran desacuerdo entre la predicción del modelo y los valores reales.

Entre las ventajas de este coeficiente están que es adecuado para datasets no balanceados (como es el caso de este ejercicio), y que considera todos los elementos de la matriz de confusión.

Comparado con otras métricas, también tiene ventajas:

- la exactitud no aporta tanto valor cuando hay una clase dominante. MCC en cambio tiene en cuenta tanto las predicciones correctas como las erróneas, y en las 2 clases.

- la precisión pone el foco en la clase positiva, el valor de F1 no tiene en cuenta los verdaderos negativos.

```
TPmultTNminusFPmultFN <- (TP * TN) - (FP * FN)
denom <- sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
MCC <- TPmultTNminusFPmultFN / denom
MCC
## [1] 0.2978186
```

También se puede calcular este coeficiente con la función mcc del paquete **mltools**:

```
if (!require(mltools)) install.packages('mltools', dependencies = T)
library(mltools)

mcc(bank_processed_test$y, prediction_rbfdot)
## [1] 0.2978186
```

Los valores obtenidos para Kappa y para MCC son muy parecidos, y bajos.

Ambos indican la pobreza de los resultados que se obtienen con este modelo.

4 Curvas ROC

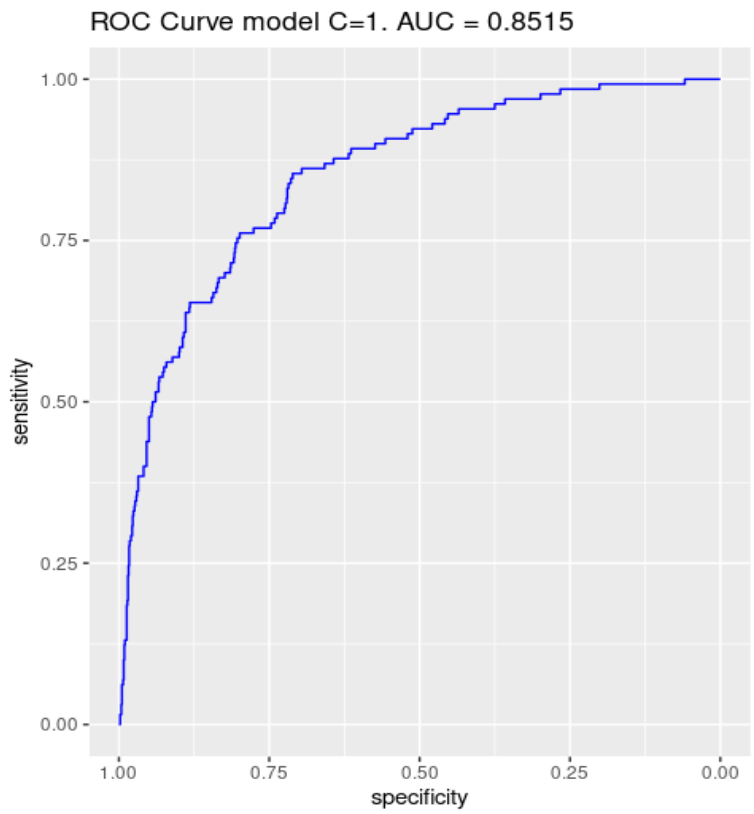
La curva ROC (del inglés “Receiver Operating Characteristic”) sirve en machine learning para evaluar la calidad de un modelo de clasificación binario. Lo hace representando en una gráfica la proporción de verdaderos positivos (VPR = Razón de Verdaderos Positivos) frente a la proporción de falsos positivos (FPR = Razón de Falsos Positivos), también según se varía el umbral de discriminación (valor a partir del cual se decide que un caso es un positivo).

Para crear una curva ROC se necesitan 2 vectores de datos: uno con los valores reales, y otro con las probabilidades predichas por el modelo.

```
set.seed(12345)
bank_roc <- roc(bank_processed_test$y, prediction_rbfdot_prob[,"yes"])
auc <- round(auc(bank_processed_test$y, prediction_rbfdot_prob[,"yes"]),4)
auc
## [1] 0.8515
```

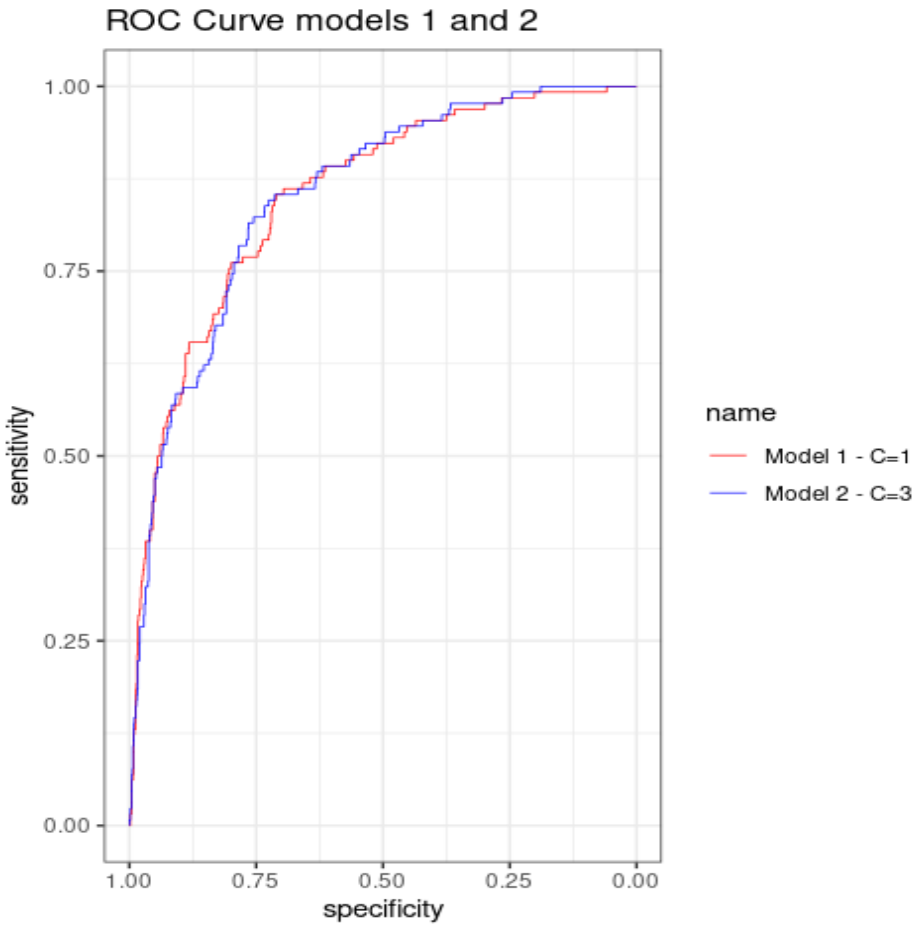
Gráfica:

```
ggroc(bank_roc, colour = 'blue', size = 0.5)  +
  ggtitle(paste0('ROC Curve model C=1. AUC = ', auc))
```



Para mostrar una comparación de las curvas ROC (y la "AUC") entre dos modelos, usaremos el modelo con el parámetro coste=1 y el calculado con coste=3:

```
#Prediction for SVM rbfdot with cost 3
set.seed(12345)

model_rbfdot_C3 <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot", C =
5, prob.model = TRUE)
prediction_rbfdot_C3 <- predict(model_rbfdot_C3, bank_processed_test)
prediction_rbfdot_C3_prob <- predict(model_rbfdot_C3, bank_processed_test, type=
"probabilities")

bank_roc_C3 <- roc(bank_processed_test$y, prediction_rbfdot_C3_prob[,"yes"])
auc_C3 <- round(auc(bank_processed_test$y, prediction_rbfdot_C3_prob[,"yes"]),4)
auc_C3
## [1] 0.8516

ggroc(list(bank_roc,bank_roc_C3), size = 0.25)  +
  ggtitle(paste0('ROC Curve models 1 and 2')) +
  theme_bw() +
  scale_colour_manual(values = c("red", "blue"), labels = c("Model 1 -
C=1","Model 2 - C=3"))
```



De igual manera que los valores Kappa, F1, MCC indican que la calidad de la predición no es muy buena, lo mismo indican estas gráficas: se ve que las líneas de ambos modelos están bastante alejadas de la esquina superior izquierda. (La diferencia entre ellos es prácticamente inexistente. La diferencia de las AUCs es una diezmilésima).

5 Conclusión final

En esta tarea, al examinar con más detalle la exactitud y compararla con otros parámetros como Kappa, MCC y las curvas ROC, se puede ver claramente las limitaciones de esta variable cuando se trata de evaluar un dataset no balanceado (el valor F1 me parece también más “realista” que la exactitud). Al incluir estas otras variables en su cálculo no solo los resultados correctamente clasificados, sus valoraciones de los resultados del modelo sobre el dataset no balanceado son más fiables (realistas).

También son capaces de reflejar las mejoras del modelo. En el anexo a continuación se puede ver cómo la exactitud del modelo con el parámetro coste = 3, solo mejora unas milésimas la del modelo inicial, mientras que Kappa pasa de 0.2391 a 0.3267, y MCC de 0.298 a 0.366.

Observación personal:

Quizá lo más apropiado sería considerar la "exactitud balanceada", no simplemente la exactitud.

6 ANEXO: Matrices de confusión con caret de los 2 modelos

```
#Confusion matrix rbfdot model 1 (C=1)

set.seed(12345)
confusionMatrix(as.factor(prediction_rbfdot), as.factor(bank_processed_test$y), positive="yes", mode =
"everything")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  987 107
##        yes  13  23
##
##                Accuracy : 0.8938
##                  95% CI : (0.8744, 0.9112)
##     No Information Rate : 0.885
##     P-Value [Acc > NIR] : 0.1886
##
##                   Kappa : 0.2391
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.17692
##             Specificity : 0.98700
##          Pos Pred Value : 0.63889
##          Neg Pred Value : 0.90219
##               Precision : 0.63889
##                  Recall : 0.17692
##                      F1 : 0.27711
##              Prevalence : 0.11504
##          Detection Rate : 0.02035
##    Detection Prevalence : 0.03186
##       Balanced Accuracy : 0.58196
##
##        'Positive' Class : yes
##

#Confusion matrix rbfdot model 2 (C=3)

set.seed(12345)
model_rbfdot_C3 <- ksvm(y ~ ., data=bank_processed_train, kernel="rbfdot", C = 3)

prediction_rbfdot_C3 <- predict(model_rbfdot_C3, bank_processed_test)

confusionMatrix(as.factor(prediction_rbfdot_C3), as.factor(bank_processed_test$y), positive="yes", mode =
"everything")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  981  96
##        yes  19  34
##
##                Accuracy : 0.8982
##                  95% CI : (0.8791, 0.9152)
##     No Information Rate : 0.885
##     P-Value [Acc > NIR] : 0.08651
##
##                   Kappa : 0.3267
##
##  Mcnemar's Test P-Value : 1.37e-12
##
##             Sensitivity : 0.26154
##             Specificity : 0.98100
##          Pos Pred Value : 0.64151
##          Neg Pred Value : 0.91086
##               Precision : 0.64151
##                  Recall : 0.26154
##                      F1 : 0.37158
##              Prevalence : 0.11504
##          Detection Rate : 0.03009
##    Detection Prevalence : 0.04690
##       Balanced Accuracy : 0.62127
##
##        'Positive' Class : yes
##
```

EVALUACIÓN TEMA 11 - MEJORANDO UN MODELO DE MACHINE LEARNING

1 Ejercicio 1 – Mejora de un modelo con caret

Utilizando el procedimiento descrito en el capítulo 11 del libro con la librería *caret*, realice una búsqueda de los parámetros óptimos para cualquiera de los modelos de las pruebas de evaluación de los temas 3, 4, 5, 6 o 7. Comente los resultados obtenidos.

Utilizaremos Naive Bayes como en el tema 4.

Examinamos en la documentación los candidatos:

http://topepo.github.io/caret/available-models.html

Candidatos:

Model	Method value	Type	Library	Tuning parameters
Naive Bayes	naive_bayes	Classification	naivebayes	laplace, usekernel, adjust
Naive Bayes	nb	Classification	klaR	fL, usekernel, adjust

Utilizaremos el método naive_bayes de la librería naivebayes, ya que esta librería fue la utilizada en el tema 4.

1 Paso 1: Carga de los datos

```
# import the CSV file
movies_raw <- read.csv(file.path("CSVs", "Movie_pang02.csv"))
```

2 Paso 2: Preparación de los datos

Preparamos el dataset de igual manera a como lo hicimos en el ejercicio 4.

Carga de paquetes:

```
if (!require(tm)) install.packages('tm', dependencies = T) # text mining
library(tm)

if (!require(SnowballC)) install.packages('SnowballC', dependencies = T) # stemming
library(SnowballC)

#if (!require(naivebayes)) install.packages('naivebayes', dependencies = T)
#library(naivebayes)

if (!require(caret)) install.packages('caret', dependencies = T)
library(caret)
```

```
#See the structure
str(movies_raw)

## 'data.frame':    2000 obs. of  2 variables:
##  $ class: chr  "Pos" "Pos" "Pos" "Pos" ...
##  $ text : chr  " films adapted from comic books have had plenty of success   whether they re
about superheroes   batman   super"| __truncated__ " every now and then a movie comes along from a
suspect studio   with every indication that it will be a stinker"| __truncated__ " you ve got mail
works alot better than it deserves to    in order to make the film a success   all they had to"|
__truncated__ "    jaws   is a rare film that grabs your attention before it shows you a single
image on screen    the movie o"| __truncated__ ...
```

La columna “class” es de tipo carácter. Ya que se trata en realidad de una variable categórica, la transformamos nuevamente en un factor:

```
#Convert class into a factor
movies_raw$class <- factor(movies_raw$class)

table(movies_raw$class) # -> 1000 Neg, 1000 Pos
##
##  Neg  Pos
## 1000 1000
```

Se trataba de un dataset totalmente balanceado.

Procesado de los textos:

```
create corpus
movies_corpus <- VCorpus(VectorSource(movies_raw$text))


print(movies_corpus)

## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 2000

# Process the reviews (we will use some functions from the packate tm)
#To lowercase
movies_corpus_clean <- tm_map(movies_corpus, content_transformer(tolower))
#Remove numbers
movies_corpus_clean <- tm_map(movies_corpus_clean, removeNumbers)
#Remove stopwords
movies_corpus_clean <- tm_map(movies_corpus_clean, removeWords, stopwords())
#Remove punctuation signs
movies_corpus_clean <- tm_map(movies_corpus_clean, removePunctuation)


#Carry out the stemming:
movies_corpus_clean <- tm_map(movies_corpus_clean, stemDocument)
#Finally eliminate unneeded whitespace produced by previous steps
movies_corpus_clean <- tm_map(movies_corpus_clean, stripWhitespace)

# Tokenization -> document-term matrix (DTM)
movies_dtm <- DocumentTermMatrix(movies_corpus_clean)


movies_dtm

## <<DocumentTermMatrix (documents: 2000, terms: 24951)>>
## Non-/sparse entries: 501761/49400239
## Sparsity           : 99%
## Maximal term length: 53
## Weighting          : term frequency (tf)
```

Volvemos a obtener 2000 documentos y 24951 términos.

Ahora hay que crear los conjuntos de entrenamiento y de test. Las críticas vienen ordenadas, primero las 1000 críticas positivas, y después las 1000

críticas negativas. Por tanto hay que crear estos dos conjuntos de manera aleatoria.

```
#Set seed to make the process reproducible
set.seed(8)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(movies_raw$class, times=1, p=.75, list=FALSE)

#create training set
movies_dtm_train <- movies_dtm[train_indices, ]
#create testing set
movies_dtm_test  <- movies_dtm[-train_indices, ]

#create labels sets
movies_train_labels <- movies_raw[train_indices, ]$class
movies_test_labels <- movies_raw[-train_indices, ]$class

#view number of rows in each set
nrow(movies_dtm_train)
## [1] 1500
nrow(movies_dtm_test)
## [1] 500
length(movies_train_labels)
## [1] 1500
length(movies_test_labels)
## [1] 500
```

Volvemos a comprobar que los dos conjuntos tienen la misma proporción de críticas positivas y negativas (si no, el entrenamiento no serviría para nada):

```
prop.table(table(movies_train_labels))
## movies_train_labels
## Neg Pos
## 0.5 0.5

prop.table(table(movies_test_labels))
## movies_test_labels
## Neg Pos
## 0.5 0.5
```

Finalizamos la preparación de los datos.
Se necesita obtener un listado con las palabras más utilizadas:

```
# Data preparation – creating indicator features for frequent words
# the function findFreqTerms() in the tm package takes a DTM and returns a character vector
containing words that appear at least a  minimum number of times
movies_freq_words <- findFreqTerms(movies_dtm_train, 100)
#movies_freq_words
```

salida abreviada:
[1] “abil” “abl” “absolut” “accept” “achiev” “across” “act” “action” “actor” “actress” “actual”
[12] “adam” “adapt” “add” “addit” “admit” “adult” “adventur” “age” “agent” “ago” “agre”
…
[991] “whatev” “whether” “white” “whole” “whose” “wife” “wild” “will” “willi” “william”
[reached ‘max’ / getOption(“max.print”) – omitted 27 entries]

Y ahora utilizamos ese listado para limitar el número de columnas/features:

```
ncol(movies_dtm_train)
[1] 24951

movies_dtm_freq_train <- movies_dtm_train[ , movies_freq_words]
movies_dtm_freq_test <- movies_dtm_test[ , movies_freq_words]

ncol(movies_dtm_freq_train)
[1] 1027
```

Finalmente, ya que las matrices DTM tienen valores numéricos, mientras que el algoritmo de clasificación naive bayes necesita operar sobre variables categóricas, se necesita realizar una última tranformación: pasar los valores numéricos a Sí/No:

```
convert_counts <- function(x) {
  factor(ifelse(x > 0, "Yes", "No"))  # IMPORTANTE: usar factor()
}

# Aplicamos la conversi?n y creamos data.frame
movies_train_df <- as.data.frame(apply(movies_dtm_freq_train, MARGIN = 2, convert_counts))
movies_train_df$class <- movies_train_labels

movies_test_df <- as.data.frame(apply(movies_dtm_freq_test, MARGIN = 2, convert_counts))

# Verificamos estructura
str(movies_train_df[1:5])
## 'data.frame':    1500 obs. of  5 variables:
##  $ abil   : chr  "No" "No" "No" "No" ...
##  $ abl    : chr  "No" "No" "No" "No" ...
##  $ absolut: chr  "No" "Yes" "No" "No" ...
##  $ accept : chr  "No" "No" "No" "No" ...
##  $ achiev : chr  "No" "No" "No" "Yes" ...
```

3 Uso de caret para mejorar el modelo

Comprobamos qué parámetros pueden ser ajustados usando caret:

```
modelLookup("naive_bayes")
##         model parameter                label forReg forClass probModel
## 1 naive_bayes   laplace   Laplace Correction  FALSE     TRUE      TRUE
## 2 naive_bayes usekernel    Distribution Type  FALSE     TRUE      TRUE
## 3 naive_bayes    adjust Bandwidth Adjustment  FALSE     TRUE      TRUE
```

Caret ayuda a encontrar los parámetros óptimos de un modelo iterando sobre los valores de una matriz (grid search = búsqueda en cuadrícula), sobre la que se ha definido previamente el método de búsqueda y la métrica de evaluación.

Creamos primeramente el objeto control con los siguientes parámetros:

```
ctrl <- trainControl(method = "cv",
                     number = 10,
                     selectionFunction = "best",
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary   # -> metric ROC in grid

                     )
```

Nota: usando la métrica exactitud, se obtuvo exactamente el mismo resultado. (Al ser un dataset totalmente balanceado, consideré que la exactitud también era válida como métrica).

Se define también la matriz/rejilla de parámetros:

Intentamos que dicha matriz tenga un rango amplio de valores.

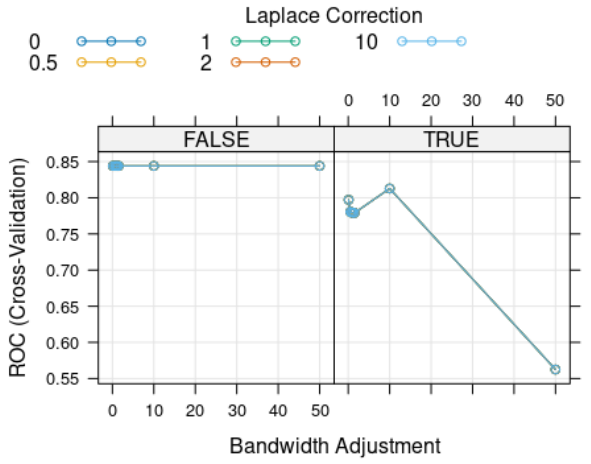
```
grid <- expand.grid(
  laplace = c(0, 0.5, 1.0, 2.0, 10),
  usekernel = c(FALSE, TRUE),
  adjust = c(0.1, 0.5, 0.75, 1.0, 1.25, 1.5, 10, 50)
)
```

Y se procede al entrenamiento:

```
set.seed(12345)

system.time({

  m <- train(class ~ ., data = movies_train_df,
                method = "naive_bayes",
                metric = "ROC",
                trControl = ctrl,
                tuneGrid = grid)

})
##    user  system elapsed
## 656.690   0.092 656.760
print(m)
## Naive Bayes
##
## 1500 samples
## 1026 predictors
##    2 classes: 'Neg', 'Pos'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1350, 1350, 1350, 1350, 1350, 1350, ...
## Resampling results across tuning parameters:
##
##   laplace  usekernel  adjust  ROC        Sens       Spec
##    0.0     FALSE       0.10   0.8440000  0.8133333  0.7133333
##    0.0     FALSE       0.50   0.8440000  0.8133333  0.7133333
##    0.0     FALSE       0.75   0.8440000  0.8133333  0.7133333
##    0.0     FALSE       1.00   0.8440000  0.8133333  0.7133333
##    0.0     FALSE       1.25   0.8440000  0.8133333  0.7133333
##    0.0     FALSE       1.50   0.8440000  0.8133333  0.7133333
##    0.0     FALSE      10.00   0.8440000  0.8133333  0.7133333
##    0.0     FALSE      50.00   0.8440000  0.8133333  0.7133333
##    0.0      TRUE       0.10   0.7972000  1.0000000  0.0000000
##    0.0      TRUE       0.50   0.7806844  1.0000000  0.0000000
##    0.0      TRUE       0.75   0.7796356  1.0000000  0.0000000
##    0.0      TRUE       1.00   0.7796622  1.0000000  0.0000000
##    0.0      TRUE       1.25   0.7790489  1.0000000  0.0000000
##    0.0      TRUE       1.50   0.7790400  1.0000000  0.0000000
##    0.0      TRUE      10.00   0.8127733  1.0000000  0.0000000
##    0.0      TRUE      50.00   0.5624889  1.0000000  0.0000000
##    0.5     FALSE       0.10   0.8440000  0.8133333  0.7133333
##    0.5     FALSE       0.50   0.8440000  0.8133333  0.7133333
##    0.5     FALSE       0.75   0.8440000  0.8133333  0.7133333
...
##    1.0     FALSE       0.50   0.8440000  0.8133333  0.7133333
##    1.0     FALSE       0.75   0.8440000  0.8133333  0.7133333
##    1.0     FALSE       1.00   0.8440000  0.8133333  0.7133333
##    1.0     FALSE       1.25   0.8440000  0.8133333  0.7133333
##    1.0     FALSE       1.50   0.8440000  0.8133333  0.7133333
##    1.0     FALSE      10.00   0.8440000  0.8133333  0.7133333
##    1.0     FALSE      50.00   0.8440000  0.8133333  0.7133333
##    1.0      TRUE       0.10   0.7972000  1.0000000  0.0000000
##    1.0      TRUE       0.50   0.7806844  1.0000000  0.0000000
##    1.0      TRUE       0.75   0.7796356  1.0000000  0.0000000
##    1.0      TRUE       1.00   0.7796622  1.0000000  0.0000000
##    1.0      TRUE       1.25   0.7790489  1.0000000  0.0000000
##    1.0      TRUE       1.50   0.7790400  1.0000000  0.0000000
##    1.0      TRUE      10.00   0.8127733  1.0000000  0.0000000
```

```
##    1.0      TRUE      50.00   0.5624889  1.0000000  0.0000000
##    2.0     FALSE       0.10   0.8440000  0.8133333  0.7133333
##    2.0     FALSE       0.50   0.8440000  0.8133333  0.7133333
##    2.0     FALSE       0.75   0.8440000  0.8133333  0.7133333
##    2.0     FALSE       1.00   0.8440000  0.8133333  0.7133333
...
##    2.0      TRUE      10.00   0.8127733  1.0000000  0.0000000
##    2.0      TRUE      50.00   0.5624889  1.0000000  0.0000000
##   10.0     FALSE       0.10   0.8440000  0.8133333  0.7133333
##   10.0     FALSE       0.50   0.8440000  0.8133333  0.7133333
##   10.0     FALSE       0.75   0.8440000  0.8133333  0.7133333
...
##   10.0      TRUE       1.00   0.7796622  1.0000000  0.0000000
##   10.0      TRUE       1.25   0.7790489  1.0000000  0.0000000
##   10.0      TRUE       1.50   0.7790400  1.0000000  0.0000000
##   10.0      TRUE      10.00   0.8127733  1.0000000  0.0000000
##   10.0      TRUE      50.00   0.5624889  1.0000000  0.0000000
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = FALSE
##  and adjust = 0.1.
##
```

```
plot(m)
```



Se puede apreciar fácilmente en la gráfica que el mejor resultado se da con userkernel=FALSE

```
m$finalModel
```

```
##
## ================================== Naive Bayes ==================================
##
## Call:
## naive_bayes.default(x = x, y = y, laplace = param$laplace, usekernel = FALSE)
##
## ---------------------------------------------------------------------------------
##
## Laplace smoothing: 0
##
## ---------------------------------------------------------------------------------
##
## A priori probabilities:
##
## Neg Pos
## 0.5 0.5
##
## ---------------------------------------------------------------------------------
##
```

```
## Tables:
##
## ------------------------------------------------------------------------------
## :: abilYes (Gaussian)
## ------------------------------------------------------------------------------
##
## abilYes        Neg        Pos
##    mean 0.08133333 0.08400000
##    sd   0.27352875 0.27757293
##
## ------------------------------------------------------------------------------
## :: ablYes (Gaussian)
## ------------------------------------------------------------------------------
##
## ablYes       Neg       Pos
##   mean 0.1240000 0.1733333
##   sd   0.3298015 0.3787878
##
##
...
```

Realizamos la predicción:

```
movies_pred <- predict(m, movies_test_df)
```

Y evaluamos el resultado:

```
#confusion matrix
confusionMatrix(reference = movies_test_labels, data = movies_pred, mode = "everything", positive =
"Pos")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Neg Pos
##        Neg 202  65
##        Pos  48 185
##
##                Accuracy : 0.774
##                  95% CI : (0.7348, 0.8099)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.548
##
##  Mcnemar's Test P-Value : 0.1323
##
##             Sensitivity : 0.7400
##             Specificity : 0.8080
##          Pos Pred Value : 0.7940
##          Neg Pred Value : 0.7566
##               Precision : 0.7940
##                  Recall : 0.7400
##                      F1 : 0.7660
##              Prevalence : 0.5000
##          Detection Rate : 0.3700
##    Detection Prevalence : 0.4660
##       Balanced Accuracy : 0.7740
##
##        'Positive' Class : Pos
##
```

En esta ocasión, se ha obtenido un resultado ligeramente peor que el que se obtuvo en la tarea del tema 4 (ver tabla comparativa en la página siguiente).

Caret es una librería muy útil, porque permite realizar fácilmente la búsqueda de parámetros óptimos para un modelo, pero su uso no garantiza al 100% que

estos parámetros sean encontrados. O dicho de otra manera: no garantiza que esos parámetros produzcan un resultado mejor que el producido por alguna librería más especializada/específica.

	Caret Naive Bayes	Naive Bayes (Tema 4)
Matriz de confusión	Reference Pred. No Yes No **202** 65 Yes 48 **185**	Reference Pred. No Yes No **207** 59 Yes 43 **191**
Exactitud	0.774	0.796
Kappa	0.548	0.592
F1	0.7660	0.7893

2 Ejercicio 2 – Meta-aprendizaje

Usando los diferentes métodos de meta-aprendizaje propuestos en el capítulo 14 (*bagging, boosting, random forests*), elabore modelos con el *dataset* de la prueba de evaluación del tema 5.

1 Paso 1: Carga de los datos

```
#Load data from CRAN package ISLR
#install.packages("ISLR")
library("ISLR")

# Carseats: Sales of Child Car Seats
# Description
#
# A simulated data set containing sales of child car seats at 400 different stores.
```

2 Paso 2: Explorar y preparar los datos

Carga de paquetes necesarios para diversas funciones.

```
# C50 Decision trees C5.0 algorithm
if (!require(C50)) install.packages('C50', dependencies = T)
## Cargando paquete requerido: C50
library(C50)

if (!require(caret)) install.packages('caret', dependencies = T)  # data partitioning, confusion
matrix
library(caret)

if (!require(vcd)) install.packages('vcd', dependencies = T)
library(vcd)

# ipred package offers a classic implementation of bagged decision trees
if (!require(ipred)) install.packages('ipred', dependencies = T)
## Cargando paquete requerido: ipred
library(ipred)

# ada boosting
# https://r-packages.io/packages/adabag/adabag-package
# It implements Freund and Schapire's Adaboost.M1 algorithm and Breiman's Bagging algorithm
#using classification trees as individual classifiers.
if (!require(adabag)) install.packages('adabag', dependencies = T)
library(adabag)

## random forests libraries:  randomForest and ranger
if (!require(randomForest)) install.packages('randomForest', dependencies = T)
library(randomForest)

if (!require(ranger)) install.packages('ranger', dependencies = T)
library(ranger)
```

Preparación de los datos:

La variable dependiente “Sales” es numérica. Para poder predecir si un carrito se venderá o no en función de las variables independientes, debemos transformarla en una variable categórica tipo Sí/No. Consideramos que si las ventas están por encima de la media será un “Sí”, y si no, un “No”:

```
#transform Sales into SalesFactor
sales_mean <- mean(Carseats$Sales)  # mean and median are almost the same
Carseats$SalesFactor <- factor(ifelse(Carseats$Sales>sales_mean,"Yes","No"))
#table(Carseats$SalesFactor)  #check

#Remove Sales variable
CarseatsNew <- Carseats[-1]
```

Conjuntos de entrenamiento y test:

```
#Set seed to make the process reproducible
set.seed(9)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(CarseatsNew$SalesFactor, times=1, p=.75, list=FALSE)

#create training set
CarseatsNew_train <- CarseatsNew[train_indices, ]

#create testing set
CarseatsNew_test  <- CarseatsNew[-train_indices, ]

#create labels sets
CarseatsNew_train_labels <- CarseatsNew[train_indices, ]$SalesFactor
CarseatsNew_test_labels <- CarseatsNew[-train_indices, ]$SalesFactor

#view number of rows in each set
#nrow(CarseatsNew_train)  # 301
#nrow(CarseatsNew_test)   # 99
#length(CarseatsNew_train_labels)  # 301
#length(CarseatsNew_test_labels)   # 99

#Check the proportion in both sets
#prop.table(table(CarseatsNew_train$SalesFactor))
#prop.table(table(CarseatsNew_test$SalesFactor))
```

3 Aplicación de métodos de meta-aprendizaje

Aplicamos los diferentes métodos: bagging, boosting, y random forests.

Bagging

El método **bagging** (también conocido como “boostrap agregating“), consiste en seleccionar de manera aleatoria muestras de un conjunto de entrenamiento (de manera aleatoria, y con reemplazo, es decir, las observaciones pueden aparecer en más de un subconjunto), y entrenar de manera independiente con ellos los “modelos débiles“. Luego, dependiendo de la tarea (regresión o clasificación), se selecciona la media o el valor mayoritario como resultado final.

A) Librería **ipred**

La librería ipred ofrece una implementación clásica de este algoritmo. Será la primera que utilicemos.

Creamos el conjunto de árboles:

```
set.seed(123)

#create ensemble
carseats_bagging <- ipred::bagging(SalesFactor ~ ., data = CarseatsNew_train, nbagg = 25)  # default value of 25 decision trees
#Omitted for brevity
#carseats_bagging$mtrees
```

Y realizamos la predicción

```
bagging_pred <- predict(carseats_bagging, CarseatsNew_test)
```

Evaluamos el resultado:

```
confusionMatrix(reference = CarseatsNew_test_labels, data = bagging_pred, mode = "everything",
positive = "Yes")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  41  13
##        Yes  9  36
##
##                Accuracy : 0.7778
##                  95% CI : (0.6831, 0.8552)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 2.178e-08
##
##                   Kappa : 0.5551
##
##  Mcnemar's Test P-Value : 0.5224
##
##             Sensitivity : 0.7347
##             Specificity : 0.8200
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.7593
##               Precision : 0.8000
##                  Recall : 0.7347
##                      F1 : 0.7660
##              Prevalence : 0.4949
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.4545
##       Balanced Accuracy : 0.7773
##
##        'Positive' Class : Yes
##
```

Es una pequeña mejora con respecto al modelo original. (Se incluye una tabla al final con todos los resultados para facilitar la comparación).

Nota:

Aumentando nbagg a 50 y 100 no se consigue mejorar este resultado.

B) Usando librerías **baguette** y **C50**

```
# baguette
if (!require(baguette)) install.packages('baguette', dependencies = T)
library(baguette)
```

```
set.seed(9)

#define ensemble model
carseats_bagging_C50 <- bag_tree(min_n = 2) %>%
                          set_engine("C5.0") %>%
                          set_mode("classification") %>%
                          translate()
#before training
carseats_bagging_C50
## Bagged Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = 0
##   min_n = 2
##
## Computational engine: C5.0
##
## Model fit template:
## baguette::bagger(x = missing_arg(), y = missing_arg(), weights = missing_arg(),
##     minCases = 2, base_model = "C5.0")

#training the model
carseats_bagging_C50 <- fit(carseats_bagging_C50, SalesFactor ~ ., data = CarseatsNew_train)

#after training
carseats_bagging_C50
## parsnip model object
##
## Bagged C5.0 (classification with 11 members)
##
## Variable importance scores include:
##
## # A tibble: 10 × 4
##    term        value std.error  used
##    <chr>       <dbl>     <dbl> <int>
##  1 Price       96.2      2.18     11
##  2 ShelveLoc   94.7      2.75     11
##  3 CompPrice   68.1      3.92     11
##  4 Advertising 49.2      3.34     11
##  5 Age         39.5      7.32     11
##  6 Income      23.1      5.06     11
##  7 Population   8.12     2.27      9
##  8 Urban        7.73     2.85      7
##  9 Education    6.19     1.36      9
## 10 US           1.75     0.467     4
```

Predicción:

```
bagging_pred_C50 <- predict(carseats_bagging_C50, CarseatsNew_test)
```

Evaluación del resultado:

```
confusionMatrix(reference = CarseatsNew_test_labels, data = bagging_pred_C50$.pred_class, mode =
"everything", positive = "Yes")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  39  12
##        Yes 11  37
##
##                Accuracy : 0.7677
##                  95% CI : (0.6721, 0.8467)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 7.312e-08
##
##                   Kappa : 0.5352
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.7551
##             Specificity : 0.7800
```

```
##          Pos Pred Value : 0.7708
##          Neg Pred Value : 0.7647
##               Precision : 0.7708
##                  Recall : 0.7551
##                      F1 : 0.7629
##              Prevalence : 0.4949
##          Detection Rate : 0.3737
##    Detection Prevalence : 0.4848
##       Balanced Accuracy : 0.7676
##
##        'Positive' Class : Yes
##
```

Resultado muy parecido a los anteriores (exactitud 77.78 y 74.75, kappa 0.555 y 0.494).

C) Usando la librería **ipred** con **caret**

NOTA: como se puede ver en http://topepo.github.io/caret/available-models.html el modelo Bagged CART y modelo "treebag" de la librería ipred no tienen parametros que tunear, por eso no es necesario un objeto "gridsearch".

```
set.seed(123)

ctrl_bagging <- trainControl(method = "cv", number = 10)

carseats_bagging_caret  <-  train(SalesFactor ~ ., data = CarseatsNew_train, method = "treebag",
trControl = ctrl_bagging)

carseats_bagging_caret

## Bagged CART
##
## 301 samples
##  10 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 270, 271, 271, 271, 271, 271, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8408602  0.6820937
```

Con este método se obtiene una exactitud y un valor de Kappa bastante buenos (aparentemente).

Lo comprobamos con el conjunto de test:

```
bagging_pred_caret <- predict(carseats_bagging_caret, CarseatsNew_test)
```

Evaluación del resultado:

```
confusionMatrix(reference = CarseatsNew_test_labels, data = bagging_pred_caret, mode = "everything",
positive = "Yes")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  40  13
##        Yes 10  36
##
```

```
##                Accuracy : 0.7677
##                  95% CI : (0.6721, 0.8467)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 7.312e-08
##
##                   Kappa : 0.535
##
##  Mcnemar's Test P-Value : 0.6767
##
##             Sensitivity : 0.7347
##             Specificity : 0.8000
##          Pos Pred Value : 0.7826
##          Neg Pred Value : 0.7547
##               Precision : 0.7826
##                  Recall : 0.7347
##                      F1 : 0.7579
##              Prevalence : 0.4949
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.4646
##       Balanced Accuracy : 0.7673
##
##        'Positive' Class : Yes
##
```

El resultado vuelve a ser muy parecido a los anteriores.

Boosting

La técnica de boosting se parece a la de bagging en que también se utilizan subconjuntos del conjunto de entrenamiento para entrenar los modelos simples/débiles. La diferencia radica en la manera en la que se realiza el entrenamiento. En el boosting, el entrenamiento no se realiza en paralelo, sino de manera secuencial. En cada iteracción, se aprovecha lo que se ha aprendido en las iteracciones anteriores (se incrementan los pesos de los datos clasificados erróneamente).

Otra diferencia es que bagging se suele utilizar en modelos con varianza alta y bias pequeña, mientras que boosting se suele utilizar con modelos que presentan varianza pequeña y mucha bias.*

* https://www.ibm.com/think/topics/bagging

A) Ada boosting (“adaptative“ boosting):

```
set.seed(123)

system.time({
  adaboost_cv <- boosting.cv(SalesFactor ~ ., data = CarseatsNew) # 10-fold CV
})
## i:  1 Sun Jun 15 21:17:40 2025
## i:  2 Sun Jun 15 21:17:43 2025
## i:  3 Sun Jun 15 21:17:47 2025
## i:  4 Sun Jun 15 21:17:51 2025
## i:  5 Sun Jun 15 21:17:54 2025
## i:  6 Sun Jun 15 21:17:58 2025
## i:  7 Sun Jun 15 21:18:02 2025
## i:  8 Sun Jun 15 21:18:06 2025
## i:  9 Sun Jun 15 21:18:10 2025
## i:  10 Sun Jun 15 21:18:14 2025
##    user  system elapsed
##  37.156   0.251  37.488

adaboost_cv
```

```
## $class
##   [1] "No"  "Yes" "Yes" "Yes" "No"  "Yes" "No"  "Yes" "No"  "No"  "Yes" "Yes"
##  [13] "No"  "Yes" "Yes" "No"  "No"  "Yes" "Yes" "Yes" "No"  "Yes" "No"  "No"
...
## [385] "Yes" "No"  "No"  "Yes" "Yes" "Yes" "No"  "No"  "No"  "Yes" "No"  "Yes"
## [397] "Yes" "No"  "No"  "Yes"
##
## $confusion
##                Observed Class
## Predicted Class  No Yes
##             No  168  37
##             Yes  33 162
##
## $error
## [1] 0.175
```

Evaluación del resultado:

```
Kappa(adaboost_cv$confusion)

##            value     ASE     z  Pr(>|z|)
## Unweighted  0.65 0.03799 17.11 1.315e-65
## Weighted    0.65 0.03799 17.11 1.315e-65
```

Se obtiene un kappa de 0.65, el mejor valor de los obtenidos.

B) Usando **C5.0** con **parsnip**

```
#define model                                  # default values: trees = 15
carseats_boost_C50 <- boost_tree(trees = 25, min_n = 2, sample_size = 0.75) %>%
                      set_engine("C5.0") %>%
                      set_mode("classification") %>%
                      translate()
#before training
carseats_boost_C50

## Boosted Tree Model Specification (classification)
##
## Main Arguments:
##   trees = 25
##   min_n = 2
##   sample_size = 0.75
##
## Computational engine: C5.0
##
## Model fit template:
## parsnip::C5.0_train(x = missing_arg(), y = missing_arg(), weights = missing_arg(),
##     trials = 25, minCases = 2, sample = 0.75)

#training the model
set.seed(123)

system.time({
  carseats_boost_C50 <- fit(carseats_boost_C50, SalesFactor ~ ., data = CarseatsNew_train)
})
##    user  system elapsed
##    0.04    0.00    0.04

#after training
carseats_boost_C50

## parsnip model object
##
##
## Call:
## C5.0.default(x = x, y = y, trials = 25, control = C50::C5.0Control(minCases
##  = 2, sample = 0.75))
##
```

```
## Classification Tree
## Number of samples: 301
## Number of predictors: 10
##
## Number of boosting iterations: 25
## Average tree size: 12.8
##
## Non-standard options: attempt to group attributes, 75% sub-sampling
```

Predicción:

```
boosting_pred_C50 <- predict(carseats_boost_C50, CarseatsNew_test)
```

Evaluación del resultado:

```
confusionMatrix(reference = CarseatsNew_test_labels, data = boosting_pred_C50$.pred_class, mode =
"everything", positive = "Yes")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  44  12
##        Yes  6  37
##
##                Accuracy : 0.8182
##                  95% CI : (0.728, 0.8885)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 9.561e-11
##
##                   Kappa : 0.6359
##
##  Mcnemar's Test P-Value : 0.2386
##
##             Sensitivity : 0.7551
##             Specificity : 0.8800
##          Pos Pred Value : 0.8605
##          Neg Pred Value : 0.7857
##               Precision : 0.8605
##                  Recall : 0.7551
##                      F1 : 0.8043
##              Prevalence : 0.4949
##          Detection Rate : 0.3737
##    Detection Prevalence : 0.4343
##       Balanced Accuracy : 0.8176
##
##        'Positive' Class : Yes
##
```

La exactitud obtenida con 25 árboles es la mejor de todas. Kappa tiene un valor más que aceptable usando estas dos librerías.

C) Adaboost con **caret**

```
set.seed(123)

ctrl_boosting <- trainControl(method = "cv", number = 10)

system.time({
  carseats_boosting_caret  <-  train(SalesFactor ~ ., data = CarseatsNew_train, method =
"AdaBoost.M1", trControl = ctrl_boosting)
})
##    user  system elapsed
## 421.589   2.096 423.705

carseats_boosting_caret
```

```
## AdaBoost.M1
##
## 301 samples
##  10 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 270, 271, 271, 271, 271, 271, ...
## Resampling results across tuning parameters:
##
##   coeflearn  maxdepth  mfinal  Accuracy   Kappa
##   Breiman    1          50     0.8008602  0.6018811
##   Breiman    1         100     0.8340860  0.6680306
##   Breiman    1         150     0.8307527  0.6615523
##   Breiman    2          50     0.8274194  0.6550725
##   Breiman    2         100     0.8473118  0.6947441
##   Breiman    2         150     0.8339785  0.6680775
##   Breiman    3          50     0.8407527  0.6817391
##   Breiman    3         100     0.8375269  0.6754270
##   Breiman    3         150     0.8541935  0.7087603
##   Freund     1          50     0.8404301  0.6808333
##   Freund     1         100     0.8372043  0.6744421
##   Freund     1         150     0.8505376  0.7011088
##   Freund     2          50     0.8374194  0.6745073
##   Freund     2         100     0.8538710  0.7076409
##   Freund     2         150     0.8572043  0.7143076
##   Freund     3          50     0.8174194  0.6348857
##   Freund     3         100     0.8240860  0.6482190
##   Freund     3         150     0.8273118  0.6547441
##   Zhu        1          50     0.8207527  0.6417391
##   Zhu        1         100     0.8374194  0.6750725
##   Zhu        1         150     0.8306452  0.6614108
##   Zhu        2          50     0.8270968  0.6542739
##   Zhu        2         100     0.8539785  0.7082369
##   Zhu        2         150     0.8672043  0.7345756
##   Zhu        3          50     0.8440860  0.6882190
##   Zhu        3         100     0.8307527  0.6615523
##   Zhu        3         150     0.8339785  0.6679167
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mfinal = 150, maxdepth = 2
##  and coeflearn = Zhu.
```

Con Adaboost se ha obtenido el mejor resultado de todos con el dataset de entrenamiento.

NOTA:

El tiempo de ejecución ha sido de 7 minutos, cuando todos los demás han necesitado solo segundos para completarse.

Predicción:

```
boosting_pred_caret <- predict(carseats_boosting_caret, CarseatsNew_test)
```

Evaluación del resultado:

```
confusionMatrix(reference = CarseatsNew_test_labels, data = boosting_pred_caret, mode =
"everything", positive = "Yes")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  40  16
##        Yes 10  33
##
##                Accuracy : 0.7374
```

```
##                  95% CI : (0.6393, 0.8207)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 2.007e-06
##
##                   Kappa : 0.474
##
##  Mcnemar's Test P-Value : 0.3268
##
##             Sensitivity : 0.6735
##             Specificity : 0.8000
##          Pos Pred Value : 0.7674
##          Neg Pred Value : 0.7143
##               Precision : 0.7674
##                  Recall : 0.6735
##                      F1 : 0.7174
##              Prevalence : 0.4949
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.4343
##       Balanced Accuracy : 0.7367
##
##        'Positive' Class : Yes
##
```

No es así a la hora de la verdad. Con adaboost y caret se obtiene el peor resultado con el conjunto de test.

Random Forests

Al igual que en bagging, también se utiliza la técnica de “sampling“ (elaborar de manera aleatoria subconjuntos de datos -con repetición-). La diferencia es que aquí también se eligen de manera aleatoria subconjuntos de las variables de las observaciones. Este factor de aleatoriedad reduce el riesgo de sobreajuste al que son tan propensos los árboles de decisión.

Utilizaremos directamente caret con las dos librerias (randomForest y ranger)

A) Librería randomForest

```
set.seed(123)

ctrl_rf <- trainControl(method = "cv", number = 10)

#grid_rf <- expand.grid(mtry=c(5,10,11,15,20))

system.time({
  carseats_rf_caret  <-  train(SalesFactor ~ ., data = CarseatsNew_train,
                               method = "rf",
                               trControl = ctrl_boosting,
                               #tuneGrid = grid_rf
                               )
})
##    user  system elapsed
##   3.486   0.020   3.506
carseats_rf_caret
##
## Random Forest
##
## 301 samples
##  10 predictor
```

```
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 270, 271, 271, 271, 271, 271, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8306452  0.6614108
##    6    0.8473118  0.6947441
##   11    0.8474194  0.6950725
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 11.
```

Predicción:

```
rf_pred_caret <- predict(carseats_rf_caret, CarseatsNew_test)
```

Evaluación del resultado:

```
confusionMatrix(reference = CarseatsNew_test_labels, data = rf_pred_caret, mode = "everything",
positive = "Yes")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  42  13
##        Yes  8  36
##
##                Accuracy : 0.7879
##                  95% CI : (0.6942, 0.8636)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 6.132e-09
##
##                   Kappa : 0.5753
##
##  Mcnemar's Test P-Value : 0.3827
##
##             Sensitivity : 0.7347
##             Specificity : 0.8400
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.7636
##               Precision : 0.8182
##                  Recall : 0.7347
##                      F1 : 0.7742
##              Prevalence : 0.4949
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.4444
##       Balanced Accuracy : 0.7873
##
##        'Positive' Class : Yes
##
```

Se obtiene una exactitud bastante parecida a las obtenidas hasta ahora. Se puede decir lo mismo del valor de kappa.

B) Librería ranger

Entrenamiento:

```
set.seed(123)

ctrl_ranger <- trainControl(method = "cv", number = 10)
```

```
grid_ranger <- expand.grid(mtry=c(1,2,3,5,10,11),
                           splitrule=c("gini","extratrees"),
                           min.node.size=c(1,2,3,5,10))

system.time({
  carseats_ranger_caret  <-  train(SalesFactor ~ ., data = CarseatsNew_train,
                             method = "ranger",
                             trControl = ctrl_ranger,
                             tuneGrid = grid_ranger)
})
##    user  system elapsed
##  57.864   1.911  33.136


carseats_ranger_caret


## Random Forest
##
## 301 samples
##  10 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 270, 271, 271, 271, 271, 271, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   min.node.size  Accuracy   Kappa
##    1    gini         1             0.7845161  0.5685115
##    1    gini         2             0.7876344  0.5748428
##    1    gini         3             0.7777419  0.5554167
##    1    gini         5             0.7676344  0.5350870
##    1    gini        10             0.7777419  0.5551464
##    1    extratrees   1             0.7277419  0.4545992
##    1    extratrees   2             0.7344086  0.4679325
….
##    3    gini         3             0.8338710  0.6677755
##    3    gini         5             0.8506452  0.7015702
##    3    gini        10             0.8372043  0.6745756
##    3    extratrees   1             0.8075269  0.6147838
##    3    extratrees   2             0.8006452  0.6010879
##    3    extratrees   3             0.8040860  0.6080306
##    3    extratrees   5             0.8107527  0.6213640
##    3    extratrees  10             0.8075269  0.6147838
##    5    gini         1             0.8573118  0.7147441
##    5    gini         2             0.8539785  0.7082369
##    5    gini         3             0.8473118  0.6945833
##    5    gini         5             0.8570968  0.7142739
##    5    gini        10             0.8375269  0.6752144
##    5    extratrees   1             0.8074194  0.6146973
##    5    extratrees   2             0.8140860  0.6280306
##    5    extratrees   3             0.8039785  0.6079167
##    5    extratrees   5             0.8106452  0.6212500
##    5    extratrees  10             0.8106452  0.6212500
##   10    gini         1             0.8539785  0.7080775
##   10    gini         2             0.8539785  0.7082369
...
##   11    gini         3             0.8506452  0.7015702
##   11    gini         5             0.8507527  0.7017391
##   11    gini        10             0.8439785  0.6880775
##   11    extratrees   1             0.8172043  0.6344421
##   11    extratrees   2             0.8139785  0.6279167
##   11    extratrees   3             0.8206452  0.6412500
##   11    extratrees   5             0.8238710  0.6477755
##   11    extratrees  10             0.8338710  0.6677755
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, splitrule = gini
##  and min.node.size = 1.
```

Predicción:

```
ranger_pred_caret <- predict(carseats_ranger_caret, CarseatsNew_test)
```

Evaluación del Resultado:

```
confusionMatrix(reference = CarseatsNew_test_labels, data = ranger_pred_caret, mode = "everything",
positive = "Yes")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  43  15
##        Yes  7  34
##
##                Accuracy : 0.7778
##                  95% CI : (0.6831, 0.8552)
##     No Information Rate : 0.5051
##     P-Value [Acc > NIR] : 2.178e-08
##
##                   Kappa : 0.5548
##
##  Mcnemar's Test P-Value : 0.1356
##
##             Sensitivity : 0.6939
##             Specificity : 0.8600
##          Pos Pred Value : 0.8293
##          Neg Pred Value : 0.7414
##               Precision : 0.8293
##                  Recall : 0.6939
##                      F1 : 0.7556
##              Prevalence : 0.4949
##          Detection Rate : 0.3434
##    Detection Prevalence : 0.4141
##       Balanced Accuracy : 0.7769
##
##        'Positive' Class : Yes
##
```

De nuevo se obtiene con caret un resultado mucho mejor con el conjunto de entrenamiento que con el de test.

Para facilitar la comparación de los métodos utilizados, se muestran en una tabla los resultados de todos ellos.

(Página siguiente)

	Tema 5 - C50	Bagging - ipred	Bagging – C50 + baguette	Bagging – ipred + caret (treebag)
Matriz de confusión	Reference Pred. No Yes No **41** 16 Yes 9 **33**	Reference Pred. No Yes No **41** 13 Yes 9 **36**	Reference Pred. No Yes No **39** 12 Yes 11 **37**	Reference Pred. No Yes No **40** 13 Yes 10 **36**
Exactitud	0.7475	0.7778	0.7677	0.7677
Kappa	0.4942	0.5551	0.5352	0.5350
F1	0.7253	0.7660	0.7629	0.7579
	Tema 5 - C50 Boosting (100 iterations)	Boosting - Adaboosting (1)	Boosting - C50 + parsnip	Boosting – adaboost + caret (Adaboost.M1)
Matriz de confusión	Reference Pred. No Yes No **43** 15 Yes 7 **34**	Reference Pred. No Yes No **168** 37 Yes 33 **162**	Reference Pred. No Yes No **44** 12 Yes 6 **37**	Reference Pred. No Yes No **40** 16 Yes 10 **33**
Exactitud	0.7778		**0.8182**	0.7374
Kappa	0.5548	**0.65**	**0.6359**	0.4740
F1	0.7556		**0.8043**	0.7174
		Random Forest - randomForest	Random Forest - ranger	
Matriz de confusión		Reference Pred. No Yes No **42** 13 Yes 8 **36**	Reference Pred. No Yes No **43** 15 Yes 7 **34**	
Exactitud		0.7879	0.7778	
Kappa		0.5753	0.5548	
F1		0.7742	0.7556	

(1) Resultado al aplicar validación cruzada en el conjunto de entrenamiento

Todos los resultados obtenidos están en el mismo rango, destacando un poco por encima de los demás la combinación de boosting C50 + parsnip.

EVALUACIÓN TEMA 12 - ASPECTOS COMPUTACIONES DEL MACHINE LEARNING

1 Ejercicio 1 - Tratamiento de valores ausentes

Utilizando el dataset airquality, incluido de serie en R, realice una imputación de valores ausentes para las variables Ozone y Solar.R usando la media, la mediana y la moda. Seguidamente obtenga los gráficos de las distribuciones de las variables corregidas, comparándolas con la distribución de los datos originales.

1 Lectura y examen del dataset

Carga de paquetes requeridos:

```
if (!require(tidyverse)) install.packages('tidyverse', dependencies = T)
library(tidyverse)
```

Carga del dataset:

```
airquality_raw <- airquality
```

Examinamos el dataset sin procesar:

```
summary(airquality_raw)
```

```
##      Ozone           Solar.R          Wind             Temp           Month            Day      
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00   Min.   :5.000   Min.   : 1.0  
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00   1st Qu.:6.000   1st Qu.: 8.0  
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00   Median :7.000   Median :16.0  
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88   Mean   :6.993   Mean   :15.8  
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00   3rd Qu.:8.000   3rd Qu.:23.0  
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00   Max.   :9.000   Max.   :31.0  
##  NA's   :37       NA's   :7
```

En el sumario ya se indica que hay 37 valores NAs para el ozono y 7 para la radiación solar.

Busquemos cuáles son esas observaciones con NAs:

```
#rows with NAs in any column
airquality_raw[!complete.cases(airquality_raw), ]
```

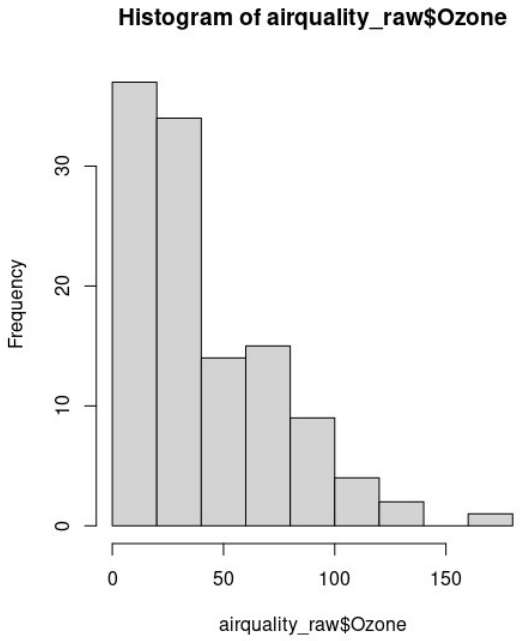
Description: df [42 × 6]

	Ozone <int>	Solar.R <int>	Wind <dbl>	Temp <int>	Month <int>	Day <int>
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6
10	NA	194	8.6	69	5	10
11	7	NA	6.9	74	5	11
25	NA	66	16.6	57	5	25
26	NA	266	14.9	58	5	26
27	NA	NA	8.0	57	5	27
32	NA	286	8.6	78	6	1
33	NA	287	9.7	74	6	2
34	NA	242	16.1	67	6	3

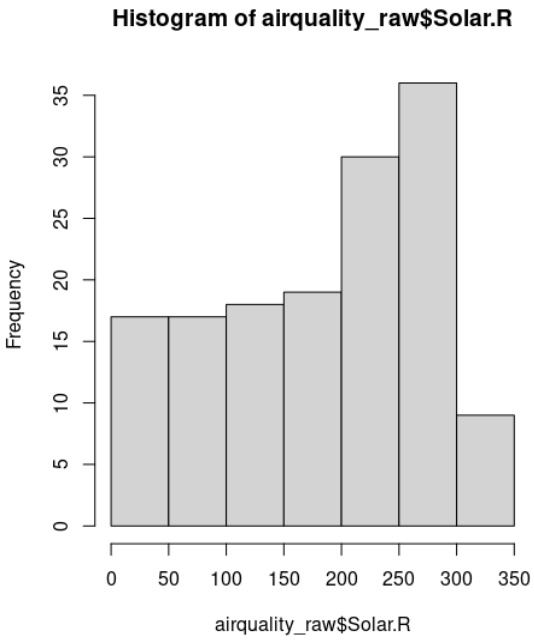
1-10 of 42 rows Previous 1 2 3 4 5 Next

2 Representación gráfica del dataset original

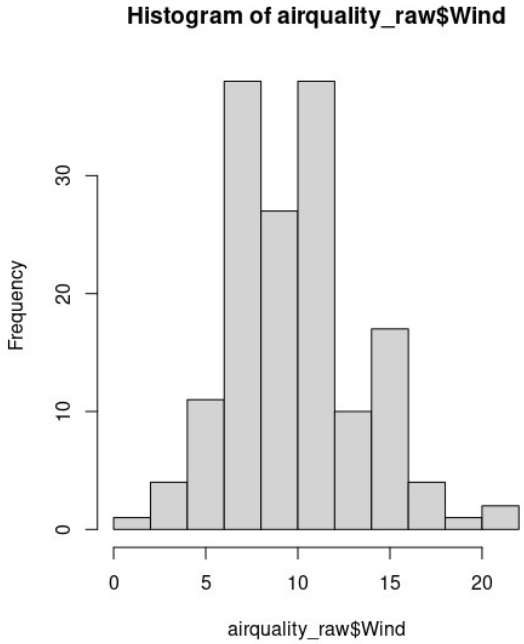
```r
hist(airquality_raw$Ozone)
# breaks = Sturges formula by default to set the number of bins
```



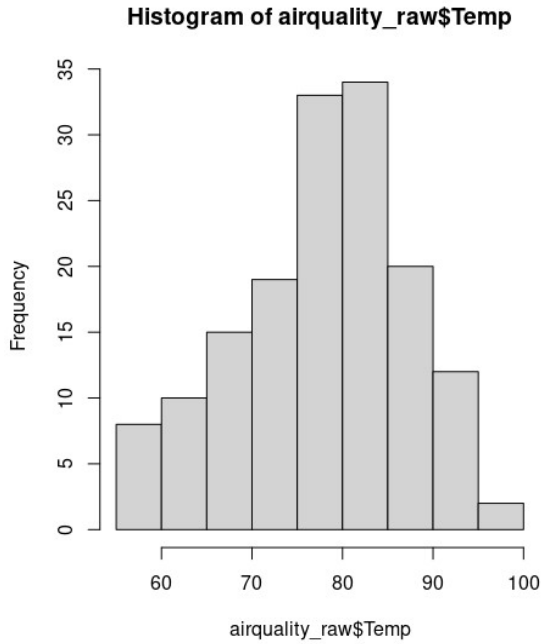

```r
hist(airquality_raw$Solar.R)
```




```r
hist(airquality_raw$Wind)
```




```r
hist(airquality_raw$Temp)
```



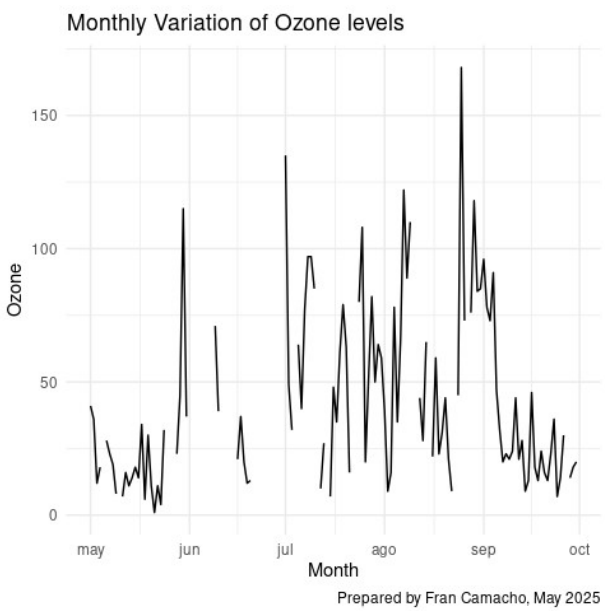
3 Gráficas de las variables usando ggplot2

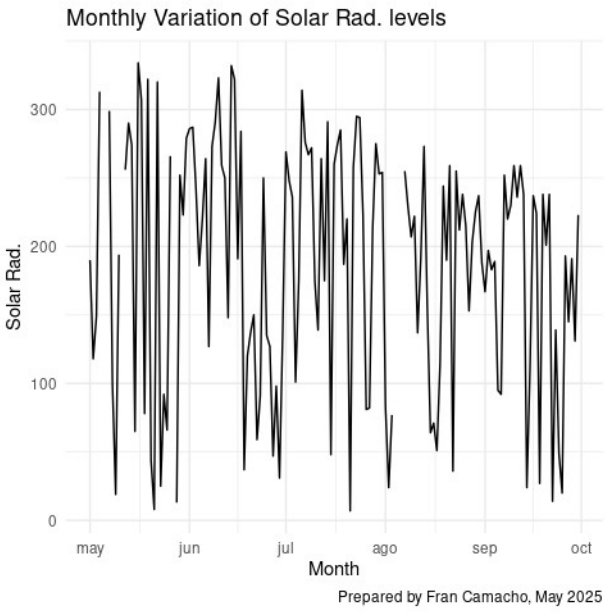
Para aprovechar la capacidad de ggplot2 de dibujar series temporales, convertimos los campos “Month” y “Day” en una fecha. (Definimos el año arbitrariamente como 2025).

```
date_2025 <- as.Date(ISOdate(2025, airquality_raw$Month, airquality_raw$Day))
#date_2025

airquality_raw_with_date <- airquality_raw
airquality_raw_with_date <- airquality_raw_with_date[-c(5,6)]
airquality_raw_with_date$date <- date_2025
```

Mostramos primero las variables para las que faltan datos:

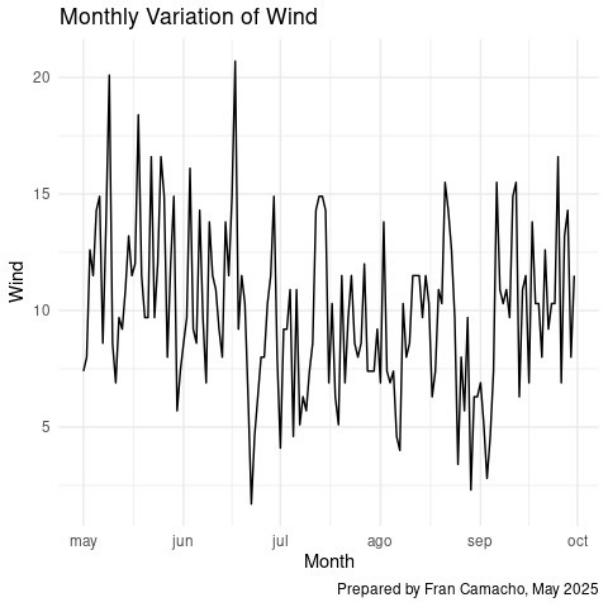
```
# aux. function to plot the variables of the dataframe
plot_airquality_var <- function(x, y, title, ylab) {

  df <- data.frame(x = x, y = y)

  p <- ggplot(df, aes(x,y)) +
       geom_line() +
       labs(title = title, caption ="Prepared by Fran Camacho, May 2025") +
       xlab("Month") +
       ylab(ylab) +
       theme_minimal()
}
ozone_plot <- plot_airquality_var(airquality_raw_with_date$date,
                                  airquality_raw_with_date$Ozone,
                                  "Monthly Variation of Ozone levels",
                                  "Ozone")

solar.r_plot <- plot_airquality_var(airquality_raw_with_date$date,
                                  airquality_raw_with_date$Solar.R,
                                  "Monthly Variation of Solar Rad. levels",
                                  "Solar Rad.")

ozone_plot
solar.r_plot
```

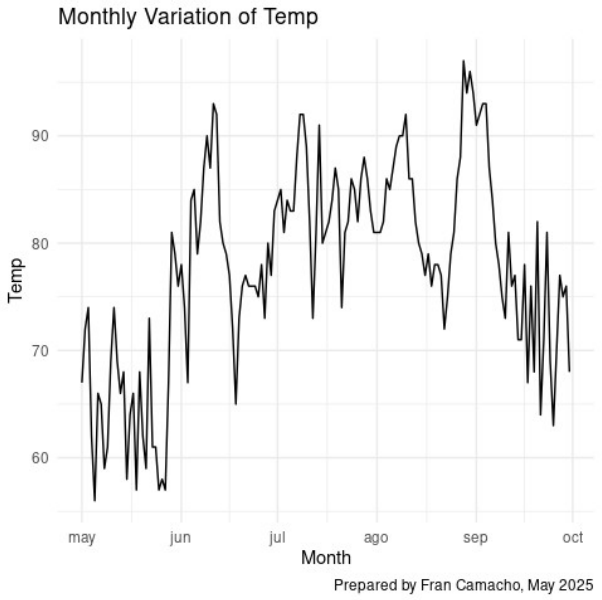




Las variables viento y temperatura están completas:

```
wind_plot <- plot_airquality_var(airquality_raw_with_date$date,
                                 airquality_raw_with_date$Wind,
                                 "Monthly Variation of Wind",
                                 "Wind")

temp_plot <- plot_airquality_var(airquality_raw_with_date$date,
                                 airquality_raw_with_date$Temp,
                                 "Monthly Variation of Temp",
                                 "Temp")

wind_plot
temp_plot
```





4 Completado de las variables

i) Completar variables usando la media

Procesado del dataset:

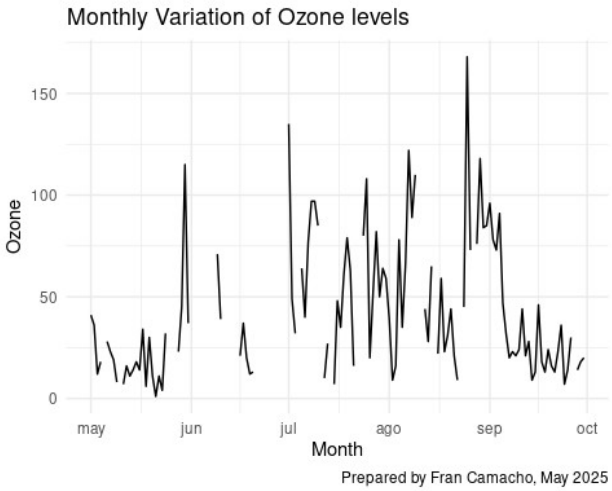
```
airquality_mean_with_date <- airquality_raw |>
            mutate(
                  Ozone_MVI = if_else(is.na(Ozone), 1, 0),
                  Ozone = if_else(is.na(Ozone), mean(Ozone, na.rm = TRUE), Ozone),
                  Solar.R_MVI = if_else(is.na(Solar.R), 1, 0),
                  Solar.R = if_else(is.na(Solar.R), mean(Solar.R, na.rm = TRUE), Solar.R)
            )
```

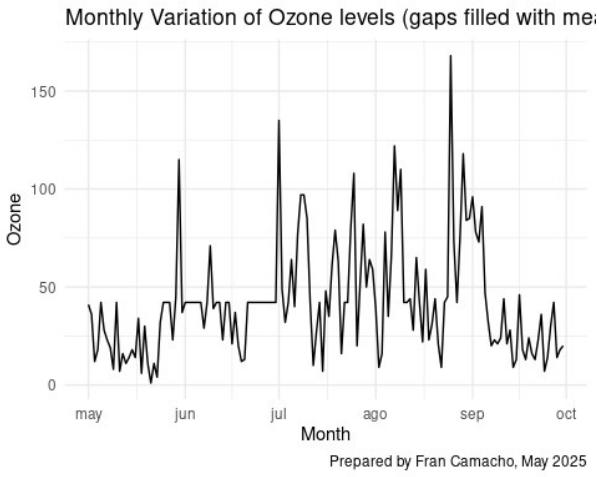
Nota:

He añadido las variables Ozono_MVI y Solar.R_MVI por disciplina. Para acordarme en el futuro de que es necesario añadirlas a la hora de hacer cualquier análisis o aplicar algún algoritmo de machine learning.

Comparación de las gráficas:

```
# add date field
airquality_mean_with_date <- airquality_mean_with_date[-c(5,6)]
airquality_mean_with_date$date <- date_2025

ozone_plot_mean <- plot_airquality_var(airquality_mean_with_date$date,
                                       airquality_mean_with_date$Ozone,
                                       "Monthly Variation of Ozone levels (gaps filled with mean value)",
                                       "Ozone")


# compare graphics
ozone_plot                                              ozone_plot_mean
```





Donde faltaba en las gráfica varios valores seguidos, se pueden apreciar ahora tramos horizontales, ya que ahora se repite el mismo valor para esos días.

ii) Completar variables usando la mediana

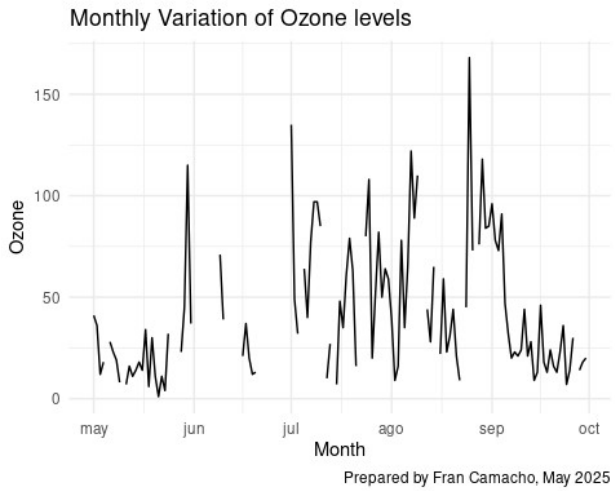
Procesado del dataset:

```
airquality_raw_completed_with_median <- airquality_raw |>
            mutate(
                  Ozone_MVI = if_else(is.na(Ozone), 1, 0),
                  Ozone = if_else(is.na(Ozone), median(Ozone, na.rm = TRUE), Ozone),
                  Solar.R_MVI = if_else(is.na(Solar.R), 1, 0),
                  Solar.R = if_else(is.na(Solar.R), median(Solar.R, na.rm = TRUE), Solar.R)
            )
```

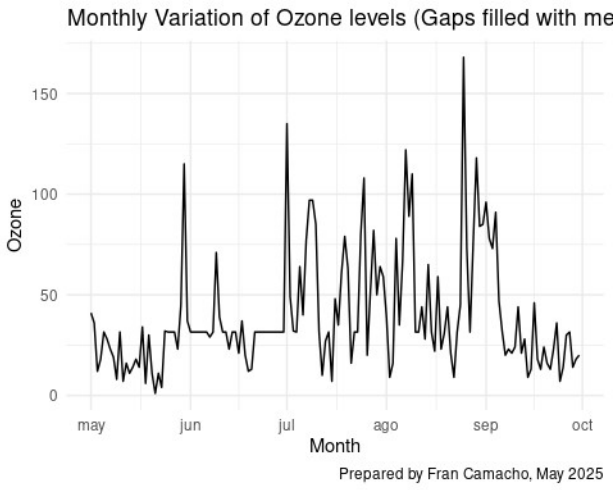
Comparación de las gráficas:

```
# add date field
airquality_median_with_date <- airquality_median_with_date[-c(5,6)]
airquality_median_with_date$date <- date_2025

ozone_plot_median <- plot_airquality_var(airquality_median_with_date$date,
                                       airquality_median_with_date$Ozone,
                                     "Monthly Variation of Ozone levels (Gaps filled with median value)",
                                       "Ozone")
```

```
ozone_plot
```




```
ozone_plot_median
```



La mediana es algo inferior a la media, así que lógicamente los valores completados son menores también que en la gráfica del apartado anterior.

Iii) Completar variables usando la moda

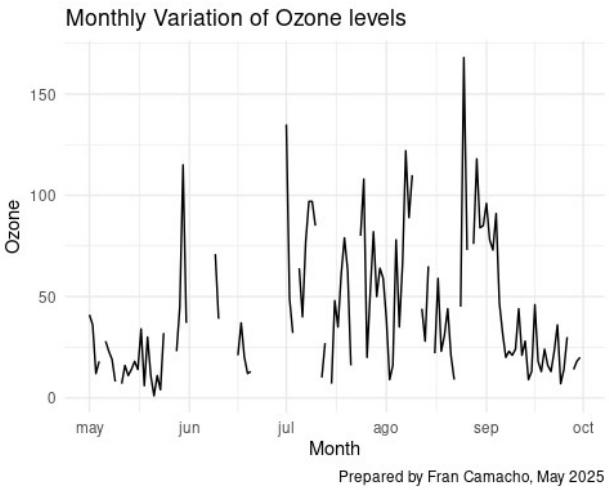
Procesado del dataset:

```
#import one library that can calculates the mode
if (!require(modeest)) install.packages('modeest', dependencies = T)
library(modeest)

ozone_mode <- mfv(airquality_raw$Ozone, na_rm = TRUE)
solar.r_mode <- mfv1(airquality_raw$Solar.R, na_rm = TRUE) # mvf returns 2 values

airquality_mode_with_date <- airquality_raw |>
            mutate(
                  Ozone_MVI = if_else(is.na(Ozone), 1, 0),
                  Ozone = if_else(is.na(Ozone), ozone_mode, Ozone),
                  Solar.R_MVI = if_else(is.na(Solar.R), 1, 0),
                  Solar.R = if_else(is.na(Solar.R), solar.r_mode, Solar.R)
            )
```

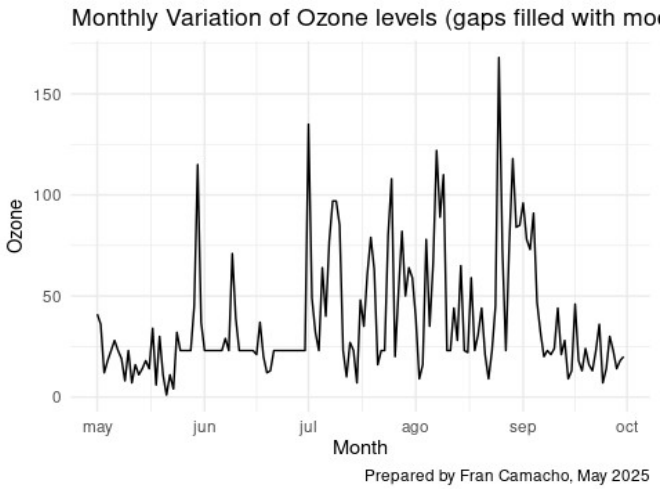
Comparación de las gráficas:

```
# add date field
airquality_mode_with_date <- airquality_mode_with_date[-c(5,6)]
airquality_mode_with_date$date <- date_2025

ozone_plot_mode <- plot_airquality_var(airquality_mode_with_date$date,
                                  airquality_mode_with_date$Ozone,
                                  "Monthly Variation of Ozone levels (gaps filled with mode value)",
                                  "Ozone")
```

```
ozone_plot
```

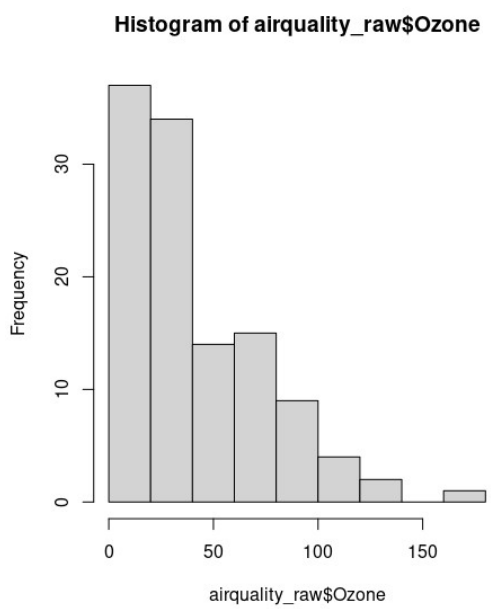



```
ozone_plot_mode
```

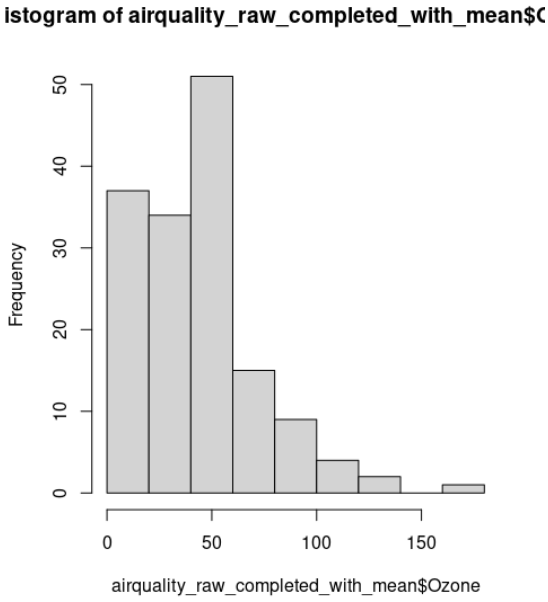


Comparamos también los histogramas:

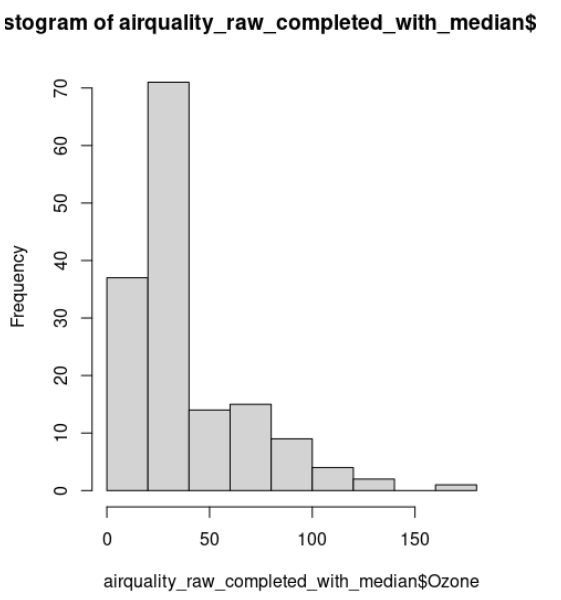
```
#Ozone
hist(airquality_raw$Ozone)
# breaks = Sturges by default to set the number of bins
```



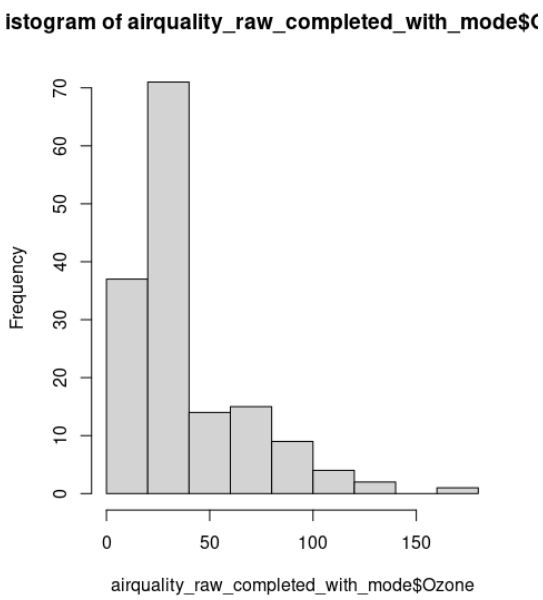

```
hist(airquality_raw_completed_with_mean$Ozone)
```



```
hist(airquality_raw_completed_with_median$Ozone)
```

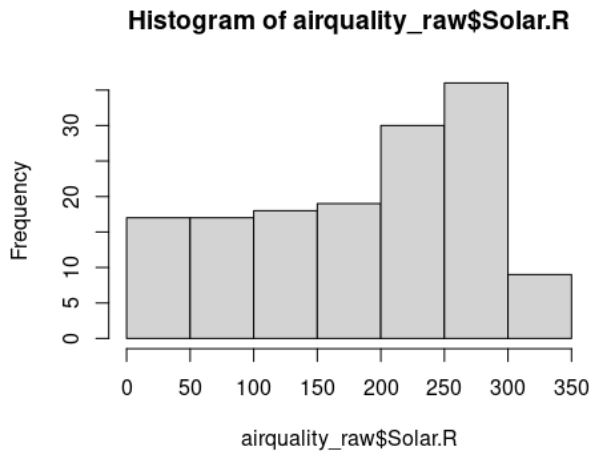



```
hist(airquality_raw_completed_with_mode$Ozone)
```

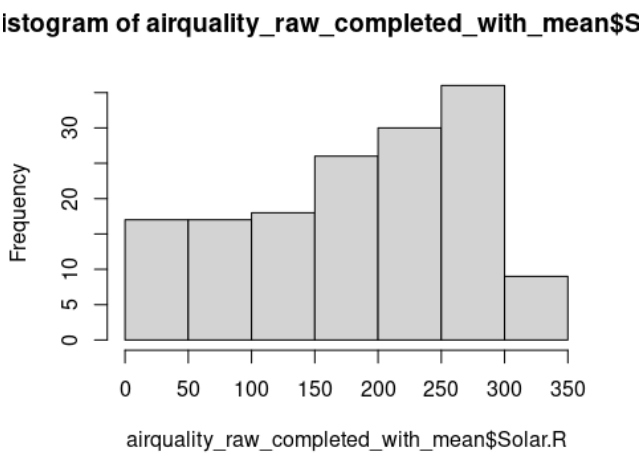


Como el valor de la media del ozono es 42, el histograma refleja de la media refleja un aumento considerable de la columna (“bin”) en la que ese valor está incluido. En el caso de la mediana, es la segunda columna la que refleja que se ha usado este valor para completar los valores que faltaban. (También en el caso de la moda).

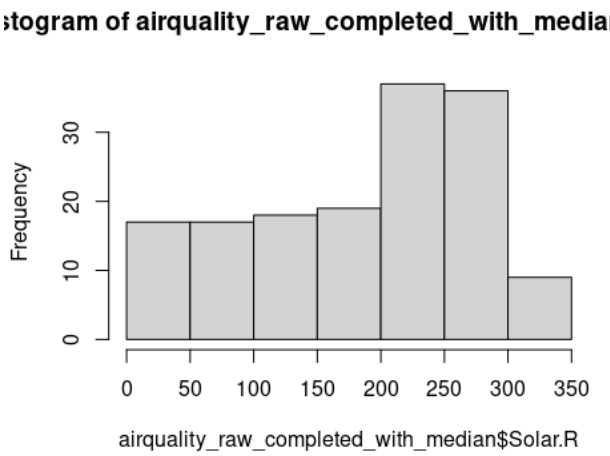
```r
#Solar.R
hist(airquality_raw$Solar.R)
```



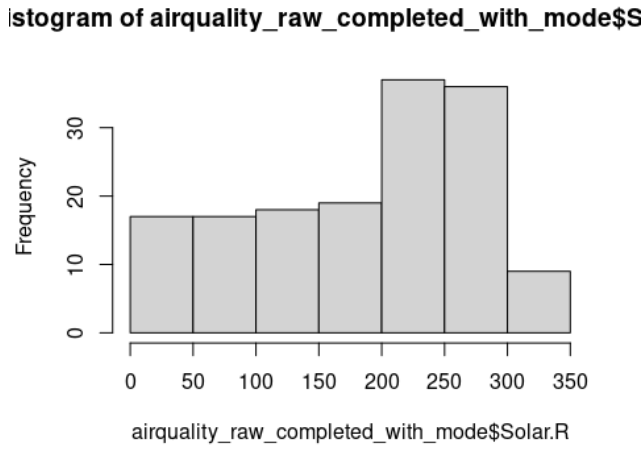

```r
hist(airquality_raw_completed_with_mean$Solar.R)
```




```r
hist(airquality_raw_completed_with_median$Solar.R)
```




```r
hist(airquality_raw_completed_with_mode$Solar.R)
```



En el caso de la radiación solar, las columnas que experimentan un incremento más apreciable son las que están entre los valores 200 y 300 (para la mediana y la moda).

5 Gráfica conjunta del ozono

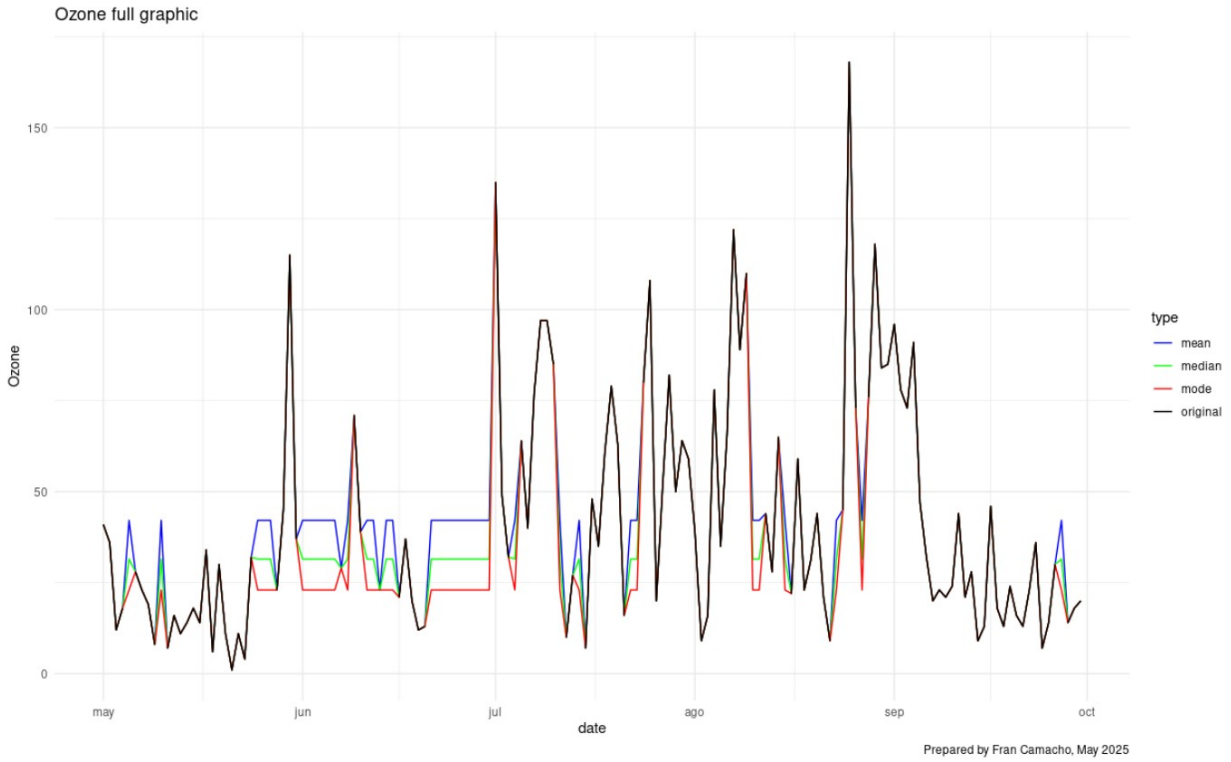
Para practicar un poco con las gráficas en R, he pensado juntar en una misma gráfica los 4 valores del ozono: el original, y los 3 resultantes de completar los valores que faltaban con la media, la mediana y la moda.

Para ello se crea un dataframe con todos los datos, y una columna nueva ("type") que especifica de donde vienen:

```
airquality_mean_with_date <- airquality_mean_with_date[,-c(5,6)]
airquality_mean_with_date$type <- "mean"

airquality_median_with_date <- airquality_median_with_date[,-c(5,6)]
airquality_median_with_date$type <- "median"

airquality_mode_with_date <- airquality_mode_with_date[,-c(5,6)]
airquality_mode_with_date$type <- "mode"

airquality_raw_with_date$type <- "original"


#create dataframe joining the 4 previous dataframes
airquality_data <- rbind(airquality_mean_with_date, airquality_median_with_date,
airquality_mode_with_date, airquality_raw_with_date)
```

Gráfica:

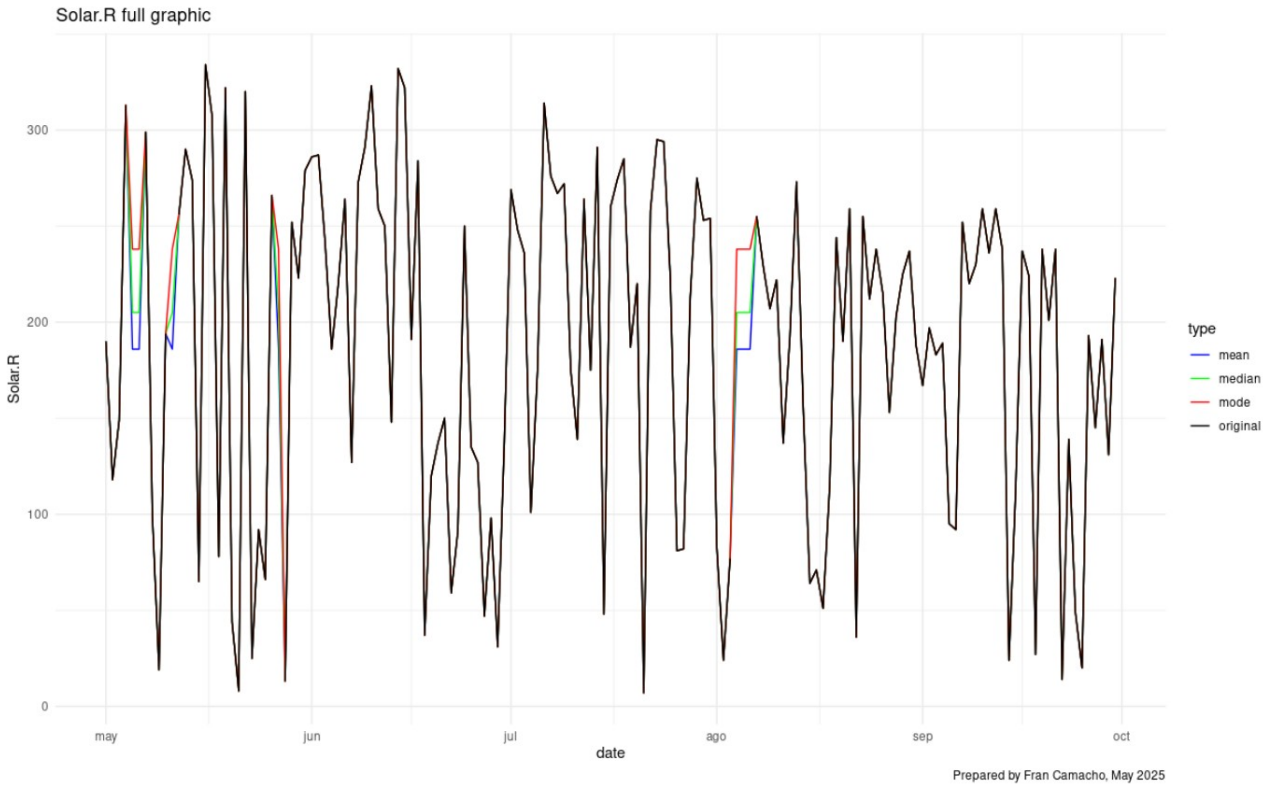
```
ggplot(data = airquality_data, aes(x = date, y = Ozone, colour = type)) +
  geom_line() +
  labs(title = "Ozone full graphic", caption = "Prepared by Fran Camacho, May 2025") +
  scale_colour_manual(values = c("blue","green","red","black")) +
  theme_minimal()
```



6 Gráfica conjunta de la radiación solar

Igualmente para la radiación solar.

```
ggplot(data = airquality_data, aes(x = date, y = Solar.R, colour = type)) +
  geom_line() +
  labs(title = "Solar.R full graphic", caption ="Prepared by Fran Camacho, May 2025") +
  scale_colour_manual(values = c("blue","green","red","black")) +
  theme_minimal()
```



Para la radiación solar, al contrario que para el ozono, la moda y la mediana añaden valores más altos que la media.

2 Ejercicio 2 – Tratamiento de datasets no balanceados con SMOTE

Utilizando el archivo Bank.csv del Tema 7, reequilibre la muestra utilizando SMOTE y repita el ejercicio usando la técnica que desee (redes neuronales o SVM), comparando los resultados obtenidos con los que se obtuvieron previamente.

Utilizaremos SVM, por eso importamos la librería kernlab.

```
#https://www.jstatsoft.org/article/view/v011i09
if (!require(kernlab)) install.packages('kernlab', dependencies = T)
library(kernlab)

if (!require(tidyverse)) install.packages('tidyverse', dependencies = T)
library(tidyverse)

if (!require(caret)) install.packages('caret', dependencies = T)
library(caret)

# themis contains extra steps for the recipes package for dealing with unbalanced data.
if (!require(themis)) install.packages('themis', dependencies = T)
library(themis)
```

1 Carga de los datos

Cargamos mismos datos que en el ejercicio 7.

```
# import the CSV file
bank_raw <- read.csv(file.path("CSVs/Bank", "bank.csv"), sep = ";", stringsAsFactors = TRUE)
```

Se trataba de un dataset con un gran desequilibrio en la variable dependiente:

```
#table(bank_raw$y)
fct_count(bank_raw$y, prop = TRUE)
##   <fct> <int> <dbl>
## 1 no     4000 0.885
## 2 yes     521 0.115
```

2 Aplicación de la técnica SMOTE (Synthetic Minority Oversampling Technique)

Procesamos primero el dataset de igual manera que se hizo en el ejercicio del Tema 7:

```
#scale numeric variables (except day and month)
maxs <- apply(bank_raw[c(1,6,12,13,14,15)], 2, max)
mins <- apply(bank_raw[c(1,6,12,13,14,15)], 2, min)

bank_norm <- data.frame(scale(bank_raw[c(1,6,12,13,14,15)], center = mins, scale = maxs - mins))

#hot encoding of categorical features
dummies <- dummyVars(" ~ job + marital + education + default + housing + loan + contact + poutcome",
data = bank_raw)
bank_hot_encoded_feat <-  data.frame(predict(dummies, newdata = bank_raw))
```

```
#encoding month (name to number)
month_to_number <- function(month_name) {
  month_and_number <- c("jan"=1, "feb"=2, "mar"=3, "apr"=4, "may"=5, "jun"=6, "jul"=7, "aug"=8,
"sep"=9, "oct"=10, "nov"=11, "dec"=12)
  return(month_and_number[as.character(month_name)])
}
bank_raw$month_num <- sapply(bank_raw$month, month_to_number)


#put all features in the same dataframe
bank_processed <-
cbind(bank_norm,as.numeric(bank_raw$day),bank_raw$month_num,bank_hot_encoded_feat,bank_raw$y)
names(bank_processed)[7:8] <- c("day","month")
names(bank_processed)[41] <- "y"
head(bank_processed,5)
```

Description: df [5 × 41]

	age <dbl>	balance <dbl>	duration <dbl>	campaign <dbl>	pdays <dbl>	previous <dbl>	day <dbl>	month <dbl>	job.admin. <dbl>
1	0.1617647	0.06845546	0.02482622	0.00000000	0.0000000	0.00	19	10	0
2	0.2058824	0.10875022	0.07149950	0.00000000	0.3899083	0.16	11	5	0
3	0.2352941	0.06258976	0.05991394	0.00000000	0.3795872	0.04	16	4	0
4	0.1617647	0.06428102	0.06454816	0.06122449	0.0000000	0.00	3	6	0
5	0.5882353	0.04446920	0.07348560	0.00000000	0.0000000	0.00	5	5	0

5 rows | 1-10 of 41 columns

...

Description: df [5 × 41]

loan.no <dbl>	loan.yes <dbl>	contact.cellular <dbl>	contact.telephone <dbl>	contact.unknown <dbl>	poutcome.failure <dbl>	poutcome.other <dbl>	poutcome.success <dbl>	poutcome.unknown <dbl>	y <fctr>
1	0	1	0	0	0	0	0	1	no
0	1	1	0	0	1	0	0	0	no
1	0	1	0	0	1	0	0	0	no
0	1	0	0	0	1	0	0	1	no
1	0	0	0	1	0	0	0	1	no

5 rows | 33-42 of 41 columns

Y ya podemos aplicar SMOTE sobre el dataset

```
#library(themis)
bank_processed_balanced <- bank_processed |> smote("y")  # using y as the feature to balance
```

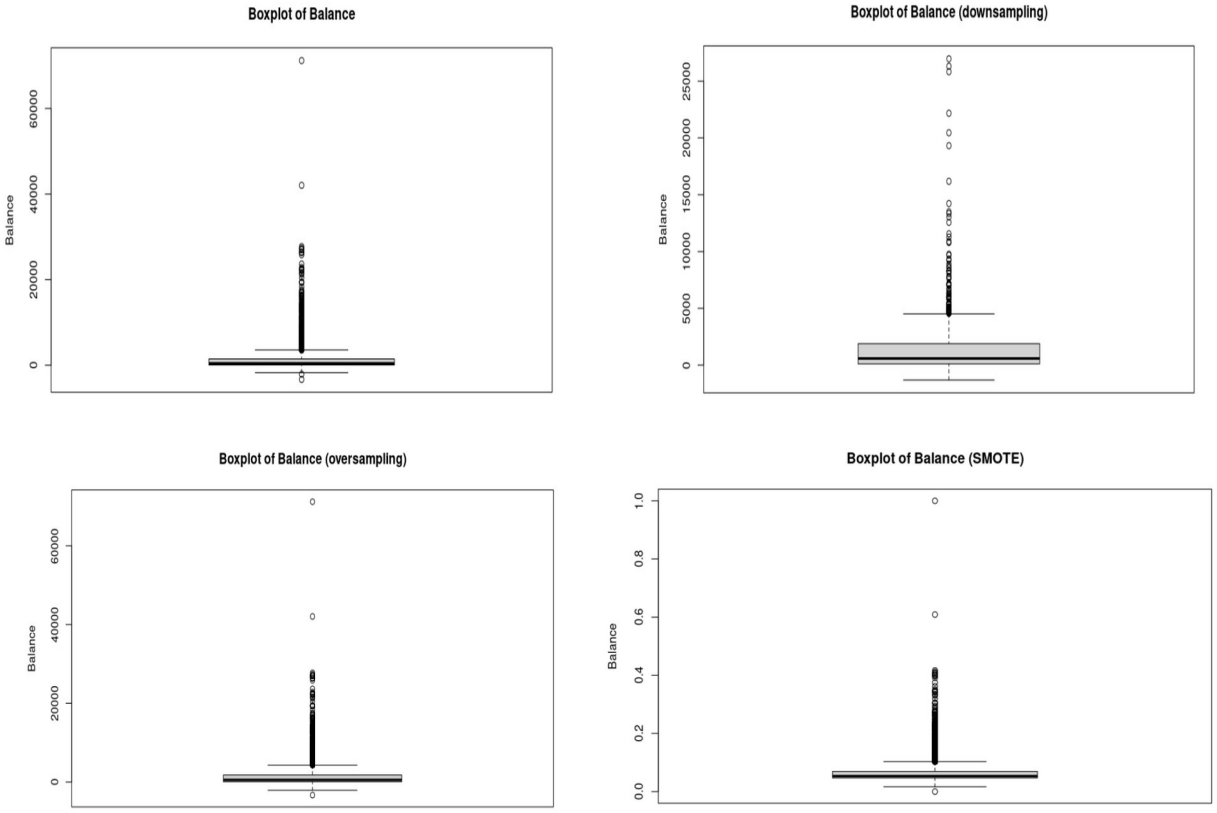
Comprobamos que queda balanceado:

```
fct_count(bank_processed_balanced$y, prop = TRUE)
##   <fct> <int> <dbl>
## 1 no     4000   0.5
## 2 yes    4000   0.5
```

Examinamos el efecto que tiene aplicar SMOTE sobre una variable, comparando los diagramas de caja de la variable balance:

(También apliqué anteriormente las técnicas de downsampling y oversampling, por eso muestro las cuatro gráficas)

```
boxplot(bank_raw$balance, main = "Boxplot of Balance", ylab = "Balance")
boxplot(bank_raw_undersample$balance, main = "Boxplot of Balance (downsampling)", ylab = "Balance")
boxplot(bank_raw_oversample$balance, main = "Boxplot of Balance (oversampling)", ylab = "Balance")
boxplot(bank_processed_balanced$balance, main = "Boxplot of Balance (SMOTE)", ylab = "Balance")
```



Viendo estos diagramas, tengo la intuición/impresión de que oversampling y SMOTE no “afectan“ a la información contenida en el dataset. (Downsampling sí, el diagrama de cajas cambia, es diferente a los otros 3).

3 Aplicación de SVM

Ahora que ya tenemos el dataset balanceado, aplicamos SVM.
Creamos los conjuntos de entrenamiento y test:

```
#Set seed to make the process reproducible
set.seed(9)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(bank_processed_balanced$y, times=1, p=.75, list=FALSE)

#create training set
bank_processed_balanced_train <- bank_processed_balanced[train_indices, ]

#create testing set
bank_processed_balanced_test  <- bank_processed_balanced[-train_indices, ]


#view number of rows in each set
nrow(bank_processed_balanced_train)  # 6000
## [1] 6000
nrow(bank_processed_balanced_test)   # 2000
## [1] 2000
```

Entrenamiento 1 (“automático“, para tener una idea de los hiperparámetros):

```
ctrl_accu <- trainControl(method = "cv", number = 10, selectionFunction = "best",
                          classProbs=TRUE)
set.seed(12345)

system.time({
  m_accu <- train(y ~ ., data = bank_processed_balanced_train, method = "svmRadial",
            metric = "Accuracy",
            trControl = ctrl_accu)
})
##    user  system elapsed
## 111.590   0.556 112.147

cat("Using Accuracy:\n")
## Using Accuracy:
m_accu
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 6000 samples
##   40 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5400, 5400, 5400, 5400, 5400, 5400, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.8811667  0.7623333
##   0.50  0.8961667  0.7923333
##   1.00  0.9091667  0.8183333
##
## Tuning parameter 'sigma' was held constant at a value of 0.01740913
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01740913 and C = 1.
```

Entrenamiento 2:

```
grid <- expand.grid(C = c( 0.9, 1.0, 1.10),
                    sigma = c(0.010, 0.015, 0.01716, 0.020, 0.025))

set.seed(12345)

system.time({
  m_accu <- train(y ~ ., data = bank_processed_balanced_train, method = "svmRadial",
            metric = "Accuracy",
            trControl = ctrl_accu,
            tuneGrid = grid)
})
##    user  system elapsed
## 482.868   1.816 484.726

cat("Using Accuracy:\n")
## Using Accuracy:

m_accu
## Support Vector Machines with Radial Basis Function Kernel
##
## 6000 samples
##   40 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5400, 5400, 5400, 5400, 5400, 5400, ...
## Resampling results across tuning parameters:
##
```

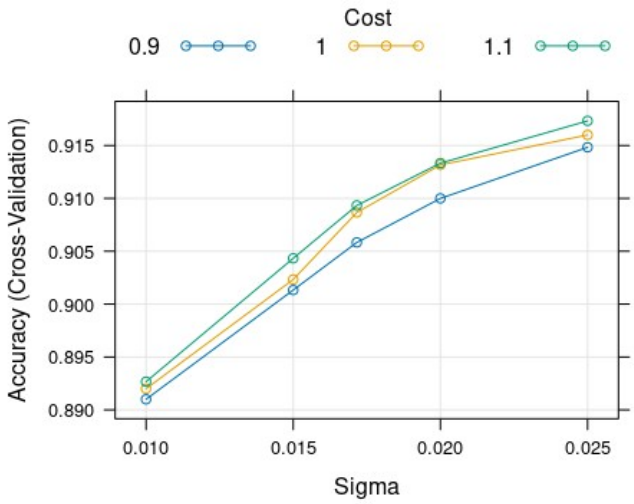
```
##   C    sigma    Accuracy   Kappa
##   0.9  0.01000  0.8910000  0.7820000
##   0.9  0.01500  0.9013333  0.8026667
##   0.9  0.01716  0.9058333  0.8116667
##   0.9  0.02000  0.9100000  0.8200000
##   0.9  0.02500  0.9148333  0.8296667
##   1.0  0.01000  0.8920000  0.7840000
##   1.0  0.01500  0.9023333  0.8046667
##   1.0  0.01716  0.9086667  0.8173333
##   1.0  0.02000  0.9131667  0.8263333
##   1.0  0.02500  0.9160000  0.8320000
##   1.1  0.01000  0.8926667  0.7853333
##   1.1  0.01500  0.9043333  0.8086667
##   1.1  0.01716  0.9093333  0.8186667
##   1.1  0.02000  0.9133333  0.8266667
##   1.1  0.02500  0.9173333  0.8346667
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.025 and C = 1.1.
```

Parece que por fin se ha conseguido pasar del 90% de precisión (con un Kappa más que bueno: 0.8347)

```
m_accu$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1.1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.025
##
## Number of Support Vectors : 2229
##
## Objective Function Value : -1681.196
## Training error : 0.059
## Probability model included.
```

```
plot(m_accu)
```



Al ver la gráfica anterior, he pensado hacer más intentos por mejorar el resultado, ya que he visto que al aumentar sigma, se obtienen mejores valores de exactitud y de kappa:

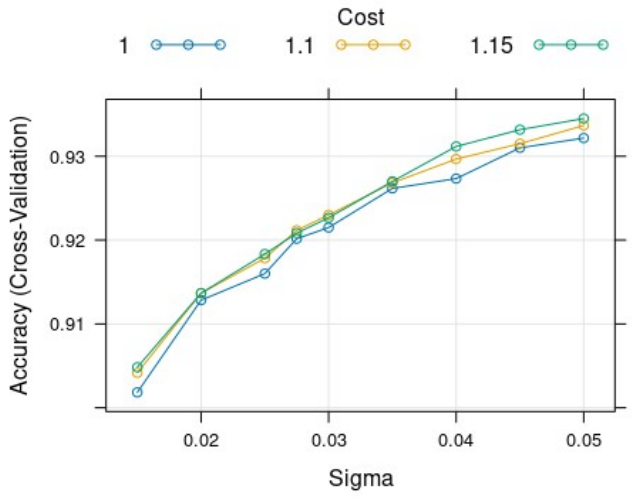
```
grid_2 <- expand.grid(C = c(1.0, 1.10, 1.15),
                      sigma = c(0.015, 0.020, 0.025, 0.0275, 0.030, 0.035, 0.040, 0.045, 0.050))

set.seed(12345)

system.time({
  m_accu_2 <- train(y ~ ., data = bank_processed_balanced_train, method = "svmRadial",
            metric = "Accuracy",
            trControl = ctrl_accu,
            tuneGrid = grid_2)
})
```

```
##    user  system elapsed
## 920.980   4.6
```

```
m_accu_2
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 6000 samples
##   40 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5400, 5400, 5400, 5400, 5400, 5400, ...
## Resampling results across tuning parameters:
##
##   C     sigma   Accuracy   Kappa
##   1.00  0.0150  0.9018333  0.8036667
##   1.00  0.0200  0.9128333  0.8256667
##   1.00  0.0250  0.9160000  0.8320000
...
##   1.15  0.0400  0.9311667  0.8623333
##   1.15  0.0450  0.9331667  0.8663333
##   1.15  0.0500  0.9345000  0.8690000
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.05 and C = 1.15.
```

```
plot(m_accu_2)
```

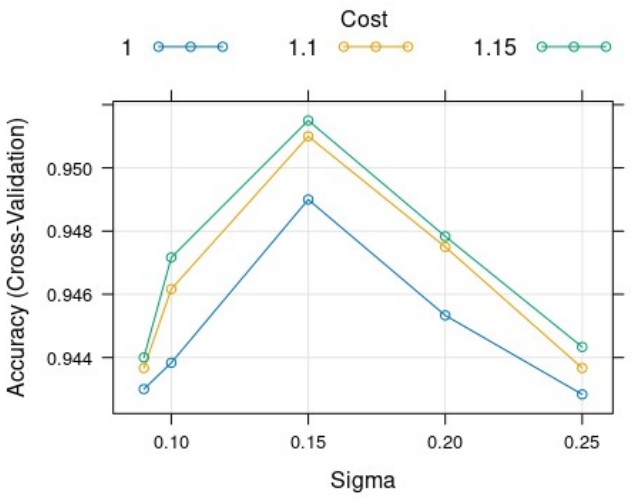


[Por brevedad se omiten el tercer y el cuarto intento, y se muestra directamente el último]

```
set.seed(12345)

system.time({
  m_accu_4 <- train(y ~ ., data = bank_processed_balanced_train, method = "svmRadial",
            metric = "Accuracy",
            trControl = ctrl_accu,
            tuneGrid = grid_4)
})
##     user   system  elapsed
## 1056.451    2.956 1060.291
m_accu_4
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 6000 samples
##   40 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5400, 5400, 5400, 5400, 5400, 5400, ...
## Resampling results across tuning parameters:
##
##   C     sigma  Accuracy   Kappa
##   1.00  0.09   0.9430000  0.8860000
##   1.00  0.10   0.9438333  0.8876667
##   1.00  0.15   0.9490000  0.8980000
##   1.00  0.20   0.9453333  0.8906667
##   1.00  0.25   0.9428333  0.8856667
##   1.10  0.09   0.9436667  0.8873333
##   1.10  0.10   0.9461667  0.8923333
##   1.10  0.15   0.9510000  0.9020000
##   1.10  0.20   0.9475000  0.8950000
##   1.10  0.25   0.9436667  0.8873333
##   1.15  0.09   0.9440000  0.8880000
##   1.15  0.10   0.9471667  0.8943333
##   1.15  0.15   0.9515000  0.9030000
##   1.15  0.20   0.9478333  0.8956667
##   1.15  0.25   0.9443333  0.8886667
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.15 and C = 1.15.
```

```
plot(m_accu_4)
```



Hemos obtenido un modelo, que con los hiperparámetros C=1.15 y sigma=0.15 da una exactitud del 95% con un Kappa de 0.9 (a primera vista, impresionante).

Vamos a evaluar finalmente el modelo resultante con el conjunto de test que creamos anteriormente:

```
m_accu_pred <- predict(m_accu_4, bank_processed_balanced_test[,1:40])
```

Matriz de confusión:

```
confusionMatrix(ref = as.factor(bank_processed_balanced_test$y), data = as.factor(m_accu_pred),
positive="yes", mode = "everything")

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  970  52
##        yes  30 948
##
##                Accuracy : 0.959
##                  95% CI : (0.9494, 0.9673)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.918
##
##  Mcnemar's Test P-Value : 0.02039
##
##             Sensitivity : 0.9480
##             Specificity : 0.9700
##          Pos Pred Value : 0.9693
##          Neg Pred Value : 0.9491
##               Precision : 0.9693
##                  Recall : 0.9480
##                      F1 : 0.9585
##              Prevalence : 0.5000
##          Detection Rate : 0.4740
##    Detection Prevalence : 0.4890
##       Balanced Accuracy : 0.9590
##
##        'Positive' Class : yes
##
```

Se ha obtenido una exactitud del 96.4!

El valor de Kappa es casi 1 y los de precisión, recall, F1 ... pasan del 95%.

El resultado creo sinceramente que es demasiado bueno para ser verdad. Así que voy a repetir el proceso entero, pero **separando el conjunto de test antes de aplicar SMOTE**. De esta manera veremos si este resultado tan bueno es real, o es solo debido a que se ha producido un sobreajuste a los datos de entrada (estamos creando con esta técnica datos “parecidos“ a los que ya tenemos).

4 Aplicación de SMOTE solo al conjunto de entrenamiento

Por brevedad, solo se se muestran los pasos que son diferentes:

[Procesado del dataset]

Creación de conjuntos de entramiento (y validación cruzada) y de test.

```
#Set seed to make the process reproducible
set.seed(9)

#partitioning data frame into training (75%) and testing (25%) sets
train_indices <- createDataPartition(bank_processed$y, times=1, p=.75, list=FALSE)

#create training set
bank_processed_train <- bank_processed[train_indices, ]

#create testing set
bank_processed_test  <- bank_processed[-train_indices, ]


#view number of rows in each set
nrow(bank_processed_train)  # 3391
## [1] 3391
nrow(bank_processed_test)   # 1130
## [1] 1130
```

Las proporciones se mantienen evidentemente:

```
fct_count(bank_processed_train$y, prop = TRUE)
##   <fct> <int> <dbl>
## 1 no     3000 0.885
## 2 yes     391 0.115

fct_count(bank_processed_test$y, prop = TRUE)
```

```
##   <fct> <int> <dbl>
## 1 no     1000 0.885
## 2 yes     130 0.115
```

Ahora se aplica la función smote() de la librería themis solo al conjunto de entrenamiento:

```
#library(themis)
bank_processed_train_balanced <- bank_processed_train |> smote("y")  # using y as the feature to
balance

fct_count(bank_processed_train_balanced$y, prop = TRUE)
##   <fct> <int> <dbl>
## 1 no     3000   0.5
## 2 yes    3000   0.5
```

Y lógicamente las proporciones se igualan en ese conjunto.

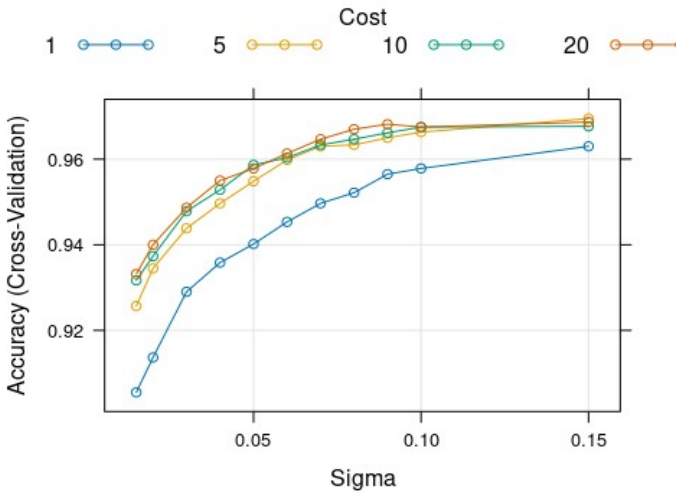
Ahora que ya tenemos el dataset de entrenamiento balanceado, aplicamos SVM:

[Por brevedad se muestra directamente el último entrenamiento]

Entrenamiento 2:

```
grid_2 <- expand.grid(C = c(1, 5, 10, 20),
                      sigma = c(0.015, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.15))
```

```
set.seed(12345)

system.time({
  m_accu_2 <- train(y ~ ., data = bank_processed_train_balanced, method = "svmRadial",
            metric = "Accuracy",
            trControl = ctrl_accu, #same control object with 10-fold cross validation
            tuneGrid = grid_2)
})
##     user   system  elapsed
## 1416.913    2.916 1420.250

m_accu_2
## Support Vector Machines with Radial Basis Function Kernel
##
## 6000 samples
##   40 predictor
##    2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5400, 5400, 5400, 5400, 5400, 5400, ...
## Resampling results across tuning parameters:
##
##   C   sigma  Accuracy   Kappa
##    1  0.015  0.9055000  0.8110000
...
##    5  0.100  0.9663333  0.9326667
##    5  0.150  0.9695000  0.9390000
##   10  0.015  0.9316667  0.8633333
...
##   20  0.100  0.9675000  0.9350000
##   20  0.150  0.9686667  0.9373333
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.15 and C = 5.

plot(m_accu_2)
```



Se vuelve a obtener un valor increíblemente bueno con el conjunto de entrenamiento (exactitud casi al 97%, y kappa muy cerca del 1).

Finalmente, realizamos la predicción y la evaluación usando el dataset que habíamos reservado para test.

Hay que recordar que este dataset no está balanceado, lo reservamos antes de aplicar SMOTE. La idea es simular con este dataset datos nuevos/reales:

```
fct_count(bank_processed_test$y, prop = TRUE)
##   <fct> <int> <dbl>
## 1 no     1000 0.885
## 2 yes     130 0.115
```

Predicción:

```
m_accu_pred <- predict(m_accu_2, bank_processed_test[,1:40])
```

Matriz de confusión:

```
confusionMatrix(ref = as.factor(bank_processed_test$y), data = as.factor(m_accu_pred),
positive="yes", mode = "everything")
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  966 100
##        yes  34  30
##
##                Accuracy : 0.8814
##                  95% CI : (0.8611, 0.8997)
##     No Information Rate : 0.885
##     P-Value [Acc > NIR] : 0.6662
##
##                   Kappa : 0.2525
##
##  Mcnemar's Test P-Value : 1.964e-08
##
##             Sensitivity : 0.23077
##             Specificity : 0.96600
##          Pos Pred Value : 0.46875
##          Neg Pred Value : 0.90619
##               Precision : 0.46875
##                  Recall : 0.23077
##                      F1 : 0.30928
##              Prevalence : 0.11504
##          Detection Rate : 0.02655
##    Detection Prevalence : 0.05664
##       Balanced Accuracy : 0.59838
##
##        'Positive' Class : yes
##
```

5 Conclusión final

Aplicando SMOTE solo a los conjuntos de entrenamiento y validación cruzada, y no al de test, el resultado final es igual que si no se aplica SMOTE.

Deduzco que esta técnica no siempre aporta valor. Incluso he encontrado referencias de expertos que aconsejan no utilizarla (aconsejan no manipular los datos disponibles).

TRABAJO FIN DE MÁSTER

El trabajo fin de Máster consistirá en un trabajo de temática libre en el que debe aplicarse alguna de las técnicas de Machine Learning aprendidas en el curso a una serie de datos o a un conjunto de datos. Previamente al inicio del trabajo, el alumno deberá proponer al Director del Máster el tema del trabajo y un breve resumen del mismo enviando un email a amunoz@cee.uned.es

El trabajo deberá estar estructurado en tres apartados: primero el alumno debe plantear una hipótesis que deberá contrastar usando la base de datos propuesta. Seguidamente deberá indicar qué técnica utilizará y explicar por qué es apropiada para contrastar la hipótesis planteada. Tras desarrollar los cálculos del modelo y obtener resultados, el alumno deberá presentar las conclusiones extraídas a partir del análisis realizado, indicando si se verifica la hipótesis inicialmente establecida o no, y por qué. Se valorará positivamente la inclusión de gráficos y tablas que ayuden a comprender mejor el análisis realizado.

Se recomienda al alumno que, en caso de no disponer de una base de datos sobre la temática que desee investigar, acuda a repositorios como UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets.php) o Kaggle (https://www.kaggle.com/datasets), donde podrá encontrar multitud de *datasets* listos para usar.

Por último, se recuerda al alumno deberá presentar todo el código utilizado para desarrollar el análisis en forma de anexo al final del trabajo.

ANEXO

Enlaces a los repositorios en GitHub del código R y Python de los ejercicios.

GitHub R:

https://github.com/fcamadi/masterML/blob/main/Tema2_Ejercicio_01.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema2_Ejercicio_02.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema3_Ejercicio.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema4_Ejercicio.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema5_Ejercicio.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema5_Ejercicio_Weka_version.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema6_Ejercicio.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema7_Ejercicio_redes_neuronas.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema7_Ejercicio_SVM.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema8_Ejercicio_reglas_asociacion.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema8_Ejercicio_reglas_asociacion_ECLAT.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema9_Ejercicio_kmeans.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema10_Ejercicio_model_eval_svm.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema11_Ejercicio1_Mejora_Modelo_NB_con_caret.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema11_Ejercicio2_Mejora_Modelo_ensemble.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema12_Ejercicio_1_missing_values.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema12_Ejercicio_2_SMOTE.Rmd

https://github.com/fcamadi/masterML/blob/main/Tema12_Ejercicio_2b_SMOTE.Rmd

GitHub Python:

https://github.com/fcamadi/masterMLpython/blob/main/Tema03/Tema3_Ejercicio.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema04/Tema4_NaiveBayes.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema05/Tema5_DecisionTrees.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema06/Tema6_linear_regression.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema07/Tema7_neural_networks.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema07/Tema7_SVM.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema08/Tema8_assoc_rules.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema09/Tema9_kmeans.ipynb

https://github.com/fcamadi/masterMLpython/blob/main/Tema10/Tema10_model_eval_SVM.ipynb