Tema7_Ejercicio_redes_neuronas

Fran Camacho

2025-02-24

Tema 7 - Ejercicio

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.

Paso 1: Carga de los datos

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

Paso 2: Explorar y preparar los datos

Carga de paquetes que son necesarios para diversas funciones.

```r
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)
}

## [1] "Choosing Keras"

## Loading required package: keras3

## Loading required package: tidyverse

## ── Attaching core tidyverse packages ────────────────────────
tidyverse 2.0.0 ──
##    dplyr     1.1.4        readr     2.1.5
##    forcats   1.0.0        stringr   1.5.1
##    ggplot2   3.5.1        tibble    3.2.1
##    lubridate 1.9.4        tidyr     1.3.1
##    purrr     1.0.4
## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
##    dplyr::filter() masks stats::filter()
##    dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to
force all conflicts to become errors
```

```
## Loading required package: jsonlite
##
##
## Attaching package: 'jsonlite'
##
##
## The following object is masked from 'package:purrr':
##
##     flatten
```

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

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
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
```

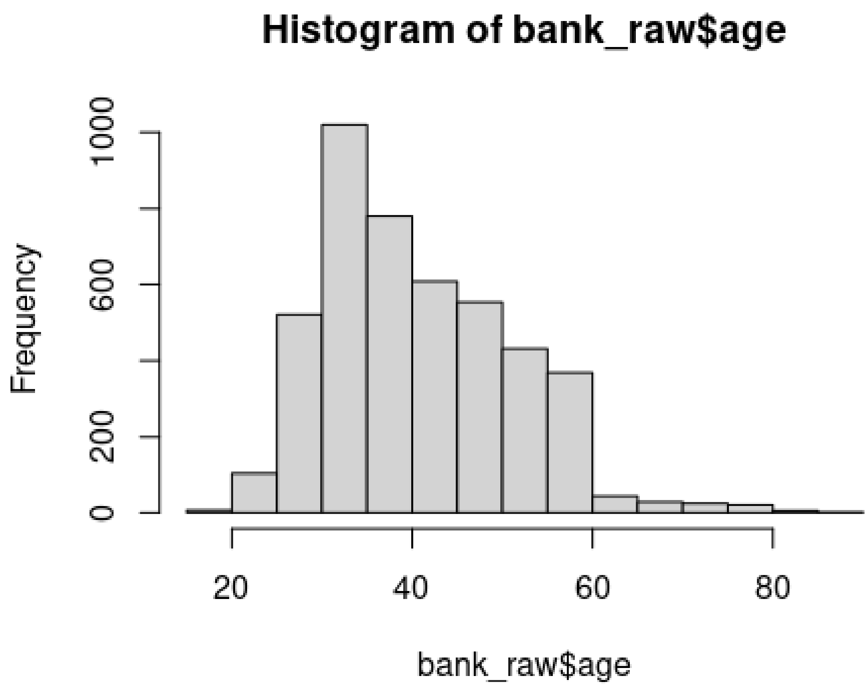
```
##  3rd Qu.: 1480                                          3rd
Qu.:21.00
##  Max.   :71188
Max.   :31.00
##

##      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)
```

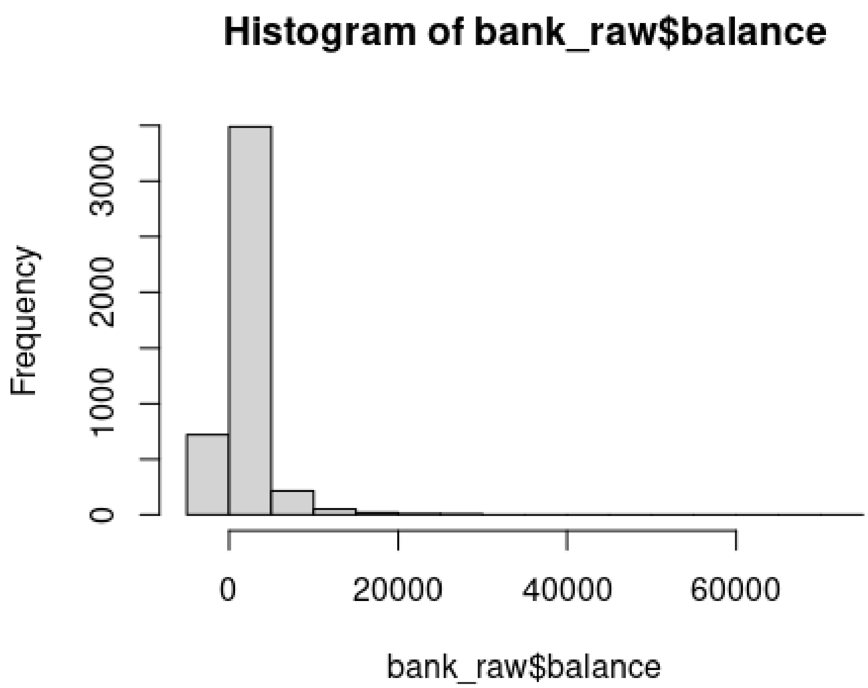
```
##   age         job marital education default balance housing loan
contact day
## 1  30  unemployed married   primary      no    1787      no   no
cellular  19
## 2  33    services married secondary      no    4789     yes  yes
cellular  11
## 3  35  management  single  tertiary      no    1350     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
```

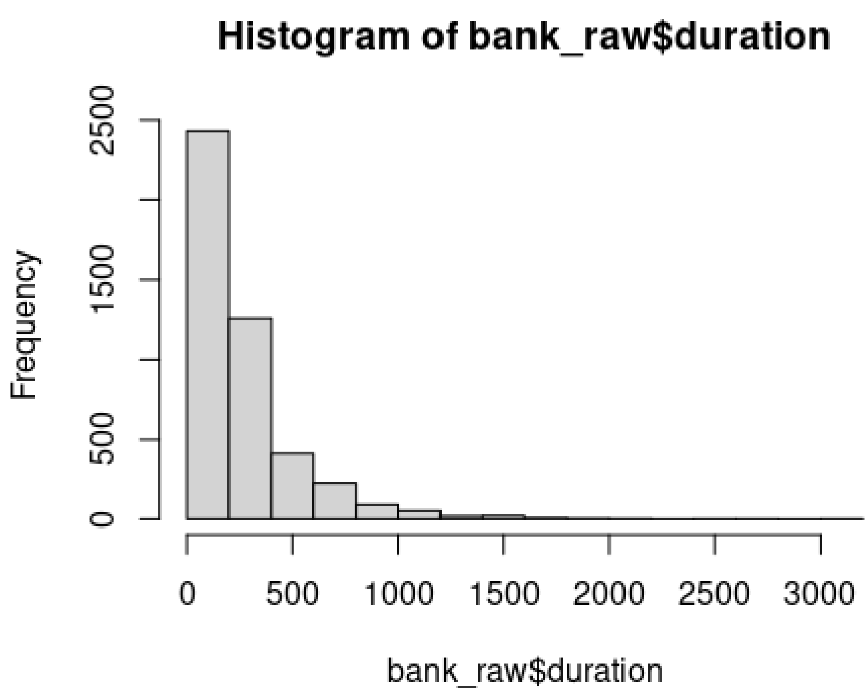
```r
#Summary
hist(bank_raw$age)
```



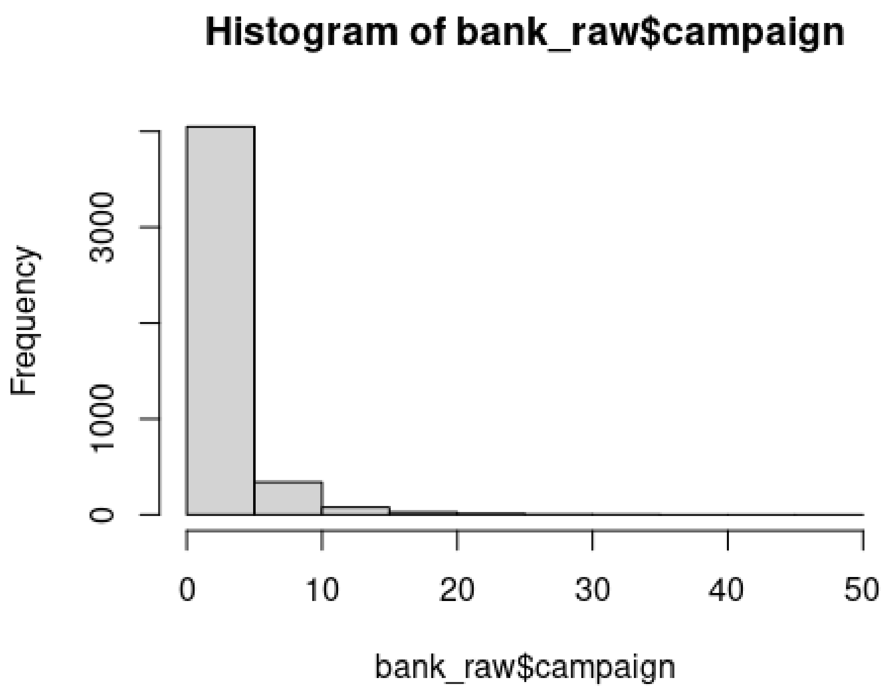

```r
hist(bank_raw$balance)
```



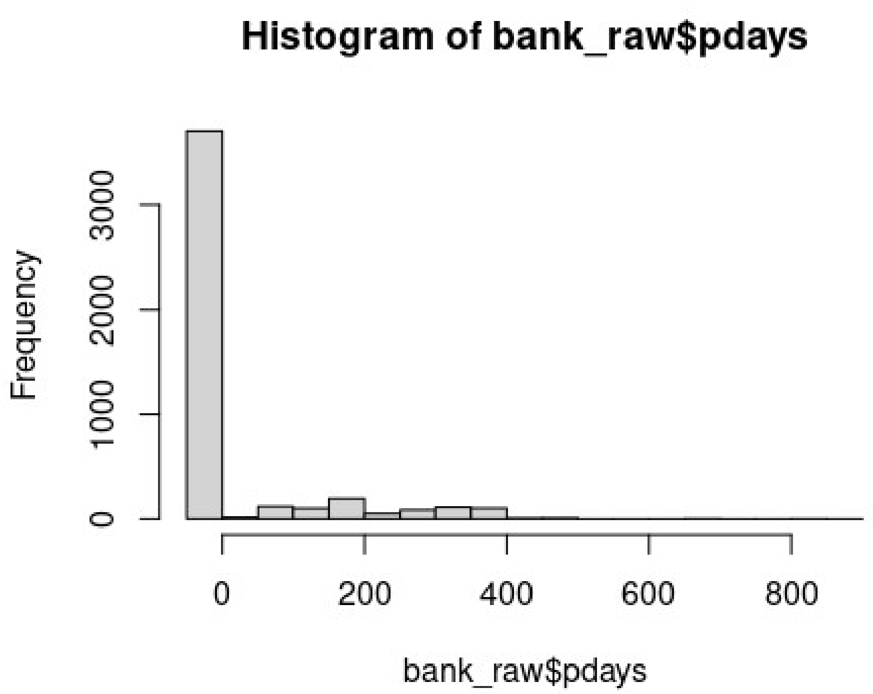

```r
hist(bank_raw$duration)
```



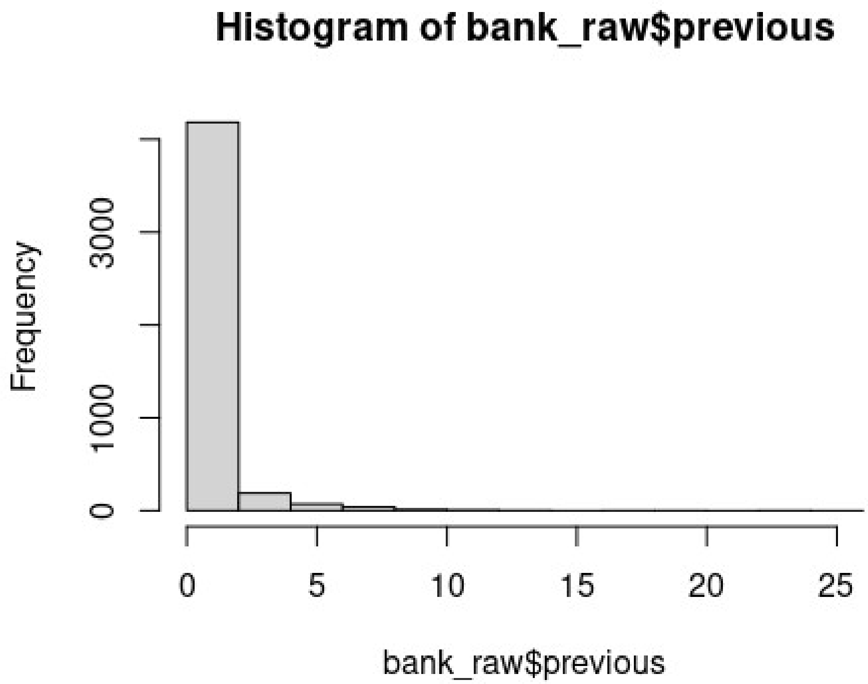

```r
hist(bank_raw$campaign)
```




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




```
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))

# normalize numeric features
#bank_norm <- sapply(bank_raw, function(x) if(is.numeric(x)) {
#                                 scale(x)
#                               } else x)

#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).

```
#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
```

Transformar 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
```

```r
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
```

```r
result <-sapply(test, month_to_number)
result
```

```
## oct may apr jun may
##  10   5   4   6   5
```

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

#bank_raw$month_num <- as.integer(factor(bank_raw$month, levels =
unique(bank_raw$month))) # codifica poniendo los números según
aparecen en los levels, no Enero=1, Febrero=2 ...
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 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
```

```r
#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 <- 
cbin
d(bank_norm,as.numeric(bank_raw$day),bank_raw$month_num,bank_hot_encod
ed_feat,dummy_y$.data_yes)
names(bank_processed)[7:8] <- c("day","month")
names(bank_processed)[43] <- c("y")
head(bank_processed,5)
```

```
##         age    balance   duration   campaign     pdays previous day
month
## 1 0.1617647 0.06845546 0.02482622 0.00000000 0.0000000     0.00  19
10
## 2 0.2058824 0.10875022 0.07149950 0.00000000 0.3899083     0.16  11
5
## 3 0.2352941 0.06258976 0.05991394 0.00000000 0.3795872     0.04  16
4
## 4 0.1617647 0.06428102 0.06454816 0.06122449 0.0000000     0.00   3
6
## 5 0.5882353 0.04446920 0.07348560 0.00000000 0.0000000     0.00   5
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 y
## 1              0                0                1    1     0 0
## 2              0                0                0    1     0 0
## 3              0                0                0    1     0 0
## 4              0                0                1    1     0 0
## 5              0                0                1    1     0 0
```

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) %>
%
    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")
```

```r
  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
```

```
## [1] 3391
```

```r
nrow(X_test_bank)   # 1130
```

```
## [1] 1130
```

```r
nrow(y_train_bank)  # 3391
```

```
## [1] 3391
```

```r
nrow(y_test_bank)   # 1130
```

```
## [1] 1130
```

Paso 3: Entrenamiento del modelo

```r
# neuralnet
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
  })
```

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

  system.time({
      #model <- mlp(bank_processed_train[1:40],
bank_processed_train[41:42], size = c(40,10,4), learnFuncParams =
c(0.05), maxit = 1000)
      model <- mlp(X_train_bank, y_train_bank, size = c(40,10,4,2),
learnFuncParams = c(0.05), maxit = 20)
      # with 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
    )
  })

}
```

```
## [1] "Choosing Keras"
## Epoch 1/1000
## 68/68 - 1s - 17ms/step - accuracy: 0.5018 - loss: 2.0005 -
val_accuracy: 0.8792 - val_loss: 0.4636
## Epoch 2/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.7094 - loss: 0.8915 -
val_accuracy: 0.8792 - val_loss: 0.4181
## Epoch 3/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.7965 - loss: 0.6825 -
val_accuracy: 0.8792 - val_loss: 0.4152
## Epoch 4/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8333 - loss: 0.6048 -
val_accuracy: 0.8792 - val_loss: 0.4213
## Epoch 5/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8547 - loss: 0.5369 -
val_accuracy: 0.8792 - val_loss: 0.4102
## Epoch 6/1000
## 68/68 - 0s - 3ms/step - accuracy: 0.8687 - loss: 0.4974 -
val_accuracy: 0.8792 - val_loss: 0.4163
## Epoch 7/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8717 - loss: 0.4965 -
val_accuracy: 0.8792 - val_loss: 0.4091
## Epoch 8/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8802 - loss: 0.4533 -
val_accuracy: 0.8792 - val_loss: 0.3963
## Epoch 9/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8838 - loss: 0.4461 -
val_accuracy: 0.8792 - val_loss: 0.3884
## Epoch 10/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8857 - loss: 0.4371 -
val_accuracy: 0.8792 - val_loss: 0.3822
## Epoch 11/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8872 - loss: 0.4200 -
val_accuracy: 0.8792 - val_loss: 0.3798
## Epoch 12/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8879 - loss: 0.3963 -
val_accuracy: 0.8792 - val_loss: 0.3692
## Epoch 13/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8879 - loss: 0.4043 -
val_accuracy: 0.8792 - val_loss: 0.3677
## Epoch 14/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8890 - loss: 0.3992 -
val_accuracy: 0.8792 - val_loss: 0.3679
```

```
...
## Epoch 990/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.9001 - loss: 0.2565 -
val_accuracy: 0.8851 - val_loss: 0.3950
## Epoch 991/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8993 - loss: 0.2579 -
val_accuracy: 0.8881 - val_loss: 0.4013
## Epoch 992/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8993 - loss: 0.2571 -
val_accuracy: 0.8881 - val_loss: 0.4021
## Epoch 993/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8979 - loss: 0.2582 -
val_accuracy: 0.8837 - val_loss: 0.4305
## Epoch 994/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.8997 - loss: 0.2537 -
val_accuracy: 0.8866 - val_loss: 0.4068
## Epoch 995/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.9008 - loss: 0.2578 -
val_accuracy: 0.8851 - val_loss: 0.4096
## Epoch 996/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.9019 - loss: 0.2520 -
val_accuracy: 0.8881 - val_loss: 0.4082
## Epoch 997/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.9052 - loss: 0.2433 -
val_accuracy: 0.8851 - val_loss: 0.4141
## Epoch 998/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.9015 - loss: 0.2522 -
val_accuracy: 0.8866 - val_loss: 0.4131
## Epoch 999/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.9034 - loss: 0.2566 -
val_accuracy: 0.8851 - val_loss: 0.4027
## Epoch 1000/1000
## 68/68 - 0s - 2ms/step - accuracy: 0.9008 - loss: 0.2477 -
val_accuracy: 0.8837 - val_loss: 0.4290

##    user  system elapsed
## 207.499  12.250 163.964
```

Visualizamos la arquitectura de la red entrenada y sus pesos:

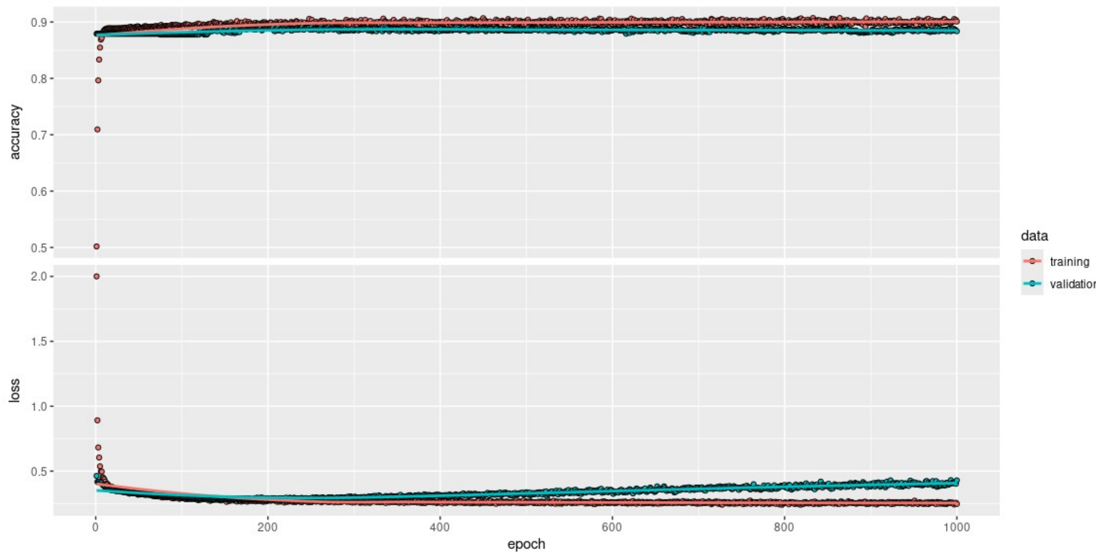
```
# neuralnet
if (library=="neuralnet") {
  plot(model)   #saved in file
"Chapter07/neuralnet_10_neurons_model.png"
```

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

```
## Model: "keras_mid_complex"
##
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━
━━━━━━━━━━┓
## ┃ Layer (type)                         ┃ Output Shape               ┃
Param # ┃
##
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━
━━━━━━━━━━┩
## │ dense_1 (Dense)                      │ (None, 40)                 │
1,640 │
##
├─────────────────────────────────────┼────────────────────────────┼───────
──────────┤
## │ dropout_1 (Dropout)                  │ (None, 40)                 │
0 │
##
├─────────────────────────────────────┼────────────────────────────┼───────
──────────┤
## │ dense_2 (Dense)                      │ (None, 10)                 │
410 │
##
├─────────────────────────────────────┼────────────────────────────┼───────
──────────┤
## │ dropout_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)
```

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



En el caso de utilizar las librerías Keras&TensorFlow, podemos obtener un Json con información del modelo:

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

```
## Registered S3 methods overwritten by 'keras':
##   method                               from
##   as.data.frame.keras_training_history keras3
##   plot.keras_training_history          keras3
##   print.keras_training_history         keras3
##   r_to_py.R6ClassGenerator             keras3
```


```
## {
##     "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
```

```
##              },
##              {
##                  "module": "keras.layers",
##                  "class_name": "Dense",
##                  "config": {
##                      "name": "dense_2",
##                      "trainable": true,
##                      "dtype": {
##                          "module": "keras",
##                          "class_name": "DTypePolicy",
##                          "config": {
##                              "name": "float32"
##                          },
##                          "registered_name": null
##                      },
##                      "units": 10,
##                      "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_2",
##                      "trainable": true,
##                      "dtype": {
##                          "module": "keras",
##                          "class_name": "DTypePolicy",
##                          "config": {
##                              "name": "float32"
##                          },
##                          "registered_name": null
##                      },
##                      "rate": 0.4,
##                      "seed": null,
##                      "noise_shape": null
##                  },
##                  "registered_name": null
##              },
##              {
##                  "module": "keras.layers",
##                  "class_name": "Dense",
##                  "config": {
##                      "name": "output_layer",
##                      "trainable": true,
##                      "dtype": {
##                          "module": "keras",
##                          "class_name": "DTypePolicy",
##                          "config": {
##                              "name": "float32"
##                          },
##                          "registered_name": null
##                      },
##                      "units": 2,
##                      "activation": "sigmoid",
##                      "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,
##                             10
##                         ]
##                     }
##                 }
##             ],
##             "build_input_shape": [
##                 null,
##                 40
##             ]
##         },
##         "registered_name": null,
##         "build_config": {
##             "input_shape": [
##                 null,
##                 40
##             ]
##         },
##         "compile_config": {
##             "optimizer": {
##                 "module": "keras.optimizers",
```

```
##                 "class_name": "Adam",
##                 "config": {
##                     "name": "adam",
##                     "learning_rate": 0.0010000000474974513,
##                     "weight_decay": null,
##                     "clipnorm": null,
##                     "global_clipnorm": null,
##                     "clipvalue": null,
##                     "use_ema": false,
##                     "ema_momentum": 0.99,
##                     "ema_overwrite_frequency": null,
##                     "loss_scale_factor": null,
##                     "gradient_accumulation_steps": null,
##                     "beta_1": 0.9,
##                     "beta_2": 0.999,
##                     "epsilon": 1e-07,
##                     "amsgrad": false
##                 },
##                 "registered_name": null
##             },
##             "loss": "binary_crossentropy",
##             "loss_weights": null,
##             "metrics": [
##                 "accuracy"
##             ],
##             "weighted_metrics": null,
##             "run_eagerly": false,
##             "steps_per_execution": 1,
##             "jit_compile": false
##         }
## }
##
```

Primeros resultados con la librería neuralnet:

(0) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c(“y”))], hidden=1) hidden=1
User System verstrichen 0.86 0.00 0.91

(1) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c(“y”))], hidden=10) Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax. <- !!!!!! User System verstrichen 484.682 1.604 489.933

(2) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = c(20,10), threshold = 0.05, algorithm = "rprop+") Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
User System verstrichen 1158.842 2.555 1161.369

(3) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = c(10,2)) Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax.
User System verstrichen 487.436 0.162 487.498

(4) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = c(10,2), threshold = 0.1, lifesign="full") User System verstrichen 274.745 0.245 274.977

(5) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = 20, algorithm = "rprop+", threshold = 0.5, lifesign="full")
User System verstrichen 177.455 1.304 179.895

(6) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = 43, threshold = 0.5, lifesign="full")

(7) Warning: Algorithm did not converge in 1 of 1 repetition(s) within the stepmax. User System verstrichen 2334.860 5.836 2356.510
model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = 43, act.fct = softplus, threshold = 0.5, lifesign="full")

Paso 4: Evaluación del modelo

```
if (library=="keras") {
  model %>% evaluate(X_test_bank, y_test_bank)
}

## 36/36 - 0s - 2ms/step - accuracy: 0.8796 - loss: 0.3553

## $accuracy
## [1] 0.879646
##
## $loss
## [1] 0.3553326
```

Una vez entrenado el modelo, pasamos a analizar su capacidad predictiva:

```r
# neuralnet
if (library=="neuralnet") {
  #prediction <- compute(model, bank_processed_test[ , -which(names(bank_processed_test) %in% c("y"))])  #compute is deprecated, 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, bank_processed_test[, 1:40])

} else {  #Keras

  predictions <- model %>% predict(X_test_bank)

}
```

```
## 36/36 - 0s - 3ms/step
```

```r
# neuralnet, RSNNS, keras

prediction <- apply(predictions,1,which.max)  #find which column has the highest value

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

if (library=="keras") {
  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"
} else {
  y_test_bank_real <- y_test_bank

  y_test_bank_real[y_test_bank_real==0] <- "no"
  y_test_bank_real[y_test_bank_real==1] <- "yes"
}
```

Matriz de confusión:

```r
# Confussion matrix
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  988   5
##        yes 131   6
##
##                Accuracy : 0.8796
##                  95% CI : (0.8592, 0.8981)
##     No Information Rate : 0.9903
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0642
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.545455
##             Specificity : 0.882931
##          Pos Pred Value : 0.043796
##          Neg Pred Value : 0.994965
##               Precision : 0.043796
##                  Recall : 0.545455
##                      F1 : 0.081081
##              Prevalence : 0.009735
##          Detection Rate : 0.005310
##    Detection Prevalence : 0.121239
##       Balanced Accuracy : 0.714193
##
##        'Positive' Class : yes
##
```

Resultados con la librería neuralnet:

(0) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden=1)

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 35 958 yes 48 89

```
          Accuracy : 0.1097


   'Positive' Class : no
```

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 64 929 yes 44 93

(4) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = c(10,2), threshold = 0.1, lifesign="full")

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 60 933 yes 52 85

```
          Accuracy : 0.1283


   'Positive' Class : no
```

(5) model <- neuralnet(y.yes+y.no ~ ., data = bank_processed_train[, -which(names(bank_processed_train) %in% c("y"))], hidden = 20, algorithm = "rprop+", threshold = 0.5, lifesign="full")
User System verstrichen 177.455 1.304 179.895

```
      Reference
```

Prediction no yes no 61 932 yes 38 99

```
          Accuracy : 0.1416
```

(6) 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")
User System verstrichen 245.882 2.815 250.844

```
      Reference
```

Prediction no yes no 88 905 yes 51 86

```
          Accuracy : 0.154
```

ATENCIÓN: Redes neuronales no son buenas con matrices dispersas … ?

Resultados con la librería RSNNS:

(1) model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(10), learnFuncParams = c(0.1), maxit = 1000) predictions targets 1 2 1 964 29 2 108 29

(2) model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(10), learnFuncParams = c(0.1), maxit = 10000) User System verstrichen 122.006 0.306 123.809

```
      Reference
```

Prediction no yes no 948 45 yes 115 22

```
           Accuracy : **0.8584**
             95% CI : (0.8367, 0.8782)
```

No Information Rate : 0.9407
P-Value [Acc > NIR] : 1

```
              Kappa : 0.1478
```

Mcnemar’s Test P-Value : 4.899e-08

```
        Sensitivity : 0.32836
        Specificity : 0.89182
     Pos Pred Value : 0.16058
     Neg Pred Value : 0.95468
         Prevalence : 0.05929
     Detection Rate : 0.01947
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.61009

```
   'Positive' Class : yes
```

(3) model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(43,10), learnFuncParams = c(0.1), maxit = 10000) User System verstrichen 681.628 0.759 687.135

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 931 62 yes 81 56

```
           Accuracy : 0.8735
             95% CI : (0.8526, 0.8923)
No Information Rate : 0.8956
P-Value [Acc > NIR] : 0.9923

              Kappa : 0.3683
```

Mcnemar’s Test P-Value : 0.1323

```
        Sensitivity : 0.47458
        Specificity : 0.91996
     Pos Pred Value : 0.40876
     Neg Pred Value : 0.93756
         Prevalence : 0.10442
     Detection Rate : 0.04956
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.69727

```
   'Positive' Class : yes
```

(4) Ajustamos el número de neuronas de la primera capa oculta, y bajamos las iteraciones a 10000. La predicción se hace en mucho menos tiempo (menos de la decima parte), y el resultado es algo mejor:

model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(40,10), learnFuncParams = c(0.1), maxit = 1000)
User System verstrichen 46.186 0.050 46.445

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 975 18 yes 117 20

```
           Accuracy : 0.8805
             95% CI : (0.8602, 0.8989)
No Information Rate : 0.9664
P-Value [Acc > NIR] : 1

              Kappa : 0.1857
```

Mcnemar’s Test P-Value : <2e-16

```
        Sensitivity : 0.52632
        Specificity : 0.89286
     Pos Pred Value : 0.14599
     Neg Pred Value : 0.98187
         Prevalence : 0.03363
     Detection Rate : 0.01770
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.70959

```
   'Positive' Class : yes
```

(5) No hay mejora al añadir 10 neuronas más en la segunda capa

model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(40,20), learnFuncParams = c(0.1), maxit = 1000)
User System verstrichen 57.033 0.060 57.301

```
      Reference
```

Prediction no yes no 965 28 yes 108 29

```
           Accuracy : 0.8796
```

(6) Sí mejora añadir una tercera capa con 4 neuronas!

model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(40,10,4), learnFuncParams = c(0.05), maxit = 1000) User System verstrichen 45.978 0.058 46.650 Toshiba user system elapsed 28.122 0.000 28.188 Lenovo

```
      Reference
```

Prediction no yes no 953 40 yes 89 48

```
           Accuracy : 0.8858


 Probar finalmente:
```

(7) Con 2 capas no mejora añadir iteraciones model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(40,10), learnFuncParams = c(0.1), maxit = 10000) User System verstrichen 643.455 0.429 645.196

```
    Reference
```

Prediction no yes no 955 38 yes 102 35

```
        Accuracy : 0.8761
```

(8) Sin añadir más capas, usar ReLu (softplus) model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(40,10), hiddenActFunc=softplus, learnFuncParams = c(0.1), maxit = 1000)

Cancelado. Llevaba más de 90 min.

(9) Con una cuarta capa, mejora ligeramente el resultado de 3 capas (**mejor resultado de todos**) model <- mlp(bank_processed_train[1:40], bank_processed_train[41:42], size = c(40,10,4,2), learnFuncParams = c(0.05), maxit = 2000) user system elapsed 65.524 0.460 57.800

Confusion Matrix and Statistics

```
          Reference
```

Prediction no yes no 965 28 yes 91 46

```
              Accuracy : 0.8947
```

Resultados con la librería Keras&TensorFlow:

keras_complex:

model <- keras_model_sequential(name = “keras_complex”, input_shape = ncol(X_train_bank)) model %>% layer_dense(name = “layer_1”,units = 40, activation = ‘relu’) %>% layer_dropout(name = “droput_2”, rate = 0.4) %>% layer_dense(name = “layer_3”, units = 20, activation = ‘relu’) %>% layer_dropout(name = “droput_4”, rate = 0.3) %>% layer_dense(name = “layer_5”, units = 10, activation = ‘relu’) %>% layer_dropout(name = “droput_6”, rate = 0.15) %>% layer_dense(name = “layer_7”, units = 4, activation = ‘relu’) %>% layer_dense(name = “output_layer_8”, 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) })

Confusion Matrix and Statistics

```
          Reference
```

Prediction no yes no 953 40 yes 91 46

```
           Accuracy : 0.8841                  <- Quitando validación, la exactitud es 0.8717.
             95% CI : (0.864, 0.9022)            Probando con otros valores en las capas dropout, se obtiene 0.8788
No Information Rate : 0.9239
P-Value [Acc > NIR] : 1

              Kappa : 0.352
```

Mcnemar’s Test P-Value : 1.251e-05

```
        Sensitivity : 0.53488
        Specificity : 0.91284
     Pos Pred Value : 0.33577
     Neg Pred Value : 0.95972
          Precision : 0.33577
             Recall : 0.53488
                 F1 : 0.41256
         Prevalence : 0.07611
     Detection Rate : 0.04071
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.72386

```
   'Positive' Class : yes
```

keras_mid_complex:

model <- keras_model_sequential(name = “keras_mid_complex”, input_shape = ncol(X_train_bank)) model %>% layer_dense(name = “layer_1”,units = 40, activation = ‘relu’) %>% layer_dropout(name = “droput_2”, rate = 0.4) %>%

```
layer_dense(name = "layer_5", units = 10, activation = 'relu') %>%
layer_dropout(name = "droput_6", rate = 0.15) %>%

layer_dense(name = "output_layer_8", 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) })

```
user  system elapsed
```

259.342 15.925 315.008

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 953 40 yes 99 38

```
           Accuracy : 0.877
             95% CI : (0.8564, 0.8956)
No Information Rate : 0.931
P-Value [Acc > NIR] : 1

              Kappa : 0.2911
```

Mcnemar’s Test P-Value : 8.677e-07

```
        Sensitivity : 0.48718
        Specificity : 0.90589
     Pos Pred Value : 0.27737
     Neg Pred Value : 0.95972
          Precision : 0.27737
             Recall : 0.48718
                 F1 : 0.35349
         Prevalence : 0.06903
     Detection Rate : 0.03363
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.69654

```
   'Positive' Class : yes
```

keras_mid_complex lots of neurons:

model <- keras_model_sequential(name = “keras_mid_complex”, input_shape = ncol(X_train_bank)) model %>% layer_dense(name = “layer_1”,units = 120, activation = ‘relu’) %>% layer_dropout(name = “droput_2”, rate = 0.5) %>% layer_dense(name = “layer_2”,units = 40, activation = ‘relu’) %>% 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) })

```
      Reference
```

Prediction no yes no 946 47 yes 92 45

```
           Accuracy : 0.877
```

Mismo modelo, pero sin validación y con epoch=500:

```
      Reference
```

Prediction no yes no 958 35 yes 91 46

```
           Accuracy : 0.8885
```

keras_40_10_2:

model <- keras_model_sequential(name = “keras_40_10_2”, input_shape = ncol(X_train_bank)) model %>% layer_dense(name = “layer_1”,units = 40, activation = ‘relu’) %>%

```
layer_dense(name = "layer_5", units = 10, activation = 'relu') %>%

layer_dense(name = "output_layer_8", 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) })

user system elapsed 249.202 16.037 309.785

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 944 49 yes 97 40

```
           Accuracy : 0.8708
             95% CI : (0.8498, 0.8898)
No Information Rate : 0.9212
P-Value [Acc > NIR] : 1.0000000
```

```
             Kappa : 0.2858
```

Mcnemar’s Test P-Value : 0.0001003

```
       Sensitivity : 0.44944
       Specificity : 0.90682
    Pos Pred Value : 0.29197
    Neg Pred Value : 0.95065
         Precision : 0.29197
            Recall : 0.44944
                F1 : 0.35398
        Prevalence : 0.07876
    Detection Rate : 0.03540
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.67813

```
   'Positive' Class : yes
```

keras_40_10_4_2

model <- keras_model_sequential(name = “keras_40_10_4_2”, input_shape = ncol(X_train_bank)) model %>% layer_dense(name = “layer_1”,units = 40, activation = ‘relu’) %>% layer_dense(name = “layer_2”, units = 10, activation = ‘relu’) %>% layer_dense(name = “layer_3”, units = 4, activation = ‘relu’) %>% layer_dense(name = “output_layer_4”, 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 = 10000, batch_size = 40, validation_split = 0.2) })

user system elapsed 2552.557 168.099 3210.650

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 926 67 yes 90 47

```
          Accuracy : 0.8611
            95% CI : (0.8395, 0.8807)
No Information Rate : 0.8991
P-Value [Acc > NIR] : 0.99998
```

```
              Kappa : 0.2971
```

Mcnemar’s Test P-Value : 0.07912

```
        Sensitivity : 0.41228
        Specificity : 0.91142
     Pos Pred Value : 0.34307
     Neg Pred Value : 0.93253
          Precision : 0.34307
             Recall : 0.41228
                 F1 : 0.37450
         Prevalence : 0.10088
     Detection Rate : 0.04159
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.66185

```
   'Positive' Class : yes
```

keras_20_2

model <- keras_model_sequential(name = “keras_20_2”, input_shape = ncol(X_train_bank)) model %>% layer_dense(name = “layer_1”,units = 20, activation = ‘relu’) %>%

```
layer_dense(name = "output_layer_8", units = 2, activation =
'sigmoid')
```

user system elapsed 246.888 16.245 306.973

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 952 41 yes 97 40

```
           Accuracy : 0.8779
             95% CI : (0.8574, 0.8964)
No Information Rate : 0.9283
P-Value [Acc > NIR] : 1

              Kappa : 0.3043
```

Mcnemar’s Test P-Value : 2.842e-06

```
        Sensitivity : 0.49383
        Specificity : 0.90753
     Pos Pred Value : 0.29197
     Neg Pred Value : 0.95871
          Precision : 0.29197
             Recall : 0.49383
                 F1 : 0.36697
         Prevalence : 0.07168
     Detection Rate : 0.03540
```

Detection Prevalence : 0.12124
Balanced Accuracy : 0.70068

```
   'Positive' Class : yes
```

keras_20_2 exponential

model <- keras_model_sequential(name = “keras_20_2”, input_shape = ncol(X_train_bank)) model %>% layer_dense(name = “layer_1”,units = 20, activation = ‘exponential’) %>%

```
layer_dense(name = "output_layer_8", 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) })

Confusion Matrix and Statistics

```
      Reference
```

Prediction no yes no 931 62 yes 94 43

```
           Accuracy : 0.8619
```

Da igual el valor de epoch, el número de capas internas, y la función de activación. Es como si hubiera un muro en el 88-89%.