Redes Neuronales en R

Jaqueline Girabel y Agustín Muñoz González

25/8/2020

Preparamos el entorno.

```
rm(list=ls())
library(ggplot2)
library(neuralnet)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::compute() masks neuralnet::compute()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

```
library(purrr)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(igraph)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:purrr':
##
##     compose, simplify
```

```
## The following object is masked from 'package:tidyr':
##
##     crossing
```

```
## The following object is masked from 'package:tibble':
##
##     as_data_frame

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```
library(ggnetwork)
```

En este trabajo exhibimos, en dos ejemplos distintos, la manera de implementar un modelo predictivo entrenando redes neuronales.

Nuestro primer ejemplo tiene como finalidad mostrar cómo funciona una red neuronal visualizando en los parámetros de la funcionalidad *neuralnet* algunas de las características más importantes de su arquitectura. Para ello, ajustaremos un modelo de regresión para los datos *lidar*, donde *logratio* es la variable respuesta.

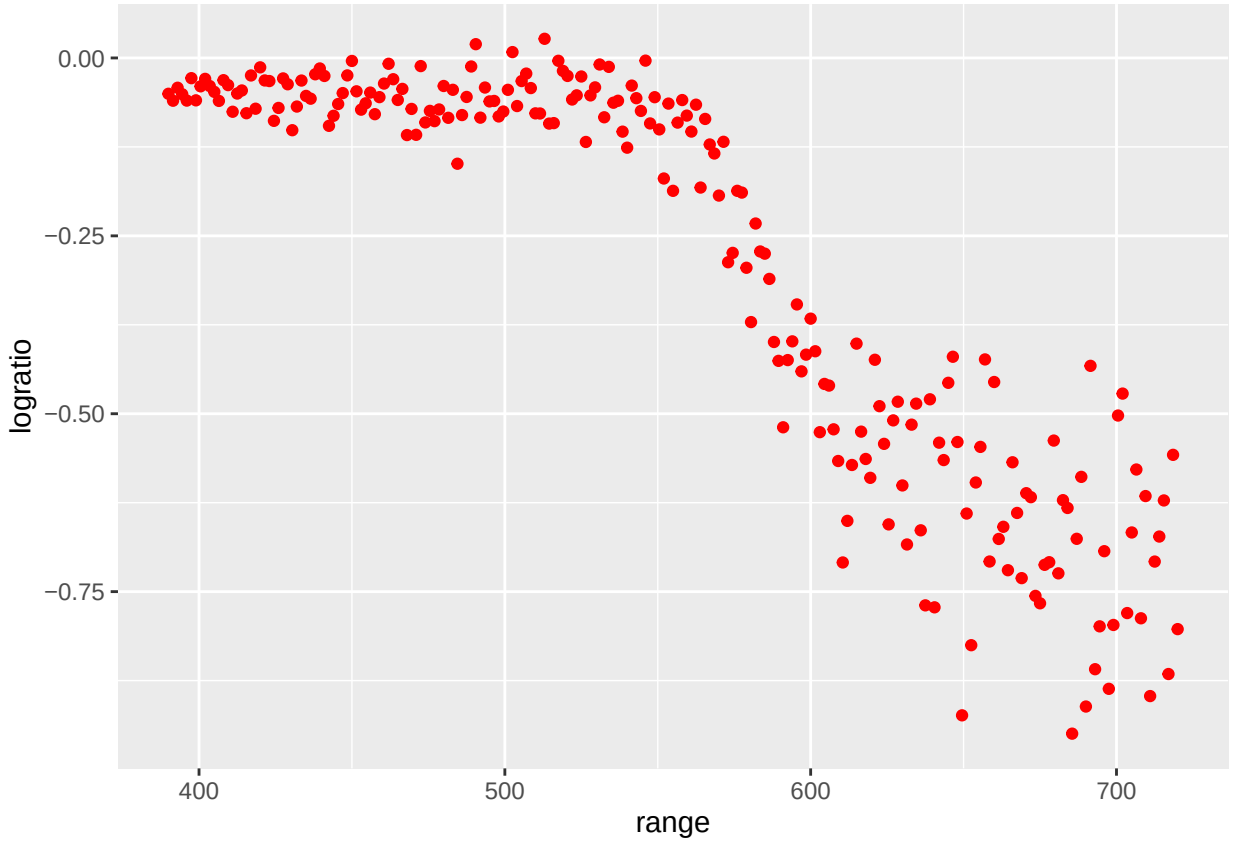
Luego, en un segundo ejemplo, modelamos un clasificador mediante redes neuronales, para lo cual, primero hacemos un análisis de reducción de variables explicativas en un conjunto de datos que dispone de más de 16000 columnas, y en la variable respuesta tenemos mútiples clases.

En todos los modelos obtenidos, en ambos problemas, calculamos el error de predicción y comparamos los resultados.

Modelo de regresión

Comenzamos cargando los datos del archivo *lidar.txt* y mostrando en un gráfico la distribución del *logratio* en función de la variable explicativa *range*.

```
data<-read.table("lidar.txt", header = T, sep = "")
names(data)<-c("range", "logratio")

ggplot(data, aes(x=range))+geom_point(aes(y=logratio), color="red")
```



Separamos un 80% de los datos para el entrenamiento del modelo, quedándonos con el 20% restante para testeo.

```
set.seed(1234)
indices <- sample(1:nrow(data),round(0.8*nrow(data)))
train.data <- as.data.frame(data[indices,])
test.data <- as.data.frame(data[-indices,])

attach(train.data)
names(train.data)
```

```
## [1] "range"    "logratio"
```

En este primer modelo basado en los datos *lidar*, usaremos la funcionalidad **neuralnet** que nos permite entrenar una (o más de una) red neuronal según las especificaciones y metodologías indicadas en sus argumentos.

```r
# Definimos la fórmula del modelo que queremos ajustar
f <- as.formula("logratio ~ range")
```

La fórmula f que acabamos de definir indica cuál es el modelo que queremos ajustar. Del resto de los argumentos requeridos por el modelo *neuralnet*, estudiemos algunos de los más importantes:

```r
nnet.1 <- neuralnet(f, data, hidden=1,
                            startweights= NULL,
                            threshold = 0.01,
                            linear.output = TRUE)
```

Especificamos TRUE o FALSE en **linear.output** si queremos un modelo de regresión o de clasificación, respectivamente.

El argumento **startweights** contiene los valores de inicialización de los pesos. Si dos neuronas tienen exactamente los mismos pesos, siempre tendrán el mismo gradiente, y no serán capaces de aprender características diferentes. Los pesos son inicializados con valores aleatorios y pequeños.

Threshold es un criterio de stopping para las derivadas parciales de la función de error. Por default, *neuralnet* requiere que las derivadas parciales cambien al menos 0.01.

```r
set.seed(1234)
nnet.1 <- neuralnet(f, data, hidden=1,
                            learningrate.factor = list(minus = 0.5, plus = 1.2),
                            algorithm = "rprop+",
                            linear.output = TRUE)
```

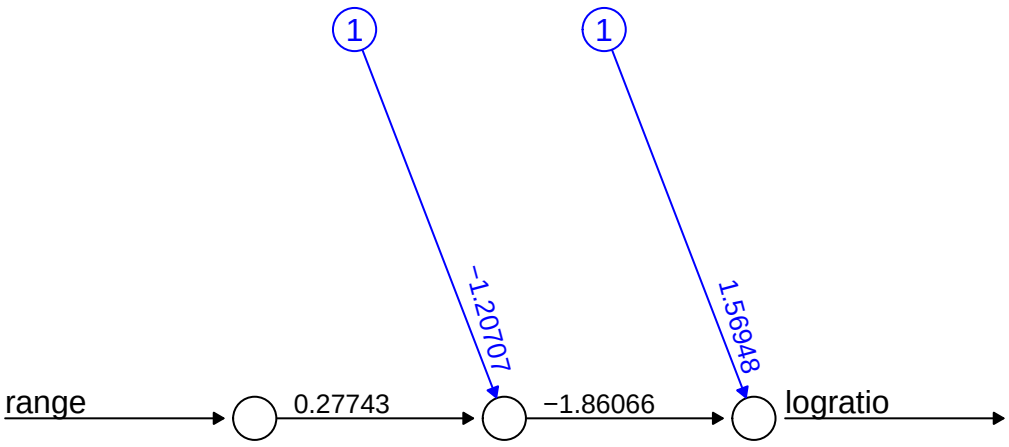
Por default, el algoritmo utilizado para calcular el gradiente de la función de error es **RPROP+** (resilient backpropagation). Este difiere del algoritmo de backpropagation clásico en el ajuste del *learning rate*, en el proceso de optimización del error para encontrar un mínimo local.

El propósito del algoritmo **RPROP+** es eliminar los efectos de las *magnitudes* de las derivadas parciales, dado que sólo el signo de la derivada determina la dirección de la actualización de los pesos, la magnitud de la derivada no tiene ningún efecto sobre esta actualización.

El tamaño del cambio de peso se determina de la siguiente manera: se incrementa en un factor δ_1 siempre que la derivada de la función de error con respecto a ese peso tenga el mismo signo para dos iteraciones sucesivas; y el valor de actualización se reduce en un factor δ_2 siempre que la derivada con respecto a ese peso cambie de signo. Siempre que los pesos oscilan, se reduce el cambio de peso. Si el peso sigue cambiando en la misma dirección durante varias iteraciones, la magnitud del cambio de peso aumenta.

RPROP no establece un learning rate fijo, estático, y esto lo hace un algoritmo más rápido comparado con el algoritmo clásico de backpropagation. Sin embargo, resilent backpropagation requiere del argumento **learningrate.factor**, que modifica la tasa de aprendizaje mediante los factores *minus* (si el algoritmo de optimización se saltó los minimos locales) y *plus* (si el algoritmo va en la dirección correcta). Por default, minus = 0.5 y plus = 1.2.

```r
plot(nnet.1, rep = "best")
```



Error: 8.777122 Steps: 30

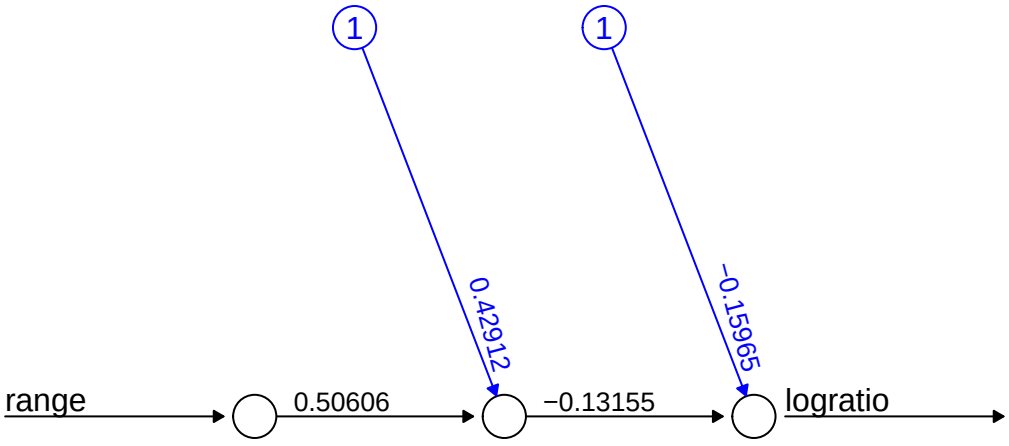
Si queremos usar el algoritmo de backpropagation clásico

```
nnet.1.1<-neuralnet(f, data, algorithm = "backprop")
```

```
## Error: Argument 'learningrate' must be a numeric value, if the backpropagation algorithm is used.
```

necesitamos especificar el learning rate, que es el parámetro más importante ascociado a backpropagation.

```
nnet.1.1<-neuralnet(f, data, algorithm = "backprop", learningrate = 0.0001)
plot(nnet.1.1, rep = "best")
```



Error: 8.777122 Steps: 219

Determinar el learning rate es otro de los puntos cruciales en la arquitectura de una red neuronal. El entrenamiento progresa muy lentamente si su learning rate es muy chico, ya que realizaría mínimos ajustes en los pesos de la red, y esto también podría provocar overfitting. Por otro lado, con una tasa alta, el algoritmo de optimización de los parámetros puede dar saltos rebotando alrededor del mínimo local; de hecho, posiblemente se pierdan los mínimos locales y el algortimo diverja.

Por último, el argumento **hidden** es un vector cuya dimensión nos indica la cantidad de *capas* de la red neuronal, y cada coordenada indica la cantidad de neuronas en ellas. En general, la cantidad de nodos de las capas inferiores debe superar a la cantidad de nodos de las capas siguientes. Sin embargo, el número de hiddens y de nodos no sólo depende de los datos y del tipo de modelo que queramos obtener (clasificación o regresión), determinar el vector de hiddens es un arte y una ciencia en sí mismo.

Si los datos pueden separarse linealmente, entonces no necesitamos ninguna capa y, por lo tanto, tampoco necesitamos redes neuronales para modelar los datos. Una red con 0 capas sólo es capaz de representar funciones lineales.

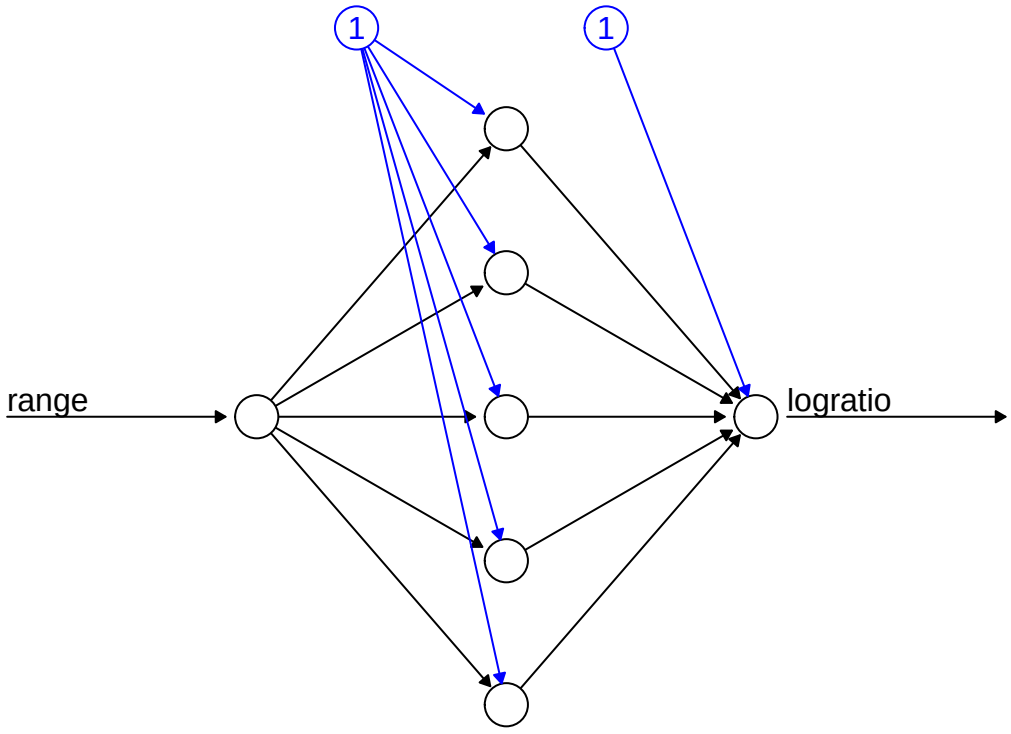
Más allá de eso, las situaciones en las que el rendimiento de la red neuronal mejora con dos o más capas son pocas, y una capa es suficiente para la mayoría de los problemas. Una red con 1 capa puede aproximar prácticamente cualquier función continua, y una red neuronal con 2 o más capas se emplea para representar, por ejemplo, bordes de decisión arbitrarios en problemas de clasificación.

En cuanto a la cantidad de neuronas, una de las reglas más comúnmente utilizadas nos dice que la cantidad de nodos suele estar entre el tamaño de la entrada y el tamaño de la salida (input/output layers).

Veamos qué obtenemos modificado la cantidad de capas y neuronas en capas.

Ocultamos los pesos en el dibujo para que quede más prolijo.

```
set.seed(1234)
nnet.2<-neuralnet(f, data, hidden=5, threshold = 0.1)
plot(nnet.2, rep = "best",show.weights=F)
```

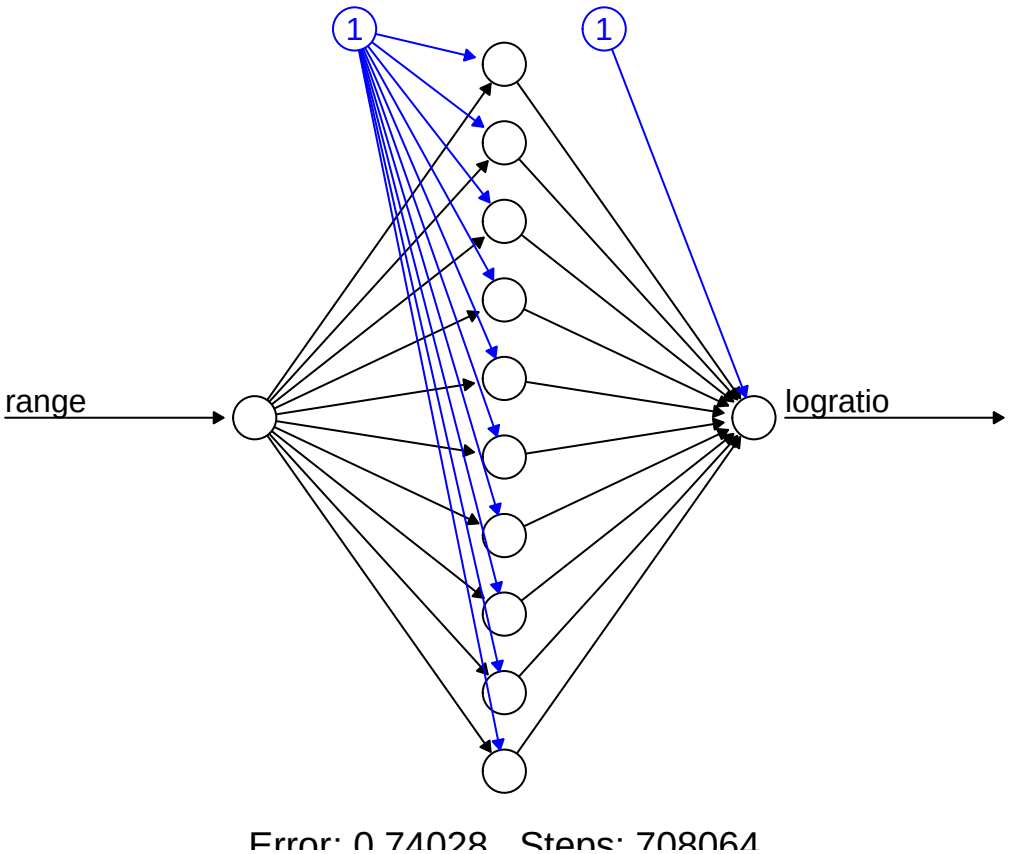


Error: 1.155095 Steps: 3896

```
nnet.2.1<-neuralnet(f, data, hidden=10,
                          threshold=0.1)
```

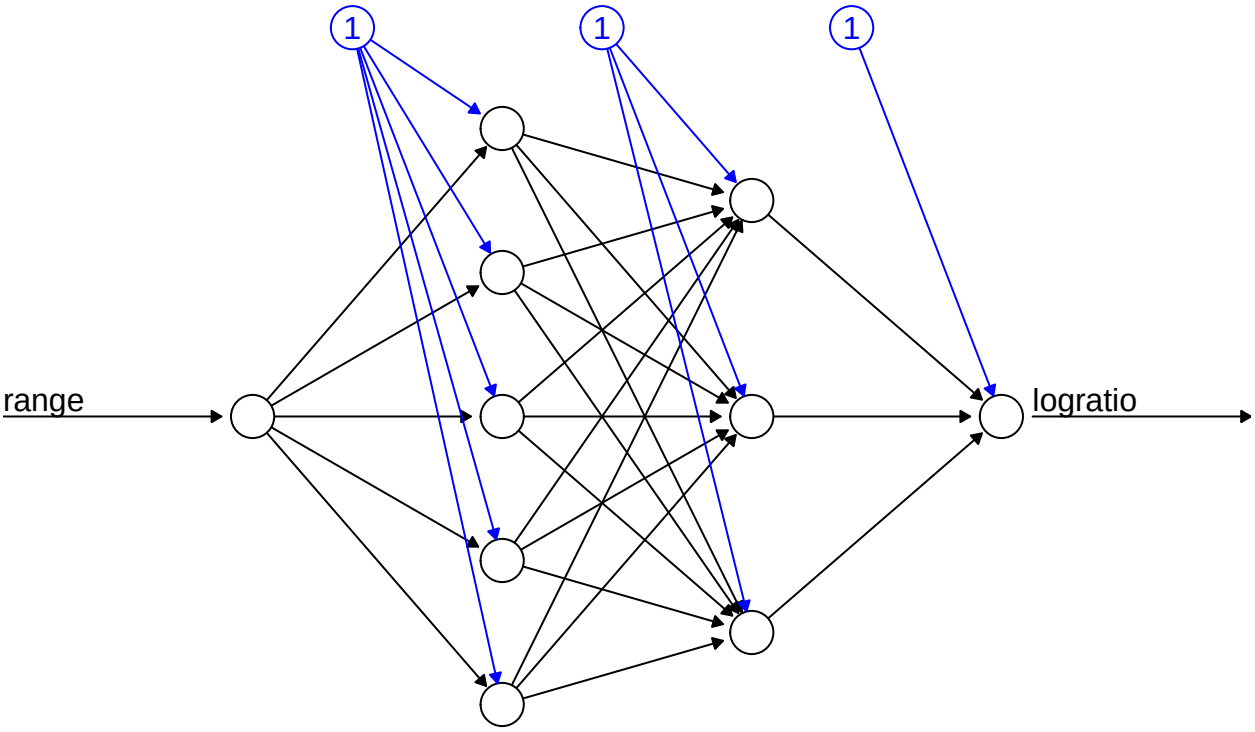
Stepmax controla cuánto se tarda en entrenar la red neuronal, en el sentido de cantidad de iteraciones. Podemos aumentar el stepmax y, así, darle más tiempo para que converja.

```
set.seed(123)
nnet.2.1 <- neuralnet(f, data, hidden=10, threshold=0.1, stepmax = 1e6)
plot(nnet.2.1, rep = "best",show.weights=F)
```



Agregamos una segunda capa:

```
set.seed(1234)
nnet.3 <- neuralnet(f, data, hidden=c(5,3), threshold=0.1, stepmax = 1e8)
plot(nnet.3, rep = "best",show.weights=F)
```



Un argumento que no estamos especificando es **act.fct** para determinar la función de activación, la cual le da al modelo la propiedad de "no linealidad". La función *logística* es la función de activación por default, una de las más utilizadas para el entrenamiento de redes neuronales.

Aplicamos los modelos al set de testing y calculamos los errores medios.

```
library(tidyverse)
prediccion.nnet.1 <- predict(nnet.1,test.data)
ECM.nnet.1<-mean((test.data$logratio - prediccion.nnet.1)^2)
ECM.nnet.1
```

```
## [1] 0.06196506
```

```
nnet.1$result.matrix
```

```
##                                [,1]
## error                   8.77712158
## reached.threshold       0.00450666
## steps                  30.00000000
## Intercept.to.1layhid1 -1.20706575
## range.to.1layhid1       0.27742924
## Intercept.to.logratio   1.56948118
## 1layhid1.to.logratio   -1.86065770
```

```
#usando backpropagation clásico
prediccion.nnet.1.1 <- predict(nnet.1.1,test.data)
```

```
ECM.nnet.1.1<-mean((test.data$logratio - prediccion.nnet.1.1)^2)
ECM.nnet.1.1
```

```
## [1] 0.06196805
```

```
prediccion.nnet.2 <- predict(nnet.2,test.data)
ECM.nnet.2<-mean((test.data$logratio - prediccion.nnet.2)^2)
ECM.nnet.2
```

```
## [1] 0.007759614
```

```
prediccion.nnet.3 <- predict(nnet.3,test.data)
ECM.nnet.3<-mean((test.data$logratio - prediccion.nnet.3)^2)
ECM.nnet.3
```

```
## [1] 0.005163067
```

En términos del error, el mejor modelo es **nnet.3**. Comparemos este con un modelo lineal utilizando las funcionalidades *lm* y *predict*. Proponemos un polinomio de grado 4:

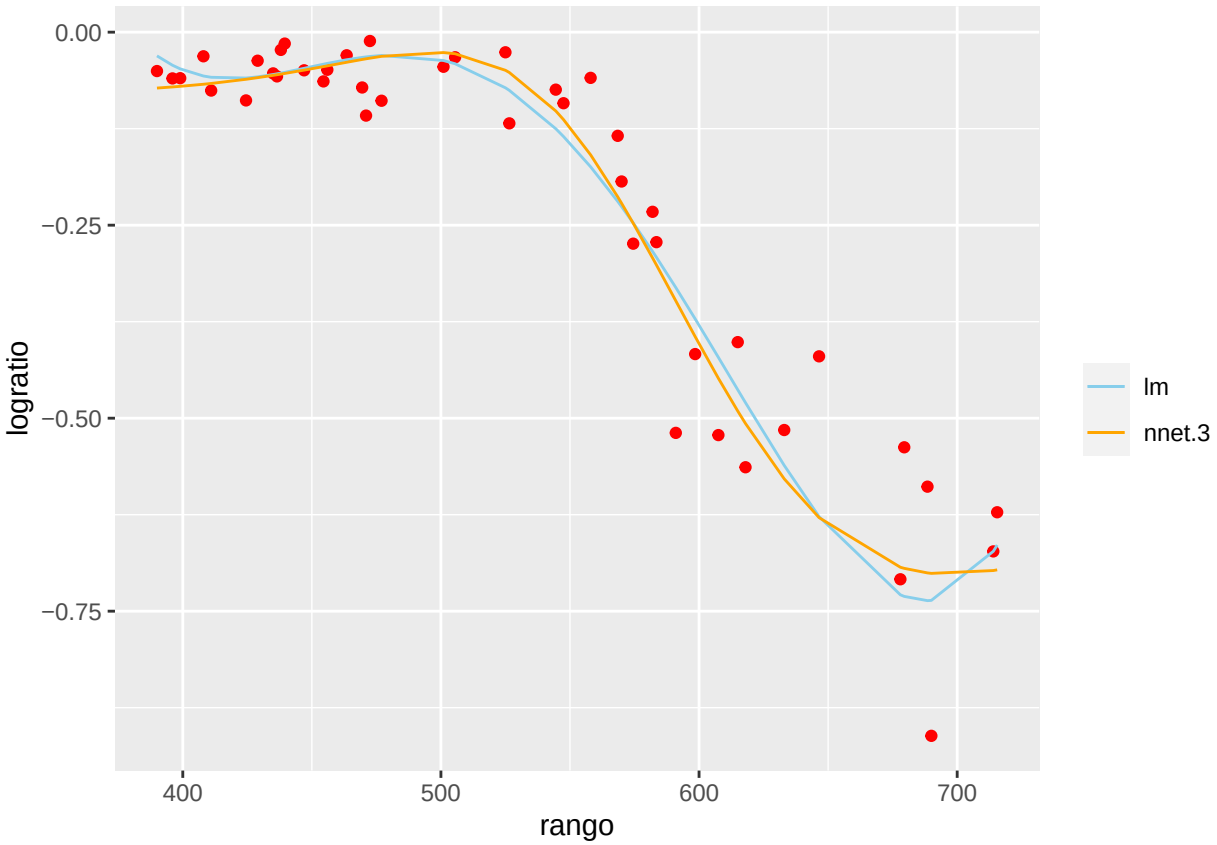
```
range2<-(train.data$range)^2
range3<-(train.data$range)^3
range4<-(train.data$range)^4

modelo.lm<-lm(logratio ~ range+range2+range3+range4, train.data)

test.data$range2<-(test.data$range)^2
test.data$range3<-(test.data$range)^3
test.data$range4<-(test.data$range)^4

prediccion.lm <- predict(modelo.lm, test.data)
ECM.lm <- mean((prediccion.lm - test.data$logratio)^2)
ECM.lm
```

```
## [1] 0.005604333
```

El error más chico lo obtenemos utilizando el modelo nnet.3

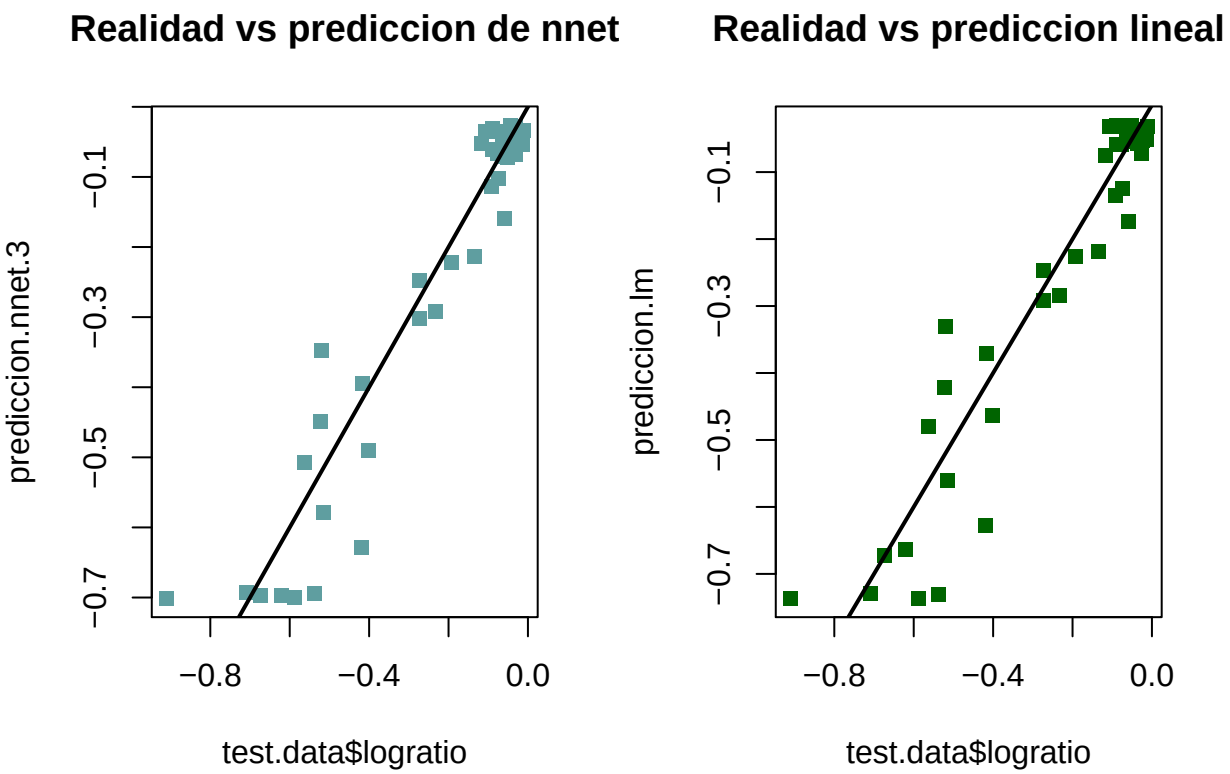
Veamos la comparación gráfica de ambas estimaciones.

```
datos_plot=data.frame(cbind('rango'=test.data$range,
                            'logratio'=test.data$logratio,
                            'prediccion.lm'=prediccion.lm,
                            'prediccion.nnet.3'=prediccion.nnet.3))
datos_plot %>%
  ggplot()+
  geom_point(aes(x=rango,y=logratio),col='red')+
  geom_path(aes(x=rango,y=prediccion.lm,col='lm'))+
  geom_line(aes(x=rango,y=prediccion.nnet.3,col='nnet.3'))+
  scale_color_manual("",
                     breaks=c('lm','nnet.3'),
                     values=c('skyblue','orange'))
```



Otra forma gráfica de ver la performance de cada modelo es hacer un scatterplot de los datos reales vs los datos predichos de cada modelo, junto con la recta $y = x$, es decir, la función identidad. De este modo aquel scatterplot con sus puntos más concentrados alrededor de esta recta será el más preciso.

```
par(mfrow=c(1,2))
plot(test.data$logratio,prediccion.nnet.3,
     col='cadetblue',main='Realidad vs prediccion de nnet',pch=15, cex=1)
abline(0,1,lwd=2)
plot(test.data$logratio,prediccion.lm,col='darkgreen',
     main='Realidad vs prediccion lineal',pch=15, cex=1)
abline(0,1,lwd=2)
```



Se puede ver, aunque no está tan claro en este gráfico, que los puntos predichos por el modelo nnet están mas concentrados alrededor de la recta.

Modelo de clasificación

En el ejemplo anterior, vimos algunos puntos básicos sobre cómo funciona un modelo predictivo entrenando redes neuronales. Ahora queremos estudiar un problema de clasificación con múltiples clases.

El objetivo es emplear y mostrar algunos métodos para reducir la cantidad de variables explicativas y, luego, ajustar un modelo de clasificación con redes neuronales. Concretamente, para este análisis empleamos datos que consisten en muestras de tumores y su expresión genética, y queremos encontrar patrones en estos niveles de expresión que permitan clasificar distintos tipos de tumores de forma correcta.

Limpiamos los registros para liberar el espacio de la memoria y cargamos los datos.

```
rm(list=ls())
datos<-read.delim("GCM_Total_clean.txt", header = TRUE,
                  sep = "\t", quote = "\"", dec = ".", fill = TRUE)

head(datos)[,1:10]
```

```
##   id_muestra tipo_muestra tipo_tumor subtipo_tumor A28102_at AA000993_at
## 1          1       Normal    Bladder            95        57       124.0
## 2          2       Normal    Bladder            BL        63        59.8
## 3          3       Normal    Bladder            BL       130       689.0
## 4          4       Normal    Bladder            BL       132       313.0
## 5          5       Normal    Bladder            BL       184        21.0
## 6          6       Normal    Bladder            BL       135       294.0
##   AA001296_s_at AA002245_at AA004231_at AA004333_at
## 1         381.0       -23.0        61.0        62.0
## 2         293.6       -11.1        70.1      -260.8
## 3         603.0       -38.0       115.0      -538.0
## 4         893.0      -161.0       258.0     -1210.0
## 5         158.0      -223.0       -46.0      -294.0
## 6         716.0      -182.0      -144.0      -553.0
```

Como podemos ver, en este dataframe disponemos de 16067 variables: una de ellas es *tipo_muestra*, que indica si la muestra es efectivamente un tumor o no, otras dos variables indican el tipo y subtipo de tumor, y el resto son variables que describen la expresión de los genes en cada muestra.

La tecnología utilizada para cuantificar la expresión de los genes tiene ciertos límites de detección: cuando los valores son inferiores a 20 unidades, se considera ruido y debe interpretarse como que no hay expresión de ese gen; y, además, el detector empleado se satura en las 16000 unidades, de manera que este es el valor máximo. Debemos, por lo tanto, sustituir los valores inferiores a 20 por 0 y los superiores a 16000 por 16000.

Además, como nos enfocaremos en la clasificación por tipo de tumor, nos restringimos a aquellas observaciones provenientes de tumores (descartando las "normales"), y suprimimos la columna de sub-tipos.

```
remplazo <- function(x){
  x[x < 20] <- 0
  x[x > 16000] <- 16000
  return(x)
}

datos <- map_at(datos, .at = 5:ncol(datos), .f = remplazo) %>% as_tibble()
attach(datos)
datos <- datos[tipo_muestra == "Tumor",]  #Solo interesan los tumores
datos <- datos[,-c(2, 4)] #descartamos los subtipos
```

Además, notamos que para cada muestra disponemos de la información correspondiente a cada tipo de gen:

```
# No hay valores ausentes
na_por_columna <- map_dbl(.x = datos, .f = function(x){sum(is.na(x))})
any(na_por_columna > 0)
```

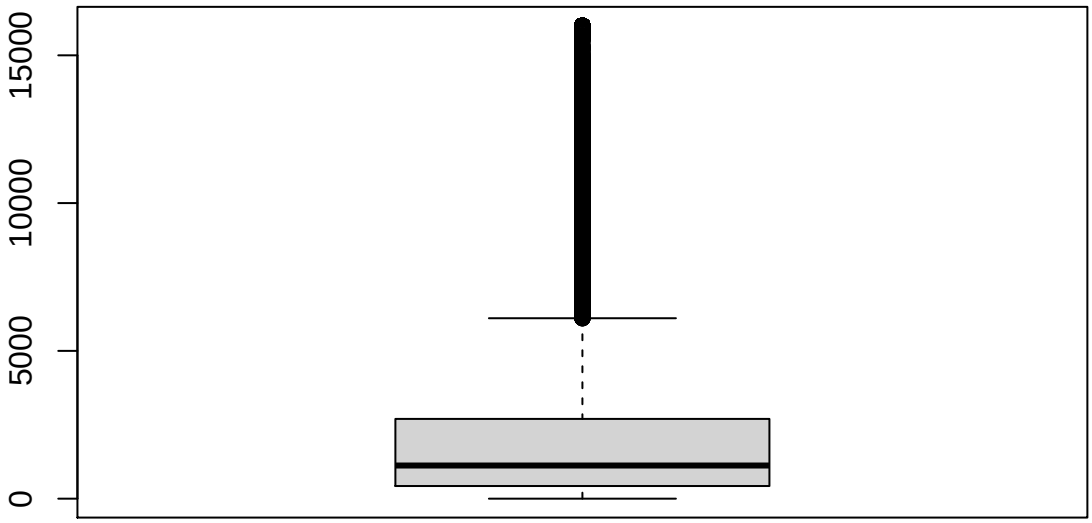
```
## [1] FALSE
```

De las 16067, queremos descartar aquellas que sólo aportan ruido a la hora de encontrar patrones entre la expresion genética y el tipo de tumor. Para eso, analizamos el rango de cada variable, como medida de dispersión:

```
rango<-function(x){max(x)-min(x)}
rango<-apply(datos[3: dim(datos)[2]], 2, rango )
summary(rango)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0     427    1121    2290    2698   16000
```

```
boxplot(rango)
```



Notamos que hay muchas columnas que valen 0 casi siempre. Analizamos la varianza de las variables:

```
desvios<-apply(datos[3: dim(datos)[2]], 2, sd)
summary(desvios)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   64.53  163.12  345.63  379.29 6951.02
```

Entendiendo que las columnas con poca variabilidad aportan cierto ruido a la hora de clasificar, vamos a descartar todas aquellas columnas cuyo rango no supere las 500 unidades, y procedemos a descartar también aquellas cuya varianza es nula.

```
# Devuelve TRUE para: las columnas no numéricas y las columnas numéricas que superen
# las condiciones de variabilidad mencionadas.
filtrado_variabilidad <- function(x){
  if(is.numeric(x)){
    rango <- max(x) - min(x)
    return(rango >= 500 & sd(x)!=0)
  }else{
    return(TRUE)
  }
```

```
}

# Las columnas que cumplen la condición son:
genes_variabilidad <- map_lgl(.x = datos, .f = filtrado_variabilidad)


# Cantidad de columnas que excluiremos:
sum(genes_variabilidad == FALSE)
```

```
## [1] 4572
```

```
datos<-datos[, genes_variabilidad]
```

Separamos, como siempre, los datos en subconjuntos de entrenamiento y de testeo.

```
set.seed(1234)
indices <- indices<-sort(sample(1:nrow(datos), nrow(datos)*0.8, replace = FALSE))
datos_train<-datos[indices, ]
datos_test<-datos[-indices, ]
```

También normalizamos los datos para que tengan media 0 y varianza 1.

```
transformacion<- preProcess(x = datos_train, method = c("center", "scale"))
datos_train <- predict(transformacion, datos_train)
datos_test <- predict(transformacion, datos_test)
```

Selección de genes y reducción de dimensionalidad

En lo que sigue, vamos a reducir la cantidad de variables explicativas de dos formas distintas.

Primero, aplicaremos el metódo **PCA** como técnica para reducir la dimension del espacio, y nos quedaremos con un nuevo dataset cuyas columnas contengan la misma información que las que tenemos ahora, para luego modelar un clasificador utilizando redes neuronales.

Por otro lado, más adelante, vamos a reducir la cantidad de columnas del actual set de datos filtrando mediante el método **signal to noise**, para modelar un clasificador de tumores utilizando un subset de genes que estén realmente relacionados con la variable respuesta (los genes que se expresan de forma diferente en una clase de tumor respecto al resto de clases).

Finalmente compararemos el resultado obtenido por ambos métodos de filtrado de los datos.

Reducción de la dimensión por método PCA

```
#Se mantienen las componentes principales hasta alcanzar un 95% de varianza explicada.
transformacion_pca <- preProcess(x = datos_train, method = "pca", thresh = 0.95)

datos_train_pca<- predict(transformacion_pca, datos_train)
datos_test_pca<- predict(transformacion_pca, datos_test)

dim(datos_train_pca)
```

```
## [1] 152  78
```

```
head(datos_train_pca)[1:10]
```

```
##   tipo_tumor       PC1        PC2       PC3        PC4       PC5         PC6
## 1    Bladder -60.01258 -28.599552  0.730376 -0.4073338  3.082906 -13.5972662
## 2    Bladder -58.49572 -23.464962  7.332198  5.6364848  5.538927  -8.6572861
## 3    Bladder -68.02109 -26.864010  3.902865  3.2600772  5.145490 -15.7101909
## 4    Bladder -30.74848  -4.724626 27.106223 21.1841299 12.810518  21.0773962
## 5    Bladder -51.75156 -17.262546 10.424521  8.8754481  6.913420  -0.5981443
## 6    Bladder -37.83066 -15.182682 12.410466  9.4917823  6.001172  10.5985095
##        PC7       PC8        PC9
## 1 1.994886 -7.637564  0.9452637
## 2 1.866593 -8.326517 -0.9763972
## 3 1.303319 -7.850482  0.7029802
## 4 1.948144 -7.747544 -8.2122354
## 5 1.167705 -8.294949 -2.6468850
## 6 1.430570 -5.699814 -2.9018241
```

Para definir los modelos usaremos la funcionalidad *train*, que sirve para ajustar distintos tipos de modelos evaluando el rendimiento sobre una grilla de parámetros mediante un proceso de *resampling*: elige el modelo óptimo en función de los parámetros y estima su performance con alguna metrica apropiada.

```
reps_boot <- 25

# Parámetros: creamos un data frame con todas las combinaciones posibles de size y decay
#(parámentros necesarios en el entrenamiento de una red neuronal)

parametros.pca <- expand.grid(size=seq(from=5, to=20, by=5),
                              decay=seq(from=0.05, to=0.15, by=0.01))


#Fijamos una lista de semillas para bootstrapping
```

```r
set.seed(1234)
semillas.pca <- vector(mode = "list", length = reps_boot + 1)
for (i in 1:reps_boot) {
  semillas.pca[[i]] <- sample(1000, nrow(parametros.pca))
}
semillas.pca[[reps_boot + 1]] <- sample(1000, 1)


control.pca <- trainControl(method = "boot", number = reps_boot,
                            seeds = semillas.pca, returnResamp = "final",
                            verboseIter = FALSE, allowParallel = TRUE)




#ajustamos el modelo

set.seed(2345)
nnet_pca<- train(tipo_tumor~.,datos_train_pca, method = "nnet",
  tunedGrid = parametros.pca,
  metric = "Accuracy",
  trControl = control.pca,
  rang =0.6,
  MaxNWts = 10000,
  trace=FALSE
)
nnet_pca
```

```
## Neural Network
##
## 152 samples
##  77 predictor
##  14 classes: 'Bladder', 'Breast', 'CNS', 'Colorectal', 'Leukemia', 'Lung', 'Lymphoma', 'Melanoma', '
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 152, 152, 152, 152, 152, 152, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   1     0e+00  0.1941419  0.1101830
##   1     1e-04  0.2004160  0.1182981
##   1     1e-01  0.2027239  0.1183785
##   3     0e+00  0.3214783  0.2609248
##   3     1e-04  0.3059050  0.2463838
##   3     1e-01  0.3532982  0.2987818
##   5     0e+00  0.3999905  0.3470704
##   5     1e-04  0.4082282  0.3573671
##   5     1e-01  0.4258250  0.3760204
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 5 and decay = 0.1.
```

El modelo más exacto se obtiene con los parámentros size=5 y decay=0.1.

Filtrado de variables por método S2N

Filtramos ahora por signal to noise (S2N). Para un determinado grupo (tipo de tumor), los genes con un valor alto de S2N son buenos representantes del mismo.

La idea es obtener el valor de este estadístico para cada uno de los genes en cada uno de los tipos de tumor, y guardar esos resultados en una lista; luego seleccionaríamos los 10 genes con mayor valor absoluto en cada uno de los grupos. Pero para evitar que esa selección de genes por grupo esté demasiado influenciada por la muestra, acudimos a un proceso de bootstraping para acercarnos un poco más a la distribución de la que esta proviene.

Por cada muestra obtenida por bootstrapping obtenemos el valor S2N para cada uno de los genes, en cada uno de los tipos de tumor, y guardamos esos resultados en una lista; calculamos la *media* de los valores de S2N por gen y seleccionamos los 10 de mayor valor absoluto por grupo.

```
# Número de iteraciones bootstrapping. Pocas, para que no tome demasiado tiempo.
n_boot <- 4

# Semillas para los muestreos
set.seed(1234)
semillas <- sample.int(1000, size = n_boot)

#Creamos una lista donde almacenar los resultados S2N, por muestra de bootstrapping
resultados_s2n<-vector(mode = "list", length = n_boot)


for (i in 1:n_boot) {
  set.seed(semillas[i])
  indices <- sample(1:nrow(datos_train),
                    size = nrow(datos_train),
                    replace = TRUE)
  pseudo_muestra <- datos_train[indices, ]

  # Guardamos los nombres de los distintos grupos (tipos de tumor)
  grupos <- unique(pseudo_muestra$tipo_tumor)

  # Creamos la lista donde guardaremos los resultados para cada grupo
  s2n_por_grupo <- vector(mode = "list", length = length(grupos))
  names(s2n_por_grupo) <- grupos

  # Se calcula el valor S2N de cada gen en cada grupo
  for (grupo in grupos){
    # Media y desviación de cada gen en el grupo
    datos_grupo  <-  pseudo_muestra[pseudo_muestra$tipo_tumor == grupo,
                                    3:ncol(pseudo_muestra)]
    medias_grupo <- map_dbl(datos_grupo, .f = mean)
    sd_grupo     <- map_dbl(datos_grupo, .f = sd)

    # Media y desviación de cada gen en el resto de grupos
    datos_otros  <- pseudo_muestra[pseudo_muestra$tipo_tumor != grupo,
                                   3:ncol(pseudo_muestra)]
    medias_otros <- map_dbl(datos_otros, .f = mean)
    sd_otros     <- map_dbl(datos_otros, .f = sd)

    # S2N
    s2n <- (medias_grupo - medias_otros)/(sd_grupo + sd_otros)
```

```
    s2n_por_grupo[[grupo]] <- s2n
  }

  resultados_s2n[[i]]<-s2n_por_grupo

}

names(resultados_s2n) <- paste("resample", 1:n_boot, sep = "_")
```

```
#Creamos un dataframe con todos los resultados y los agrupamos por tipo de tumor.
#Hacemos las modificaciones paso por paso:

#junto todo en una lista
resultados_s2n_grouped <- unlist(resultados_s2n)

#miro la lista como data frame
resultados_s2n_grouped<-as.data.frame(resultados_s2n_grouped)

resultados_s2n_grouped<-rownames_to_column(resultados_s2n_grouped, var = "id")

#separo la columna de nombres en muchas columnas por tipo de tumor, gen, etc
resultados_s2n_grouped<-separate(resultados_s2n_grouped,
                                 col = id, sep = "[.]", remove = TRUE,
                                 into = c("resample", "tipo_tumor", "gen"),
                                 convert = TRUE)

#renombro la ultima columna
resultados_s2n_grouped<-rename(resultados_s2n_grouped,
                               s2n = resultados_s2n_grouped)

#un data frame por cada tipo de tumor, todos dispuestos en una lista
resultados_s2n_grouped<-nest(group_by(resultados_s2n_grouped, tipo_tumor))


extraer_top_genes <- function(data, n=10){
  #agrego una col aux para que spread no de error si hay filas identicas
  data$aux<-1:nrow(data)

  data <- spread(data, key = "resample", value = s2n)
  data$s2n_medio<-abs(rowMeans(data[, c("resample_1", "resample_2",
                                        "resample_3", "resample_4")],
                               na.rm = TRUE))
  data<-data[order(data$s2n_medio,decreasing = FALSE),]
  top_genes <- head(data$gen, n)
  return(as.character(top_genes))
}

aux<-list()
for (k in 1:nrow(resultados_s2n_grouped)) {
  data<-resultados_s2n_grouped[[2]][[k]]
  aux[[k]]<- extraer_top_genes(data)
}
resultados_s2n_grouped$genes<-aux
```

```
resultados_s2n_grouped
```

```
## # A tibble: 14 x 3
## # Groups:   tipo_tumor [14]
##    tipo_tumor   data                  genes
##    <chr>        <list>                <list>
##  1 Prostate     <tibble [45,964 x 3]> <chr [10]>
##  2 CNS          <tibble [45,964 x 3]> <chr [10]>
##  3 Lymphoma     <tibble [45,964 x 3]> <chr [10]>
##  4 Uterus       <tibble [45,964 x 3]> <chr [10]>
##  5 Melanoma     <tibble [45,964 x 3]> <chr [10]>
##  6 Ovary        <tibble [45,964 x 3]> <chr [10]>
##  7 Bladder      <tibble [45,964 x 3]> <chr [10]>
##  8 Colorectal   <tibble [45,964 x 3]> <chr [10]>
##  9 Renal        <tibble [45,964 x 3]> <chr [10]>
## 10 Breast       <tibble [45,964 x 3]> <chr [10]>
## 11 Pancreas     <tibble [45,964 x 3]> <chr [10]>
## 12 Lung         <tibble [45,964 x 3]> <chr [10]>
## 13 Mesothelioma <tibble [45,964 x 3]> <chr [10]>
## 14 Leukemia     <tibble [45,964 x 3]> <chr [10]>
```

Para quedarnos, finalmente, con un data frame habiendo filtrado las columnas más relevantes, queremos indentificar antes los genes que se repiten por tipo de tumor, para no tener columnas repetidas.

```
genes_s2n <-unlist(pull(resultados_s2n_grouped,genes))

#no hay genes repetidos
filter(as.data.frame(table(genes_s2n)), Freq > 1)
```

```
## [1] genes_s2n Freq
## <0 rows> (or 0-length row.names)
```

El set de datos obtenido es

```
datos_train_s2n<-select(datos_train, c("tipo_tumor", all_of(genes_s2n)))
```

con dimensión

```
dim(datos_train_s2n)
```

```
## [1] 152 141
```

Definimos el modelo usando este último data frame:

```
parametros.s2n <- expand.grid(size = c(10, 15, 20), decay = c(0.01, 0.1))


reps_boot<-25


set.seed(1234)
semillas.s2n <- vector(mode = "list", length = reps_boot + 1)
for (i in 1:reps_boot) {
  semillas.s2n[[i]] <- sample(1000, nrow(parametros.s2n))
}
semillas.s2n[[reps_boot + 1]] <- sample(1000, 1)
```

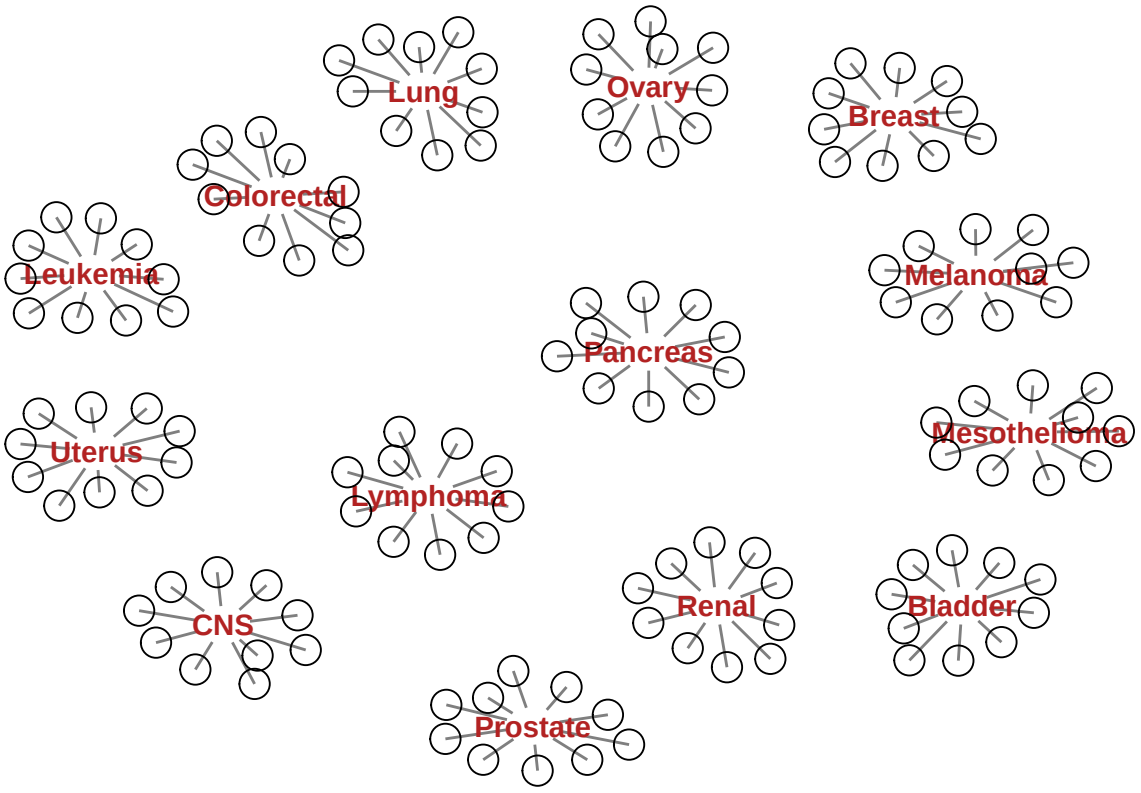
```r
control.s2n <- trainControl(method = "boot", number = reps_boot,
                            seeds = semillas.s2n, returnResamp = "final",
                            verboseIter = FALSE, allowParallel = TRUE)



set.seed(2345)
nnet_s2n<- train(
    tipo_tumor ~ .,
    datos_train_s2n,
    method = "nnet",
    tuneGrid = parametros.s2n,
    metric = "Accuracy",
    trControl = control.s2n,
    # Rango de inicialización de los pesos
    rang = 0.7,
    # Número máximo de pesos
    MaxNWts = 10000,
    # Para que no se muestre cada iteración por pantalla
    trace = FALSE
)
nnet_s2n
```

```
## Neural Network
##
## 152 samples
## 140 predictors
##  14 classes: 'Bladder', 'Breast', 'CNS', 'Colorectal', 'Leukemia', 'Lung', 'Lymphoma', 'Melanoma', '
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 152, 152, 152, 152, 152, 152, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##   10    0.01   0.4927105  0.4488440
##   10    0.10   0.5011751  0.4574101
##   15    0.01   0.5146086  0.4723808
##   15    0.10   0.5374998  0.4967865
##   20    0.01   0.5211749  0.4794746
##   20    0.10   0.5420633  0.5019414
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 20 and decay = 0.1.
```

Antes de pasar a la comparación de los modelos, mostramos los grupos de genes característicos de cada tipo de tumor en el siguiente gráfico.

```r
datos_graph <- resultados_s2n_grouped %>% select(genes, tipo_tumor) %>% unnest()

# Se convierte el dataframe en un objeto igraph
datos_graph <- graph.data.frame(d = datos_graph, directed = TRUE)
# Se convierte el objeto igraph en un objeto tipo ggnetwork
datos_graph_tidy <- ggnetwork(datos_graph)
```

```r
# Se convierte en un dataframe estándar
datos_graph_tidy <- datos_graph_tidy %>% map_df(as.vector)
datos_graph_tidy <- distinct(datos_graph_tidy)

# Se separan los nodos que representan los tipos de tumores de los nodos que
# representan genes

datos_graph_tidy_tumores <- datos_graph_tidy %>%
  filter(name %in% c(unique(datos_train$tipo_tumor)) &
         x == xend)

datos_graph_tidy_genes <- datos_graph_tidy %>%
  filter(!name %in% c(unique(datos_train$tipo_tumor)))

# Se crea el gráfico
ggplot(datos_graph_tidy, aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_edges(color = "grey50") +
  geom_nodetext(data = datos_graph_tidy_tumores, aes(label = name),
                size = 4, color = "firebrick", fontface = "bold") +
  geom_nodes(data = datos_graph_tidy_genes, size = 5, shape = 1) +
  #geom_nodetext(data = datos_graph_tidy_genes, aes(label = vertex.names),
  #              size = 2) +
  theme_blank()
```



Comparación de modelos

En lo que sigue, utilizamos los datos de testeo para comparar los modelos nnet_pca y nnet_s2n mediante el cálculo del error, y mostramos la evolución de la exactitud (o accuracy) en cada uno de ellos.

Error de validación

```
modelos <- list(
  NNET_pca      = nnet_pca,
  NNET_s2n      = nnet_s2n
  )
resultados_resamples <- resamples(modelos)

# Se trasforma el dataframe devuelto por resamples() para separar el nombre del
# modelo y las métricas en columnas distintas.
metricas_resamples <- resultados_resamples$values %>%
                        gather(key = "modelo", value = "valor", -Resample) %>%
                        separate(col = "modelo", into = c("modelo", "metrica"),
                                 sep = "~", remove = TRUE)
# Accuracy y Kappa promedio de cada modelo
promedio_metricas_resamples <- metricas_resamples %>%
  group_by(modelo, metrica) %>%
  summarise(media = mean(valor)) %>%
  spread(key = metrica, value = media) %>%
  arrange(desc(Accuracy))
```

```
## `summarise()` regrouping output by 'modelo' (override with `.groups` argument)
```

```
promedio_metricas_resamples
```

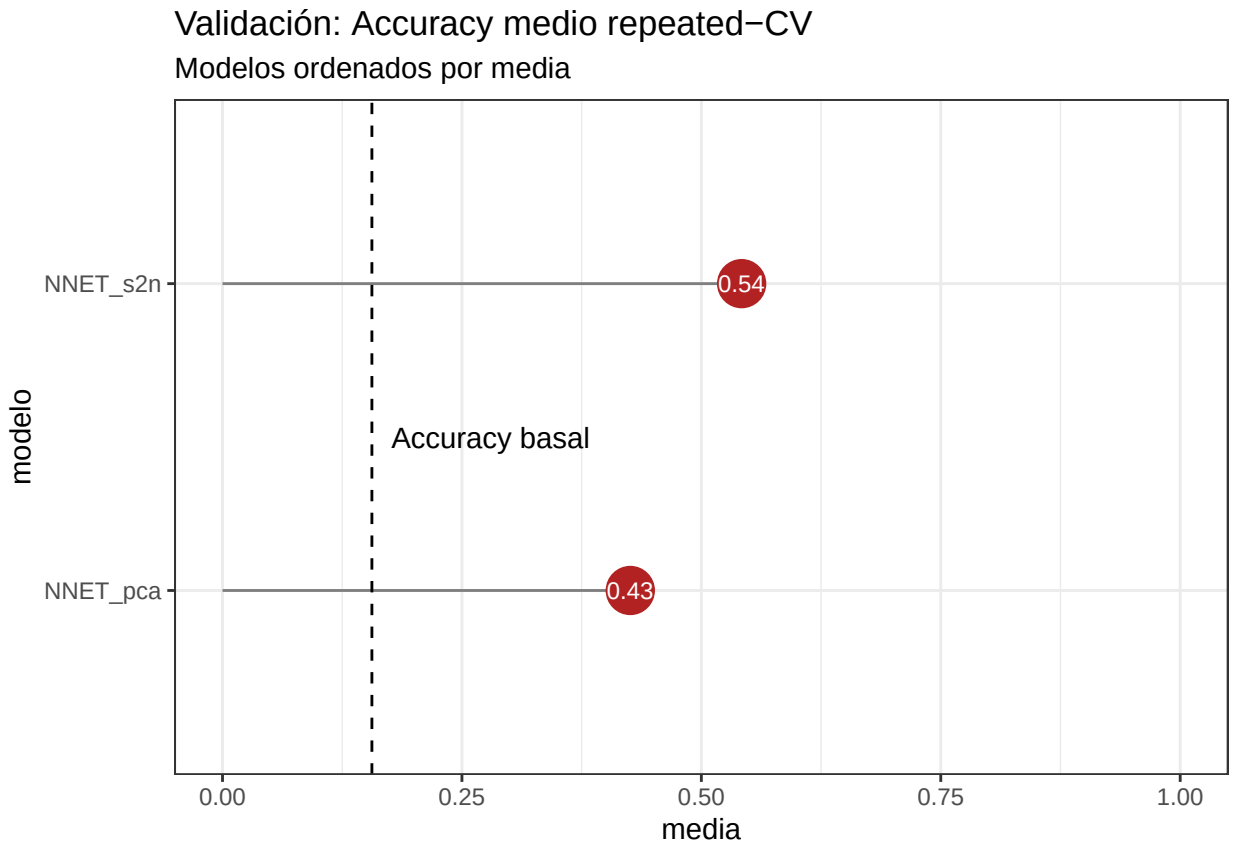
```
## # A tibble: 2 x 3
## # Groups:   modelo [2]
##   modelo   Accuracy Kappa
##   <chr>       <dbl> <dbl>
## 1 NNET_s2n    0.542 0.502
## 2 NNET_pca    0.426 0.376
```

Veamos las comparaciones de Accuracy medio en los plots siguientes.

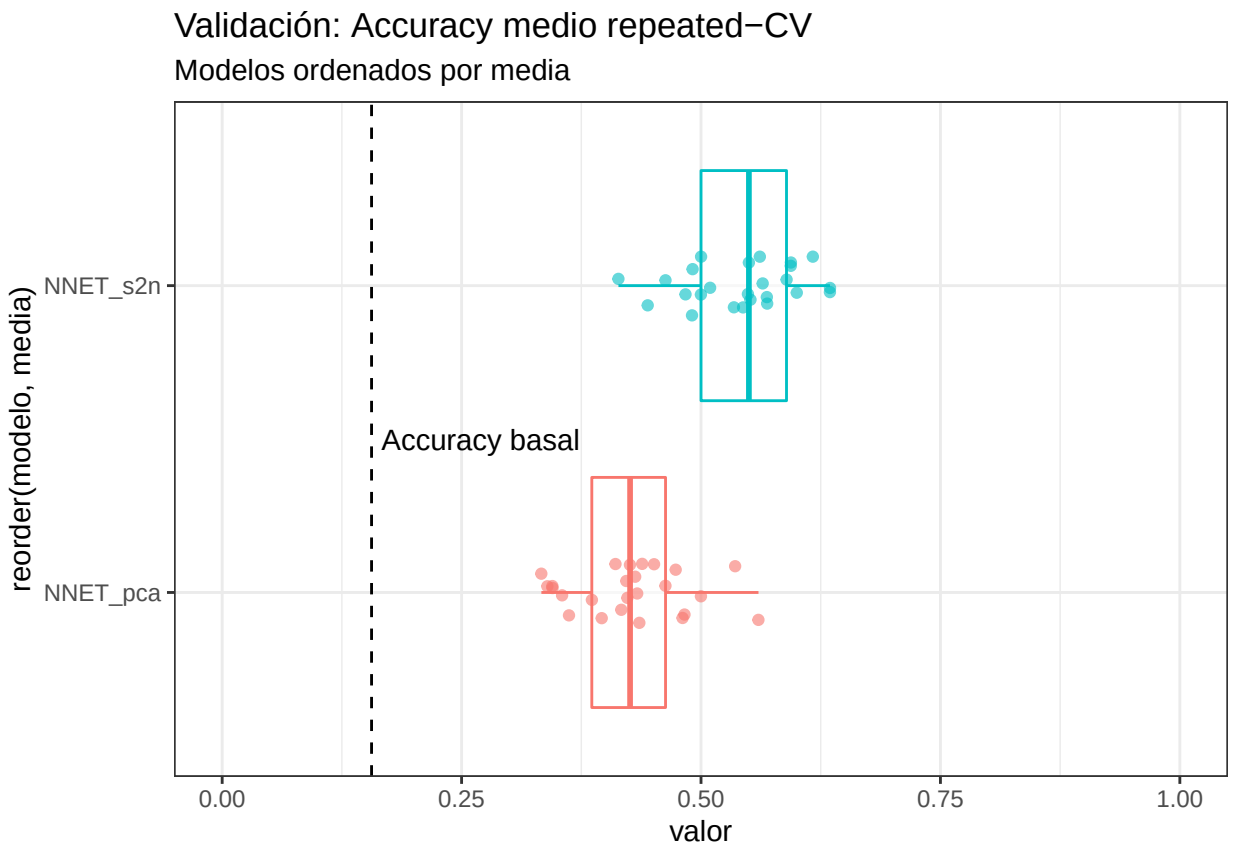
Ploteamos información de los resamples de cada modelo.

```
metricas_resamples %>%
  filter(metrica == "Accuracy") %>%
  group_by(modelo) %>%
  summarise(media = mean(valor)) %>%
  ggplot(aes(x = reorder(modelo, media), y = media, label = round(media, 2))) +
    geom_segment(aes(x = reorder(modelo, media), y = 0,
                     xend = modelo, yend = media),
                     color = "grey50") +
    geom_point(size = 8, color = "firebrick") +
    geom_text(color = "white", size = 3) +
    scale_y_continuous(limits = c(0, 1)) +
    # Accuracy basal
    geom_hline(yintercept = 0.156, linetype = "dashed") +
    annotate(geom = "text", y = 0.28, x = 1.5, label = "Accuracy basal") +
    labs(title = "Validación: Accuracy medio repeated-CV",
         subtitle = "Modelos ordenados por media",
         x = "modelo") +
    coord_flip() +
    theme_bw()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
metricas_resamples %>% filter(metrica == "Accuracy") %>%
  group_by(modelo) %>%
  mutate(media = mean(valor)) %>%
  ungroup() %>%
  ggplot(aes(x = reorder(modelo, media), y = valor, color = modelo)) +
    geom_boxplot(alpha = 0.6, outlier.shape = NA) +
    geom_jitter(width = 0.1, alpha = 0.6) +
    scale_y_continuous(limits = c(0, 1)) +
    # Accuracy basal
    geom_hline(yintercept = 0.156, linetype = "dashed") +
    annotate(geom = "text", y = 0.27, x = 1.5, label = "Accuracy basal") +
    theme_bw() +
    labs(title = "Validación: Accuracy medio repeated-CV",
         subtitle = "Modelos ordenados por media") +
    coord_flip() +
    theme(legend.position = "none")
```



En términos de Accuracy el mejor modelo es **nnet_s2n**.

Error de test

```
prediccion_nnet_pca<-predict(nnet_pca, datos_test_pca)

prediccion_nnet_s2n<-predict(nnet_s2n, datos_test)

head(data.frame(prediccion_nnet_pca, prediccion_nnet_s2n))
```

```
##   prediccion_nnet_pca prediccion_nnet_s2n
## 1             Bladder             Bladder
## 2            Prostate              Uterus
## 3             Bladder                Lung
## 4               Ovary                Lung
## 5              Breast            Melanoma
## 6            Melanoma              Breast
```

```
prediccion_nnet_pca<-factor(prediccion_nnet_pca, levels=unique(datos$tipo_tumor))
prediccion_nnet_s2n<-factor(prediccion_nnet_s2n, levels=unique(datos$tipo_tumor))
datos_test$tipo_tumor<-factor(datos_test$tipo_tumor, levels=unique(datos$tipo_tumor))
```

```
Error_pca<-mean(prediccion_nnet_pca != datos_test_pca$tipo_tumor)
Error_s2n<-mean(prediccion_nnet_s2n != datos_test$tipo_tumor)

Error_pca
```

```
## [1] 0.3421053
```

```
Error_s2n
```

```
## [1] 0.4210526
```

En términos del error obtenido para cada modelo el mejor es **nnet_pca**.

Veamos esta situación gráficamente.

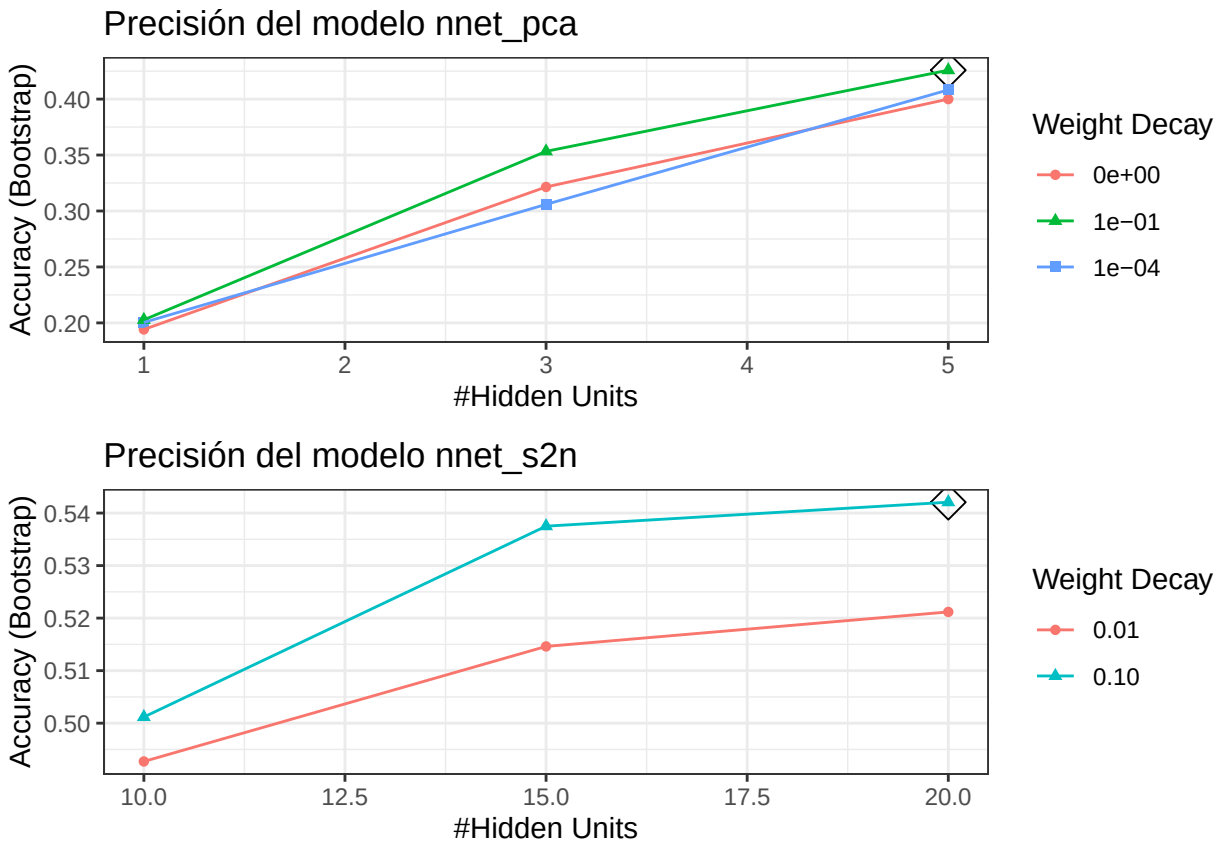
```
p1=ggplot(nnet_pca,highlight = TRUE) +
  labs(title = "Precisión del modelo nnet_pca") +
  theme_bw()
p2=ggplot(nnet_s2n, highlight = TRUE) +
  labs(title = "Precisión del modelo nnet_s2n") +
  theme_bw()
require(gridExtra)
```

```
## Loading required package: gridExtra
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

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



Clustering

Una de las premisas en la que se basa el análisis realizado es la idea de que, tumores de distintos tejidos, tienen un perfil de expresión genética distinto y, por lo tanto, puede emplearse esta información para clasificarlos. Una forma de explorar si esto es cierto, es mediante el uso de técnicas de aprendizaje no supervisado, en concreto el clustering.

Vamos a organizar los tipos de tumores a través de un dendograma. Primero una breve descripción.

Un dendrograma es un tipo de representación gráfica o diagrama de datos en forma de árbol que organiza los datos en subcategorías que se van dividiendo en otros hasta llegar al nivel de detalle deseado (asemejándose a las ramas de un árbol que se van dividiendo en otras sucesivamente).

Este tipo de representación permite apreciar claramente las relaciones de agrupación entre los datos e incluso entre grupos de ellos aunque no las relaciones de similitud o cercanía entre categorías.

```
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
# Se unen de nuevo todos los datos en un único dataframe
datos_clustering <- bind_rows(datos_train, datos_test)
datos_clustering <- datos_clustering %>% arrange(tipo_tumor)

# La librería factoextra emplea el nombre de las filas del dataframe para
# identificar cada observación.
datos_clustering <- datos_clustering %>% as.data.frame()
rownames(datos_clustering) <- paste(1:nrow(datos_clustering),
                                    datos_clustering$tipo_tumor,
                                    sep = "_")

# Se emplean únicamente los genes filtrados
datos_clustering <- datos_clustering %>% select(tipo_tumor, all_of(genes_s2n))

# Se calculan las distancias en base a la correlación de Pearson
mat_distancias <- get_dist(datos_clustering[, -1],
                           method = "pearson",
                           stand = FALSE)
```

```
library(cluster)

# HIERARCHICAL CLUSTERING
# ==============================================================================
set.seed(101)
hc_average <- hclust(d = mat_distancias, method = "complete")

# VISUALIZACIÓN DEL DENDROGRAMA
# ==============================================================================

# Vector de colores para cada observacion: Se juntan dos paletas para tener
# suficientes colores
library(RColorBrewer)
colores <- c(brewer.pal(n = 8, name = "Dark2"),
             brewer.pal(n = 8, name = "Set1")) %>%
          unique()
# Se seleccionan 14 colores, uno para cada tipo de tumor
colores <- colores[1:14]
```

```r
# Se asigna a cada tipo de tumor uno de los colores. Para conseguirlo de forma
# rápida, se convierte la variable tipo_tumor en factor y se emplea su codificación
# numérica interna para asignar los colores.
colores <- colores[as.factor(datos_clustering$tipo_tumor)]

# Se reorganiza el vector de colores según el orden en que se han agrupado las
# observaciones en el clustering
colores <- colores[hc_average$order]

fviz_dend(x = hc_average,
          label_cols = colores,
          cex = 0.5,
          lwd = 0.3,
          main = "Linkage completo",
          type = "circular")
```

El algoritmo de clustering no es capaz de diferenciar los 14 grupos de tumores, lo que pone de manifiesto la dificultad de la clasificación. El dendrograma muestra que los tumores de Breast, Blader y Colorectal tienen un perfil de expresión menos definido.