codigo4clases

Amelia Martínez Sequera

codigo4clases

Amelia Martínez Sequera

8/6/2021

2. Librerías

```
## Warning: package 'timevis' was built under R version 4.0.5

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

##
## Attaching package: 'optCluster'

## The following objects are masked from 'package:clValid':
##
##     clusterMethods, clusters, measNames, measures, nClusters,
##     optimalScores

## Loading required package: Biobase

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get,
grep,
```

```
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which.max, which.min

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Setting options('download.file.method.GEOquery'='auto')

## Setting options('GEOquery.inmemory.gpl'=FALSE)

##
## Attaching package: 'rlang'

## The following object is masked from 'package:Biobase':
##
##     exprs

##
## Attaching package: 'limma'

## The following object is masked from 'package:BiocGenerics':
##
##     plotMA

## Loading required package: amap

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:Biobase':
##
##     combine

## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union

## The following object is masked from 'package:kableExtra':
##
##     group_rows

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Loading required package: AnnotationDbi

## Loading required package: stats4

## Loading required package: IRanges

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:tidyr':
##
##     expand

## The following objects are masked from 'package:dplyr':
##
##     first, rename

## The following object is masked from 'package:base':
##
##     expand.grid

##
## Attaching package: 'IRanges'

## The following objects are masked from 'package:dplyr':
##
##     collapse, desc, slice

## The following object is masked from 'package:grDevices':
##
##     windows

##
## Attaching package: 'AnnotationDbi'

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: org.Hs.eg.db

##

##

## Welcome to beadarray version 2.40.0

##
## Attaching package: 'beadarray'
```

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

## The following object is masked from 'package:limma':
##
##     imageplot

##

## Loading required package: marray

##
## Attaching package: 'marray'

## The following object is masked from 'package:edgeR':
##
##     maPlot

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Warning: package 'GenomeInfoDb' was built under R version 4.0.5

## Loading required package: SummarizedExperiment

## Loading required package: MatrixGenerics

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:genefilter':
##
##     rowSds, rowVars

## The following object is masked from 'package:dplyr':
##
##     count

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians

##
## Attaching package: 'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
```

```
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars

## The following objects are masked from 'package:genefilter':
##
##     rowSds, rowVars

## The following object is masked from 'package:Biobase':
##
##     rowMedians

##
## Attaching package: 'purrr'

## The following object is masked from 'package:GenomicRanges':
##
##     reduce

## The following object is masked from 'package:IRanges':
##
##     reduce

## The following objects are masked from 'package:rlang':
##
##     %@%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
##     flatten_lgl, flatten_raw, invoke, list_along, modify, prepend,
##     splice

## Warning: package 'caret' was built under R version 4.0.5

## Loading required package: lattice

##
## Attaching package: 'caret'

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

3. Datos

```
gse<- "GSE136411"
gse_exprs <- getGEO(GEO=gse,  GSEMatrix=TRUE)
```

```
A<-Biobase::pData(gse_exprs$`GSE136411-GPL10558_series_matrix.txt.gz`)
%>% as_tibble()

A<- dplyr::select(A, title, geo_accession,
characteristics_ch1,description)

B<-Biobase::pData(gse_exprs$`GSE136411-GPL6104_series_matrix.txt.gz`) %>%
as_tibble()

B<- dplyr::select(B, title, geo_accession,
characteristics_ch1,description)

(Sampleinf<- rbind(B,A))

getGEOSuppFiles(gse, makeDirectory=T, baseDir="geo_downloads")

GEOquery::gunzip("geo_downloads/GSE136411/GSE136411_Matrix-merged-
normalized-batch.corrected.txt.gz",overwrite = TRUE, remove = FALSE)

GEOquery::gunzip("geo_downloads/GSE136411/GSE136411_Matrix-merged-non-
normalized-raw.txt.gz",overwrite = TRUE, remove = FALSE)
```

3.1. Reestructuración de los datos

Se realizan una serie de modificaciones para almacenar la información en un dataframe en el que cada fila representa una muestra y las columnas contienen la información relativa a ellas (información descriptiva sobre el tipo de EM y la expresión de los genes).

```
datos_raw <- readr::read_delim("geo_downloads/GSE136411/GSE136411_Matrix-
merged-normalized-batch.corrected.txt", delim = "\t", col_names = TRUE,
                    progress = FALSE)

descripcion_genes <- datos_raw[,1]

datos<- data.frame(datos_raw[,-1], row.names= datos_raw$`ID_REF `)

datos<- t(datos)

datos<- cbind(rownames(datos), data.frame(datos, row.names = NULL))
names(datos)[1] = "description"

datos <- dplyr::full_join(Sampleinf, datos, by="description")

library(stringr)

datos$title %>% as.factor() %>% str_replace_all('.*(PBMC_CIS_).*','CIS')
%>% str_replace_all('.*(PBMC_RR_).*','RR')%>%
str_replace_all('.*(PBMC_PP_).*','PP')%>%
```

```
str_replace_all('.*(PBMC_SP_).*','SP')%>%
str_replace_all('.*(PBMC_HC_).*','HC')%>%
str_replace_all('.*(PBMC_OND_).*','OND') -> datos$type

datos$characteristics_ch1 %>% as.factor()%>%
  str_replace_all('.*(disease: multiple sclerosis).*','MS')%>%
  str_replace_all('.*(disease: other neurological disease).*','Other')%>%
  str_replace_all('.*(disease: none).*','health') -> datos$grupo

info_muestras <-datos%>% dplyr::select(description, geo_accession, title,
type, grupo)

datos<-datos%>% dplyr::select(-description, -title, -geo_accession, -
characteristics_ch1, -type, -grupo)

datos<- log2(datos)

datos2<- cbind(info_muestras$description, datos)
names(datos2)[1] <- "muestra"
```

3.2. Filtrado de calidad de datos.

La tecnología microarray empleada para cuantificar los niveles de expresión tiene unos límites de detección, fuera de los cuales, la lectura no es fiable.

Si la señal es inferior a 10 unidades, se considera ruido y debe interpretarse como que no hay expresión de ese gen. En el extremo opuesto, 34 unidades es el valor máximo.

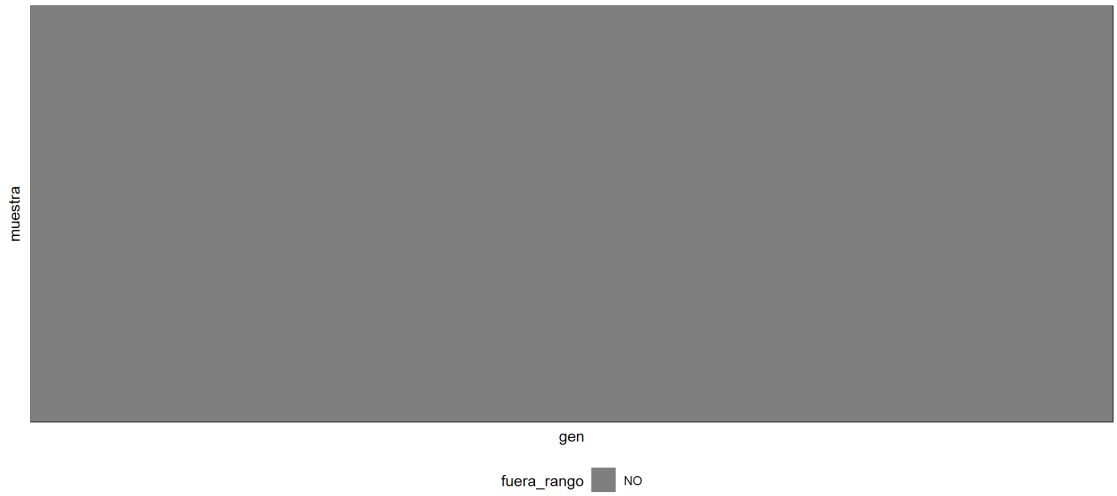
Prácticamente ninguno de los valores de este dataset se corresponden a lecturas por debajo del límite de detección (ruido), porque ya viene corregido.

Es importante remarcar que esta trasformación es un paso de limpieza, no un preprocesado de datos para mejorar el modelo, por lo tanto, sí puede hacerse sobre todas las observaciones antes de separarlas en conjunto de entrenamiento y test.

```
datos_long <- datos2 %>% gather(key= "gen",  value= "expresion", -
muestra) %>%
  mutate(fuera_rango = if_else(expresion< 10| expresion> 34,
                               "SI", "NO"))
# Proporción de valores por debajo del mínimo de detección
nrow(datos_long %>% filter(expresion < 10)) / nrow(datos_long)

## [1] 0

# Proporción de valores por encima del máximo de detección
nrow(datos_long %>% filter(expresion > 34)) / nrow(datos_long)

## [1] 0

# Proporción de valores fuera de rango de detección
nrow(datos_long %>% filter(fuera_rango == "SI")) / nrow(datos_long)
```

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

```
# Representación gráfica de los valores fuera de rango de detección
ggplot(data = datos_long, aes(x = gen, y = muestra, fill = fuera_rango))
+
  geom_raster() +
  scale_fill_manual(values = c("gray50", "orangered2")) +
  theme_bw() +
  theme(legend.position = "bottom",
        axis.text = element_blank(),
        axis.ticks = element_blank())
```



3.4. Exploración de los datos.

3.4.1. Cantidad y tipo de muestras.

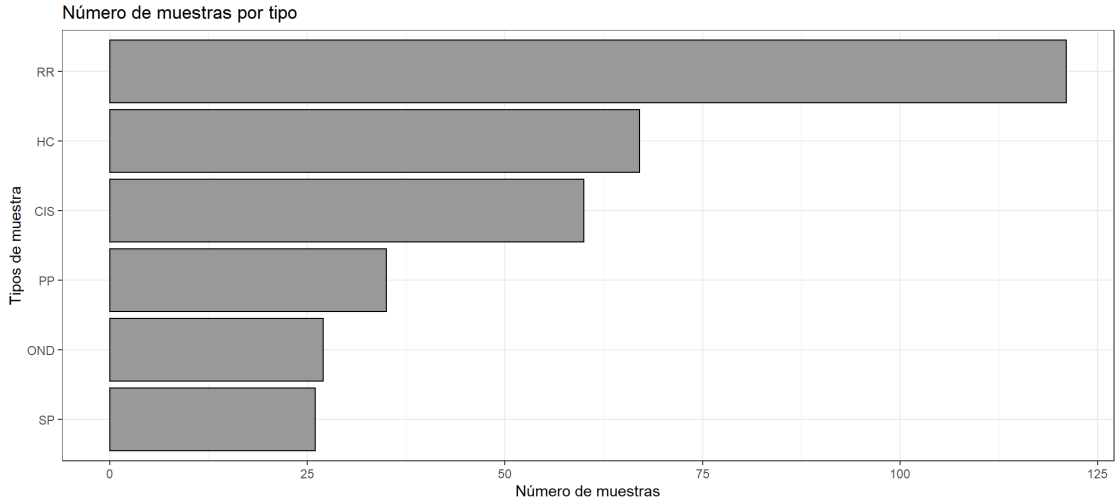
El set de datos se dispone de 336 muestras, agrupadas en 3 grupos, que a su vez se subdividen en tipos: sanos (HC), con EM (CIS,RR, PP, SP) y con otras enfermedades neurológicas (OND).

```
info_muestras %>%
  dplyr::group_by(grupo) %>%
  dplyr::count()
```

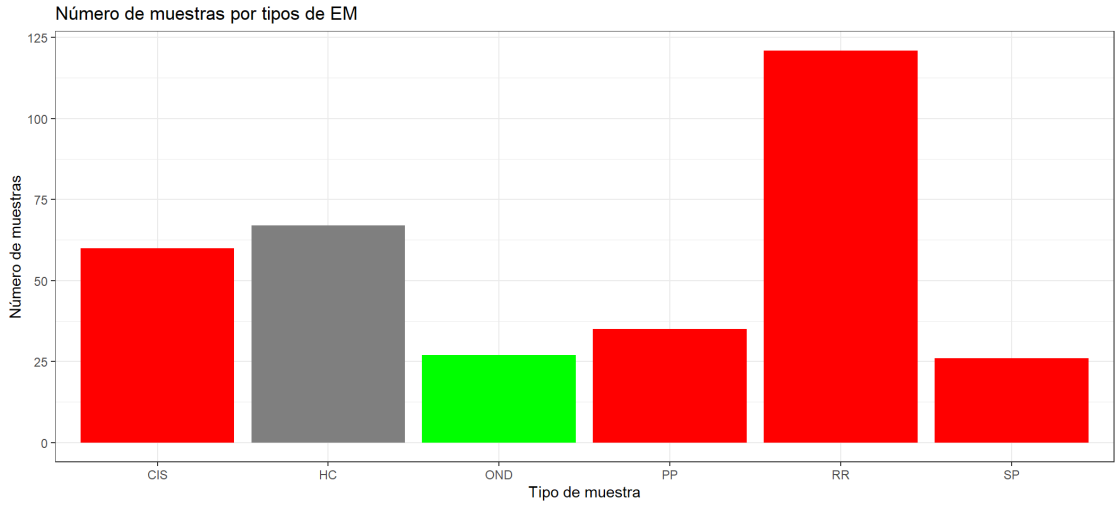
```
info_muestras %>%
  dplyr::filter(grupo == "MS") %>%
  dplyr::group_by(type) %>%
  dplyr::count()
```

```
info_muestras %>%
  dplyr::group_by(type) %>%
  dplyr::count() %>%
  ggplot(aes(x = reorder(type, n), y = n)) +
    geom_col(fill = "gray60", color = "black") +
    coord_flip() +
    theme_bw() +
```

```
    labs(x = "Tipos de muestra", y = "Número de muestras",
         title = "Número de muestras por tipo") +
    theme(legend.position = "bottom")
```




```
info_muestras %>%
  dplyr::group_by(type) %>%
  dplyr::count() %>%
  ggplot(aes(x = type, y = n, fill = type)) +
    geom_col() +
    scale_fill_manual(values = c("red","gray50", "green", "red", "red",
"red")) +
    theme_bw() +
    labs(x = "Tipo de muestra", y = "Número de muestras",
         title = "Número de muestras por tipos de EM") +
    theme(legend.position = "none")
```



3.4.2. Valores ausentes.

```
na_por_columna <- map_dbl(.x = datos, .f = function(x){sum(is.na(x))})
any(na_por_columna > 0)
```

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

Se comprueba que para todas las muestras se dispone del valor de expresión .El set de datos está completo, no hay valores ausentes.

4. División y preprocesado de los datos.

4.1. División de los datos.

```
datost <- cbind(info_muestras$type, datos)
names(datost)[1] <- "type"

datost <- datost %>% dplyr::filter(type== c("CIS", "PP","SP","RR"))

# Se crean los índices de las observaciones de entrenamiento (80%)
set.seed(123)
train <- createDataPartition(y = datost$type, p = 0.8, list = FALSE,
times = 1)
datos_train <- datost[train, ]
datos_test  <- datost[-train, ]
```

Es importante verificar que la distribución de la variable respuesta es similar en el conjunto de entrenamiento y en el de test. Por defecto, la función createDataPartition() garantiza una distribución aproximada (reparto estratificado).

```
distribucion_train <- prop.table(table(datos_train$type)) %>% round(3)
distribucion_test  <- prop.table(table(datos_test$type)) %>% round(3)
data.frame(train = distribucion_train, test = distribucion_test )
```

Aciertos si se emplea la clase mayoritaria como predictor:

```
# Aciertos si se emplea la clase mayoritaria como predictor
mean(datos_train$type == "RR")
```

```
## [1] 0.4897959
```

4.2. Preprocesado.

4.2.1. Genes con varianza próxima a cero.

```
sum(datos_train%>% nearZeroVar(saveMetrics = TRUE) == FALSE)
```

```
## [1] 20322
```

4.2.2. Estandarización.

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

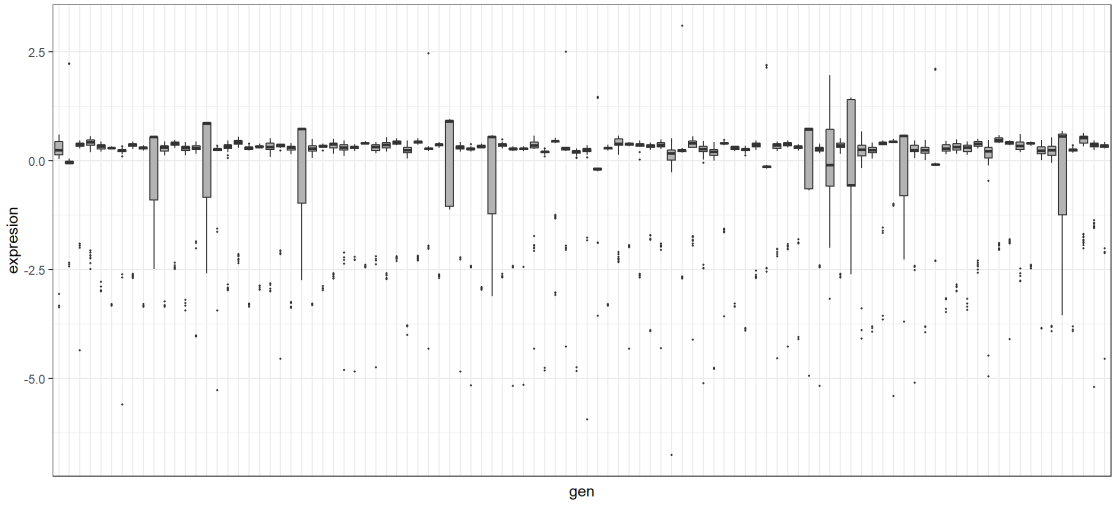
5. Selección de genes y reducción de dimensionalidad:

- Anova p-value
- Signal to noise (S2N)

- Comparación de filtrados
- Reducción de dimensionalidad

5.1. Anova p-value

```
# Representación de la expresión de 100 genes seleccionados de forma
aleatoria.
set.seed(123)
datos_train %>% select_at(sample(4:ncol(datos_train), 100)) %>%
  gather(key = "gen", value = "expresion") %>%
  ggplot(aes(x = gen , y = expresion)) +
    geom_boxplot(outlier.size = 0.3, fill = "gray70") +
    theme_bw() +
    theme(axis.text.x = element_blank(),
          axis.ticks.x = element_blank())
```



Se aplica un análisis ANOVA para cada uno de los genes.

```
custom_anova <- function(x,y){
  anova <- summary(aov(x ~ as.factor(y)))
  return(unlist(anova)["Pr(>F)1"])
}

p_values <- datos_train %>%
            dplyr:: select(-type) %>%
            map_dbl(.f = custom_anova, y = datos_train$type) %>%
            sort()
p_values %>% head(10)
```

```
## ILMN_1805535 ILMN_1695902 ILMN_1704672 ILMN_1696708 ILMN_1653115
ILMN_1724437
## 1.218995e-05 1.302412e-05 2.110580e-05 2.276982e-05 2.359147e-05
4.047832e-05
## ILMN_1677113 ILMN_1684553 ILMN_1746670 ILMN_1709623
## 6.108061e-05 6.896280e-05 7.974954e-05 9.153729e-05
```

.

VERSIÓN NO PARALELIZADA

```
# Se emplea un número de resampling bajo para que no tarde demasiado.
Para valores más elevados emplear la versión paralelizada que se describe
más adelante.

n_boot <- 3
resultados_anova <- vector(mode = "list", length = n_boot)

# Semillas para que los muestreos sean reproducibles
set.seed(123)
seeds = sample.int(1000, size = n_boot)

# Función ANOVA
custom_anova <- function(x,y){
  anova <- summary(aov(x ~ as.factor(y)))
  return(unlist(anova)["Pr(>F)1"])
}

for (i in 1:n_boot){
  # Se crea una muestra bootstrapping
  set.seed(seeds[i])
  indices <- sample(1:nrow(datos_train), size = nrow(datos_train),
replace = TRUE)
  pseudo_muestra <- datos_train[indices, ]

   # Se calculan los p-values con la nueva muestra
  resultados_anova[[i]] <- pseudo_muestra %>%
                           dplyr::select(-type) %>%
                           map_dbl(.f = custom_anova, y =
pseudo_muestra$type)
}

# Los resultados almacenados en forma de lista se convierten en dataframe
names(resultados_anova) <-  paste("resample", 1:n_boot, sep = "_")
resultados_anova <- data.frame(resultados_anova)

resultados_anova<- cbind(rownames(resultados_anova),
data.frame(resultados_anova, row.names = NULL))

names(resultados_anova)[1] = "gen"

resultados_anova <- resultados_anova %>%
                      mutate(pvalue_medio = rowMeans(resultados_anova[, -
1])) %>%
                      arrange(pvalue_medio)
head(resultados_anova)
```

Los resultados almacenados en forma de lista se convierten en dataframe.

Para agilizar el proceso, es recomendable paralelizar el loop externo.

VERSIÓN PARALELIZADA DE BOOTSTRAPPING PARA FILTRADO POR ANOVA

```
library(doParallel)
# Se especifica el número de cores a utilizar (esto depende del ordenador
empleado)
registerDoParallel(cores = 3)
getDoParWorkers()
```

```
## [1] 3
```

```
# Número de iteraciones bootstrapping
n_boot <- 100

# Semillas para que los muestreos sean reproducibles
set.seed(123)
seeds = sample.int(1000, size = n_boot)

# Función ANOVA
custom_anova <- function(x,y){
  anova <- summary(aov(x ~ as.factor(y)))
  return(unlist(anova)["Pr(>F)1"])
}

### LOOP PARALELIZADO
library(parallel)
# La función foreach devuelve los resultados de cada iteración en una
lista

resultados_anova_pvalue <- foreach(i = 1:n_boot) %dopar% {

  require(dplyr)
  require(purrr)

  # Se crea una muestra por bootstrapping
  set.seed(seeds[i])
  indices <- sample(1:nrow(datos_train), size = nrow(datos_train),
replace = TRUE)
  pseudo_muestra <- datos_train[indices, ]


  # Se calculan los p-values para la nueva muestra
  p_values <- pseudo_muestra %>%
    select(-type) %>%
    map_dbl(.f = custom_anova, y = pseudo_muestra$type)


  # Se devuelven los p-value
```

```
  p_values
}

options(cores = 1)

require(dplyr)
require(tidyverse)

## Error in completeSubclasses(classDef2, class1, obj, where) :
##   tentativa de obtener un slot "subclasses" de un objeto de una clase
básica ("NULL") sin slots

# Los resultados almacenados en forma de lista se convierten en dataframe
names(resultados_anova_pvalue) <-  paste("resample", 1:n_boot, sep = "_")

resultados_anova_pvalue <- data.frame(resultados_anova_pvalue)

resultados_anova_pvalue <- resultados_anova_pvalue %>%
tibble::rownames_to_column(var = "gen")

resultados_anova_pvalue <- resultados_anova_pvalue %>%
mutate(pvalue_medio = rowMeans(resultados_anova_pvalue[, -1])) %>%
  arrange(pvalue_medio)

# Se guarda en disco el objeto creado para no tener que repetir de nuevo
toda la computación.
saveRDS(object = resultados_anova_pvalue, file =
"resultados_anova_pvalue.rds")

head(resultados_anova_pvalue)
```

Se guarda en disco el objeto creado para no tener que repetir de nuevo toda la computación: “resultados_anova_pvalue.rds”

```
resultados_anova_pvalue %>% dplyr::select(1,2,3,4) %>% head()
```

Anotación de los genes.

```
ann<- select(illuminaHumanv3.db, keys = resultados_anova_pvalue$gen,
columns=c("ENTREZID","SYMBOL","GENENAME"))

ann[1:25,]

# Se filtran los 100, 50 y 25 genes identificados como más relevantes
mediante anova
filtrado_anova_pvalue_100 <- resultados_anova_pvalue %>% pull(gen) %>%
head(100)
filtrado_anova_pvalue_50  <- resultados_anova_pvalue %>% pull(gen) %>%
head(50)
filtrado_anova_pvalue_25 <-  resultados_anova_pvalue %>% pull(gen) %>%
head(25)
```

5.2. Signal to noise (S2N)

S2N=\(\frac{μ_{grupo i} − μ_{resto de grupos}}{σ_{grupo i} + σ_{resto de grupos}}\)

```
# Se identifica el nombre de los distintos grupos (tipos de tumor)
grupos <- unique(datos_train$type)

# Se crea una lista donde almacenar 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 i
  datos_grupo <- datos_train %>% filter(type == grupo) %>%
dplyr::select(-type)
  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 <- datos_train %>% filter(type != grupo) %>%
dplyr::select(-type)
  medias_otros <- map_dbl(datos_otros, .f = mean)
  sd_otros <- map_dbl(datos_otros, .f = sd)

  # Calculo S2N
  s2n <- (medias_grupo - medias_otros)/(sd_grupo + sd_otros)
  s2n_por_grupo[[grupo]] <- s2n
}

extraer_top_genes <- function(x, n=10, abs=TRUE){
  if (abs == TRUE) {
  x <- abs(x)
  x <- sort(x)
  x <- x[1:n]
  return(names(x))
  }else{
  x <- sort(x)
  x <- x[1:n]
  return(names(x))
  }
}

s2n_por_grupo <- s2n_por_grupo %>% map(.f = extraer_top_genes)

genes_seleccionados_s2n <- unique(unlist(s2n_por_grupo))
length(genes_seleccionados_s2n)

## [1] 40
```

```
annotation<- select(illuminaHumanv3.db, keys = genes_seleccionados_s2n,
columns=c("ENTREZID","SYMBOL","GENENAME"))

annotation[1:25,]
```

Al igual, que en el filtrado por ANOVA, para evitar que la selección este excesivamente influenciada por la muestra de entrenamiento, es conveniente recurrir a un proceso de resampling y agregar los resultados. Esta vez, como método de agregación se emplea la media.

VERSIÓN PARALELIZADA DE BOOTSTRAPPING PARA FILTRADO POR SIGNAL TO NOISE

```
# Warning: Este cálculo puede tardar varias horas.

library(doParallel)
# Se especifica el número de cores a utilizar (esto depende del
ordenador)
registerDoParallel(cores = 3)

# Número de iteraciones bootstrapping
n_boot <- 100

# Semillas para que los muestreos sean reproducibles
set.seed(123)
seeds = sample.int(1000, size = n_boot)

# LOOP PARALELIZADO

resultados_s2n <- foreach(i = 1:n_boot) %dopar% {
  require(purrr)
  require(dplyr)

  # Se crea una nueva muestra por bootstrapping
  set.seed(seeds[i])
  indices <- sample(1:nrow(datos_train),
                    size = nrow(datos_train),
                    replace = TRUE)
  pseudo_muestra <- datos_train[indices, ]

  # Se identifica el nombre de los distintos grupos (tipos de tumor)
  grupos <- unique(pseudo_muestra$type)

  # Se crea una lista donde almacenar 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 i
    datos_grupo  <- pseudo_muestra %>% filter(type == grupo) %>% select(-
```

```
type)
    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 %>% filter(type != grupo) %>% select(-
type)
    medias_otros <- map_dbl(datos_otros, .f = mean)
    sd_otros     <- map_dbl(datos_otros, .f = sd)

    # Calculo S2N
    s2n <- (medias_grupo - medias_otros)/(sd_grupo + sd_otros)
    s2n_por_grupo[[grupo]] <- s2n
  }

  s2n_por_grupo

}
options(cores = 1)

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

# Se guarda en disco el objeto creado
saveRDS(object = resultados_s2n, file = "resultados_s2n.rds")
```

En cada elemento de la lista resultados_s2n se ha almacenado el resultado de una repetición bootstrapping, que a su vez, es otra lista con los valores S2N de cada gen en cada grupo. Para obtener un único listado final por tipo, se tienen que agregar los valores obtenidos en las diferentes repeticiones.

```
require(tidyverse)
```

```
## Error in completeSubclasses(classDef2, class1, obj, where) :
##   tentativa de obtener un slot "subclasses" de un objeto de una clase
básica ("NULL") sin slots
```

```
require(dplyr)
resultados_s2n_grouped <- resultados_s2n %>%
  unlist() %>%
  as.data.frame() %>%
  tibble:: rownames_to_column(var = "id") %>%
  separate(col = id, sep = "[.]",
           remove = TRUE,
           into = c("resample", "type", "gen")) %>%
  dplyr::rename(s2n =".") %>%
  group_by(type) %>%
  nest()

# Para cada tipo  se calcula el s2n medio de los genes y se devuelven los
10 genes con mayor S2N absoluto
```

```
extraer_top_genes <- function(df, n=10){
  df <- df %>% spread(key = "resample", value = s2n)
  df <- df %>% mutate(s2n_medio = abs(rowMeans(df[, -1])))
  top_genes <- df %>% arrange(desc(s2n_medio)) %>% pull(gen) %>% head(n)
  return(as.character(top_genes))
}

resultados_s2n_grouped <- resultados_s2n_grouped %>%
  mutate(gen = map(.x = data, .f = extraer_top_genes))

resultados_s2n_grouped %>%
  head()

saveRDS(object = resultados_s2n_grouped, file =
"resultados_s2n_grouped.rds")
```

Para cada tipo se calcula el s2n medio de los genes y se devuelven los 10 genes con mayor S2N absoluto: “resultados_s2n_grouped.rds”

se identifica la intersección entre los genes seleccionados para cada tipo , y se eliminan aquellos que son comunes para varios estadíos (aparecen más de dos veces): “filtrado_s2n_60.rds”

```
genes_repetidos <-  resultados_s2n_grouped %>%
                    pull(gen) %>%
                    unlist() %>%
                    table() %>%
                    as.data.frame() %>%
                    filter(Freq > 1) %>%
                    pull(".") %>%
                    as.character()


filtrado_s2n_60 <- resultados_s2n_grouped %>%
                    pull(gen) %>%
                    unlist()
filtrado_s2n_60 <- filtrado_s2n_60[!(filtrado_s2n_60 %in%
genes_repetidos)]
saveRDS(object = filtrado_s2n_60, file = "filtrado_s2n_60.rds")
```

El mismo proceso se repite pero seleccionando únicamente los top 5 genes por grupo:“filtrado_s2n_30.rds”

```
extraer_top_genes <- function(df, n=5){
  df <- df %>% spread(key = "resample", value = s2n)
  df <- df %>% mutate(s2n_medio = abs(rowMeans(df[, -1])))
  top_genes <- df %>% arrange(desc(s2n_medio)) %>% pull(gen) %>% head(n)
  return(as.character(top_genes))
}
```

```
resultados_s2n_grouped <- resultados_s2n_grouped %>%
  mutate(gen = map(.x = data, .f = extraer_top_genes))

resultados_s2n_grouped %>%
  head()

saveRDS(object = resultados_s2n_grouped, file =
"resultados_s2n_grouped.rds")

genes_repetidos <-  resultados_s2n_grouped %>%
                    pull(gen) %>%
                    unlist() %>%
                    table() %>%
                    as.data.frame() %>%
                    filter(Freq > 1) %>%
                    pull(".") %>%
                    as.character()


filtrado_s2n_30 <- resultados_s2n_grouped %>%
                    pull(gen) %>%
                    unlist()
filtrado_s2n_30 <- filtrado_s2n_30[!(filtrado_s2n_30 %in%
genes_repetidos)]
saveRDS(object = filtrado_s2n_30, file = "filtrado_s2n_30.rds")
```

5.3. Comparación de filtrados

Se estudia cuantos genes en común se han seleccionado con cada uno de los métodos.

```
length(intersect(filtrado_anova_pvalue_100, filtrado_s2n_60))
```

```
## [1] 16
```

```
length(intersect(filtrado_anova_pvalue_50, filtrado_s2n_30))
```

```
## [1] 7
```

La selección de genes resultante con ambos métodos es muy distinta.

5.4. Reducción de dimensionalidad

Se aplica un PCA a los niveles de expresión y se conservan las componentes principales hasta alcanzar un 95% de varianza explicada.

```
transformacion_pca <- preProcess(x = datos_train, method = "pca", thresh
= 0.95)
transformacion_pca
```

```
## Created from 49 samples and 10161 variables
##
```

```
## Pre-processing:
##   - centered (10160)
##   - ignored (1)
##   - principal component signal extraction (10160)
##   - scaled (10160)
##
## PCA needed 46 components to capture 95 percent of the variance
```

```
datos_train_pca    <- predict(object = transformacion_pca, newdata =
datos_train)
datos_test_pca     <- predict(object = transformacion_pca, newdata =
datos_test)
```

6. Modelos:

- SVM
- RandomForest
- Neural Network

6.1. SVM: Máquinas de Vector Soporte (Support Vector Machines, SVMs)

El método svmRadial de caret emplea la función ksvm() del paquete kernlab. Este algoritmo tiene 2 hiperparámetros:

- sigma: coeficiente del kernel radial.
- C: penalización por violaciones del margen del hiperplano.

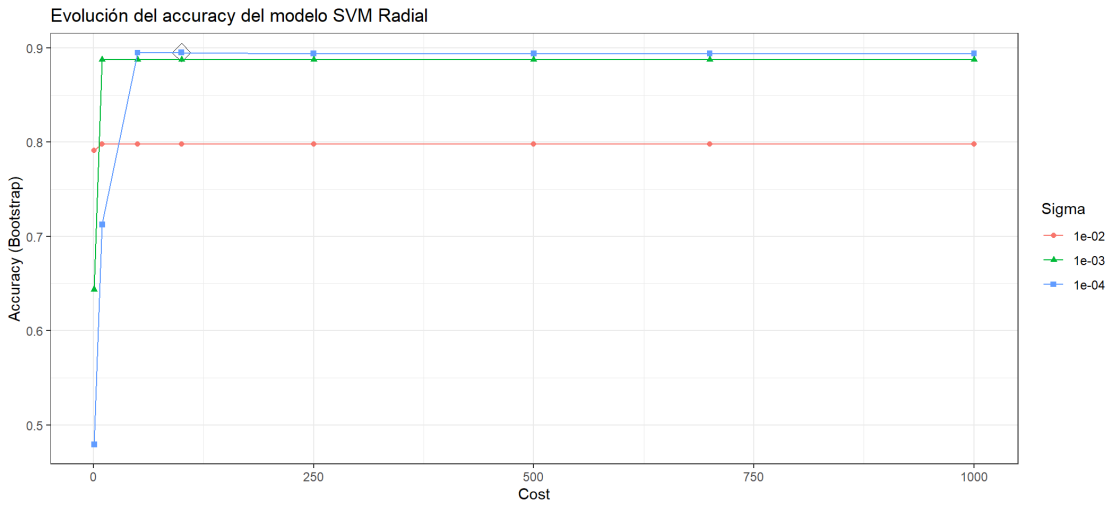
6.1.1. Filtrado por ANOVA p-value 100

```
# PARALELIZACIÓN DE PROCESO
library(doParallel)
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(sigma = c(0.0001, 0.001, 0.01),
                               C = c(1, 10, 50, 100, 250, 500, 700,
1000))
set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
```

```
                            verboseIter = FALSE, allowParallel = TRUE)
# AJUSTE DEL MODELO
set.seed(342)
svmrad_pvalue_100 <- train(
                     form = type ~ .,
                     data = datos_train[c("type",
filtrado_anova_pvalue_100)],
                     method = "svmRadial",
                     tuneGrid = hiperparametros,
                     metric = "Accuracy",
                     trControl = control_train
                    )
registerDoParallel(cores = 1)
saveRDS(object = svmrad_pvalue_100, file = "svmrad_pvalue_100.rds")

svmrad_pvalue_100

## Support Vector Machines with Radial Basis Function Kernel
##
##  49 samples
## 100 predictors
##   4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04     1  0.4791786  0.01329193
##   1e-04    10  0.7125241  0.51300862
##   1e-04    50  0.8948747  0.83663585
##   1e-04   100  0.8950512  0.83718797
##   1e-04   250  0.8938747  0.83537956
##   1e-04   500  0.8938747  0.83537956
##   1e-04   700  0.8938747  0.83537956
##   1e-04  1000  0.8938747  0.83537956
##   1e-03     1  0.6437233  0.39844722
##   1e-03    10  0.8873258  0.82974473
##   1e-03    50  0.8873258  0.82923811
##   1e-03   100  0.8873258  0.82923811
##   1e-03   250  0.8873258  0.82923811
##   1e-03   500  0.8873258  0.82923811
##   1e-03   700  0.8873258  0.82923811
##   1e-03  1000  0.8873258  0.82923811
##   1e-02     1  0.7911679  0.68263479
##   1e-02    10  0.7977563  0.69416438
##   1e-02    50  0.7977563  0.69416438
##   1e-02   100  0.7977563  0.69416438
##   1e-02   250  0.7977563  0.69416438
```

```
##   1e-02   500  0.7977563  0.69416438
##   1e-02   700  0.7977563  0.69416438
##   1e-02  1000  0.7977563  0.69416438
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1e-04 and C = 100.
```

```
ggplot(svmrad_pvalue_100, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo SVM Radial") +
  theme_bw()
```



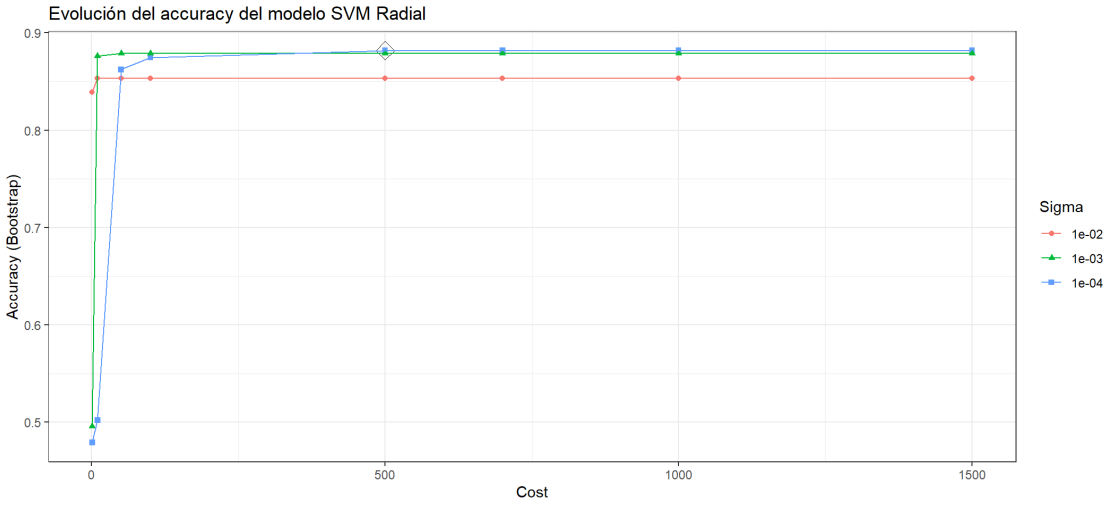
6.1.2. Filtrado por ANOVA p-value 50

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN

repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(sigma = c(0.0001, 0.001, 0.01),
                               C = c(1, 10, 50, 100, 500, 700, 1000,
1500))
set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
```

```
                           verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
svmrad_pvalue_50 <- train(
                      form = type ~ .,
                      data = datos_train[c("type",
filtrado_anova_pvalue_50)],
                      method = "svmRadial",
                      tuneGrid = hiperparametros,
                      metric = "Accuracy",
                      trControl = control_train
                     )
registerDoParallel(cores = 1)
saveRDS(object = svmrad_pvalue_50, file = "svmrad_pvalue_50.rds")

svmrad_pvalue_50

## Support Vector Machines with Radial Basis Function Kernel
##
## 49 samples
## 50 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04     1  0.4791786  0.01329193
##   1e-04    10  0.5021160  0.07645399
##   1e-04    50  0.8623623  0.77153603
##   1e-04   100  0.8748250  0.79363558
##   1e-04   500  0.8819139  0.80780597
##   1e-04   700  0.8819139  0.80780597
##   1e-04  1000  0.8819139  0.80780597
##   1e-04  1500  0.8819139  0.80780597
##   1e-03     1  0.4957089  0.06386795
##   1e-03    10  0.8761419  0.80099625
##   1e-03    50  0.8787958  0.80724572
##   1e-03   100  0.8787958  0.80724572
##   1e-03   500  0.8787958  0.80724572
##   1e-03   700  0.8787958  0.80724572
##   1e-03  1000  0.8787958  0.80724572
##   1e-03  1500  0.8787958  0.80724572
##   1e-02     1  0.8391626  0.74789133
##   1e-02    10  0.8535151  0.77103453
##   1e-02    50  0.8535151  0.77103453
##   1e-02   100  0.8535151  0.77103453
```

```
##   1e-02   500  0.8535151  0.77103453
##   1e-02   700  0.8535151  0.77103453
##   1e-02  1000  0.8535151  0.77103453
##   1e-02  1500  0.8535151  0.77103453
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1e-04 and C = 500.
```

```
ggplot(svmrad_pvalue_50, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo SVM Radial") +
  theme_bw()
```



6.1.3. Filtrado por ANOVA p-value 25

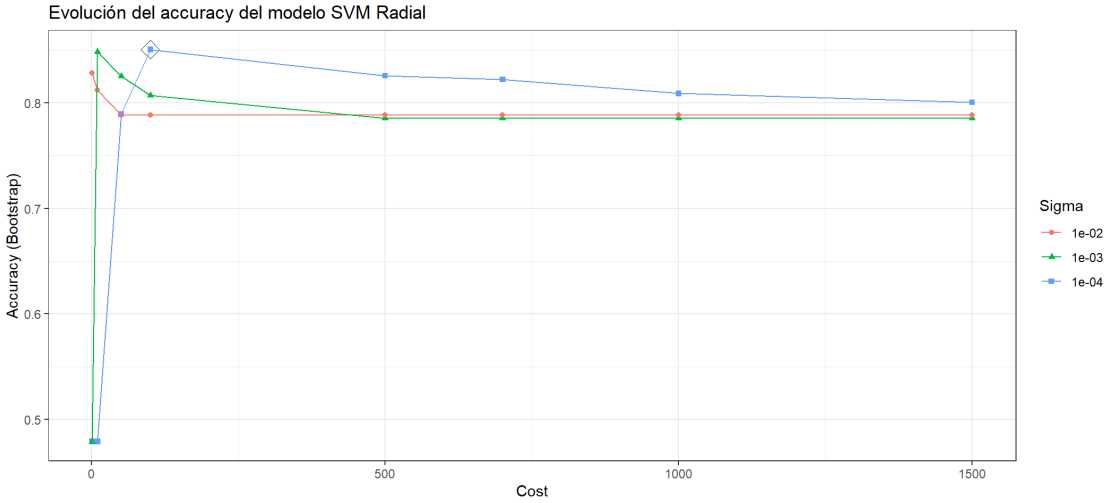
```
# PARALELIZACIÓN DE PROCESO

registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN

repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(sigma = c(0.0001, 0.001, 0.01),
                               C = c(1, 10, 50, 100, 500, 700, 1000,
1500))
set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
```

```r
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO

set.seed(342)
svmrad_pvalue_25 <- train(
                      form = type ~ .,
                      data = datos_train[c("type",
filtrado_anova_pvalue_25)],
                      method = "svmRadial",
                      tuneGrid = hiperparametros,
                      metric = "Accuracy",
                      trControl = control_train
                     )
registerDoParallel(cores = 1)
saveRDS(object = svmrad_pvalue_25, file = "svmrad_pvalue_25.rds")

svmrad_pvalue_25
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 49 samples
## 25 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04     1  0.4791786  0.01357143
##   1e-04    10  0.4791786  0.01357143
##   1e-04    50  0.7892523  0.63535263
##   1e-04   100  0.8505650  0.74779394
##   1e-04   500  0.8257308  0.71172037
##   1e-04   700  0.8220904  0.70666939
##   1e-04  1000  0.8089549  0.68633023
##   1e-04  1500  0.8005975  0.67210640
##   1e-03     1  0.4791786  0.01357143
##   1e-03    10  0.8481841  0.74491587
##   1e-03    50  0.8251649  0.71220236
##   1e-03   100  0.8072501  0.68481045
##   1e-03   500  0.7853111  0.65073789
##   1e-03   700  0.7853111  0.65073789
##   1e-03  1000  0.7853111  0.65073789
##   1e-03  1500  0.7853111  0.65073789
```

```
##   1e-02     1  0.8284625  0.72024107
##   1e-02    10  0.8120096  0.70006383
##   1e-02    50  0.7887026  0.66521928
##   1e-02   100  0.7887026  0.66521928
##   1e-02   500  0.7887026  0.66521928
##   1e-02   700  0.7887026  0.66521928
##   1e-02  1000  0.7887026  0.66521928
##   1e-02  1500  0.7887026  0.66521928
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1e-04 and C = 100.
```

```
ggplot(svmrad_pvalue_25, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo SVM Radial") +
  theme_bw()
```



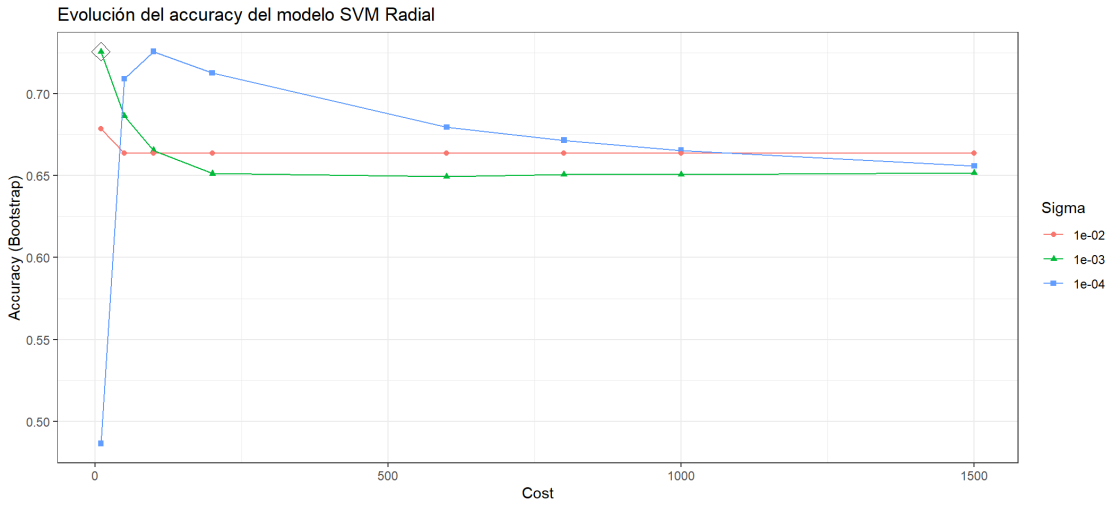
6.1.4. Filtrado por S2N 60

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(sigma = c(0.0001, 0.001, 0.01),
                               C = c(10, 50, 100, 200, 600, 800, 1000,
1500))
set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)
```

```r
# DEFINICIÓN DEL ENTRENAMIENTO

control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO

set.seed(342)
svmrad_s2n_60  <- train(
                     form = type ~ .,
                     data = datos_train[c("type", filtrado_s2n_60)],
                     method = "svmRadial",
                     tuneGrid = hiperparametros,
                     metric = "Accuracy",
                     trControl = control_train
                     )
registerDoParallel(cores = 1)
saveRDS(object = svmrad_s2n_60, file = "svmrad_s2n_60.rds")

svmrad_s2n_60

## Support Vector Machines with Radial Basis Function Kernel
##
## 49 samples
## 20 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04    10  0.4864917  0.03076805
##   1e-04    50  0.7091570  0.48233046
##   1e-04   100  0.7257777  0.53176088
##   1e-04   200  0.7126450  0.52662072
##   1e-04   600  0.6795512  0.49902179
##   1e-04   800  0.6716300  0.49519440
##   1e-04  1000  0.6654850  0.49129428
##   1e-04  1500  0.6560171  0.48037477
##   1e-03    10  0.7257777  0.53167200
##   1e-03    50  0.6864158  0.50676533
##   1e-03   100  0.6654850  0.49031464
##   1e-03   200  0.6512483  0.47725611
##   1e-03   600  0.6494919  0.47626209
##   1e-03   800  0.6507419  0.47797112
##   1e-03  1000  0.6507419  0.47797112
```

```
##   1e-03  1500  0.6517946  0.48003461
##   1e-02    10  0.6786010  0.50609606
##   1e-02    50  0.6636528  0.49192368
##   1e-02   100  0.6636528  0.49192368
##   1e-02   200  0.6636528  0.49192368
##   1e-02   600  0.6636528  0.49192368
##   1e-02   800  0.6636528  0.49192368
##   1e-02  1000  0.6636528  0.49192368
##   1e-02  1500  0.6636528  0.49192368
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.001 and C = 10.
```

```r
ggplot(svmrad_s2n_60, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo SVM Radial") +
  theme_bw()
```



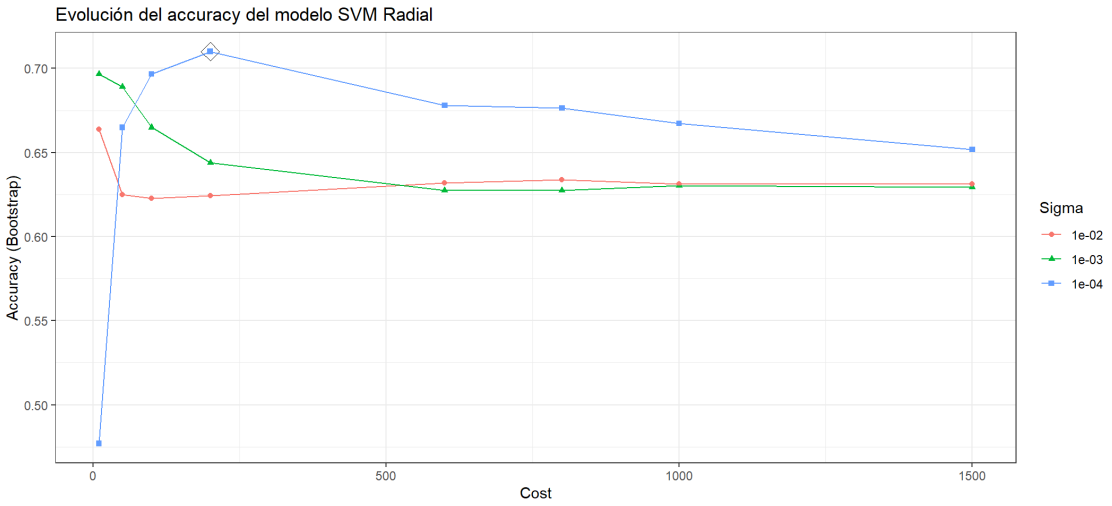
6.1.5. Filtrado por S2N 30

```r
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN

repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(sigma = c(0.0001, 0.001, 0.01),
                               C = c(10, 50, 100, 200, 600, 800, 1000,
1500))
set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
```

```r
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO

control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO

set.seed(342)
svmrad_s2n_30 <- train(
                      form = type ~ .,
                      data = datos_train[c("type", filtrado_s2n_30)],
                      method = "svmRadial",
                      tuneGrid = hiperparametros,
                      metric = "Accuracy",
                      trControl = control_train
                      )
registerDoParallel(cores = 1)
saveRDS(object = svmrad_s2n_30, file = "svmrad_s2n_30.rds")

svmrad_s2n_30

## Support Vector Machines with Radial Basis Function Kernel
##
## 49 samples
## 10 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04    10  0.4769564  0.009714286
##   1e-04    50  0.6648663  0.402539057
##   1e-04   100  0.6966755  0.464503611
##   1e-04   200  0.7100331  0.499058760
##   1e-04   600  0.6779873  0.486986496
##   1e-04   800  0.6764801  0.493523617
##   1e-04  1000  0.6672388  0.483504339
##   1e-04  1500  0.6517586  0.469170081
##   1e-03    10  0.6966755  0.464503611
##   1e-03    50  0.6890635  0.498065252
##   1e-03   100  0.6649055  0.479133741
##   1e-03   200  0.6439611  0.462105554
##   1e-03   600  0.6273788  0.444329426
```

```
##   1e-03   800  0.6275152  0.445678473
##   1e-03  1000  0.6302944  0.448523190
##   1e-03  1500  0.6295012  0.446742484
##   1e-02    10  0.6637464  0.478358819
##   1e-02    50  0.6248017  0.438119899
##   1e-02   100  0.6228527  0.437842916
##   1e-02   200  0.6241604  0.444385364
##   1e-02   600  0.6319136  0.458254620
##   1e-02   800  0.6338191  0.460649282
##   1e-02  1000  0.6314159  0.457549648
##   1e-02  1500  0.6314191  0.454872180
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1e-04 and C = 200.
```

```
ggplot(svmrad_s2n_30, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo SVM Radial") +
  theme_bw()
```



6.1.6. Reducción PCA

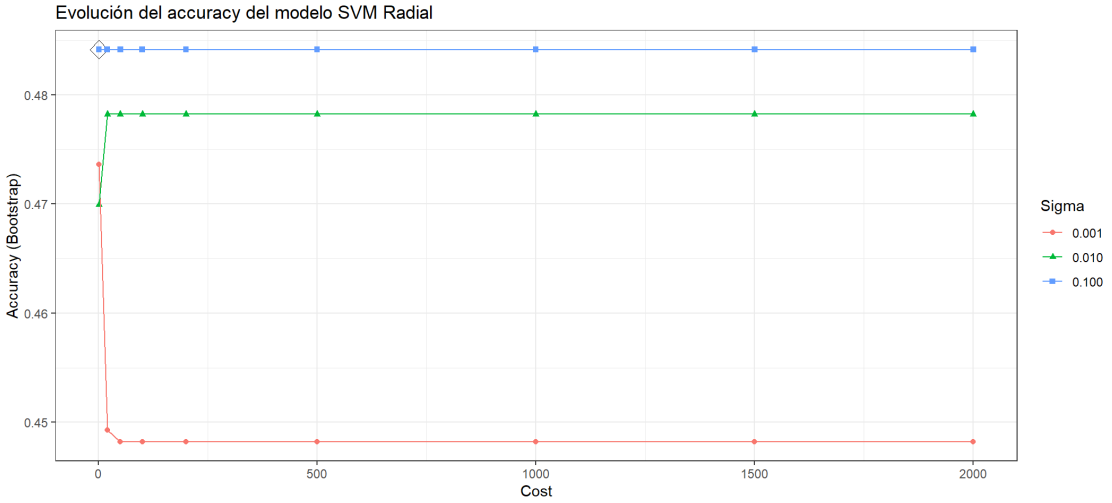
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(sigma = c(0.001, 0.01, 0.1),
                               C = c(1, 20, 50, 100, 200, 500, 1000,
1500, 2000))
set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
```

```r
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
svmrad_pca <- train(form = type ~ .,
                    data = datos_train_pca,
                    method = "svmRadial",
                    tuneGrid = hiperparametros,
                    metric = "Accuracy",
                    trControl = control_train)
registerDoParallel(cores = 1)
saveRDS(object = svmrad_pca, file = "svmrad_pca.rds")

svmrad_pca
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 49 samples
## 46 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   0.001     1  0.4736231   0.0004081633
##   0.001    20  0.4492847   0.0154918078
##   0.001    50  0.4482391   0.0210599500
##   0.001   100  0.4482391   0.0210599500
##   0.001   200  0.4482391   0.0210599500
##   0.001   500  0.4482391   0.0210599500
##   0.001  1000  0.4482391   0.0210599500
##   0.001  1500  0.4482391   0.0210599500
##   0.001  2000  0.4482391   0.0210599500
##   0.010     1  0.4699213  -0.0077414960
##   0.010    20  0.4782170   0.0252188278
##   0.010    50  0.4782170   0.0252188278
##   0.010   100  0.4782170   0.0252188278
##   0.010   200  0.4782170   0.0252188278
##   0.010   500  0.4782170   0.0252188278
##   0.010  1000  0.4782170   0.0252188278
##   0.010  1500  0.4782170   0.0252188278
```

```
##   0.010  2000  0.4782170   0.0252188278
##   0.100     1  0.4841459   0.0000000000
##   0.100    20  0.4841459   0.0000000000
##   0.100    50  0.4841459   0.0000000000
##   0.100   100  0.4841459   0.0000000000
##   0.100   200  0.4841459   0.0000000000
##   0.100   500  0.4841459   0.0000000000
##   0.100  1000  0.4841459   0.0000000000
##   0.100  1500  0.4841459   0.0000000000
##   0.100  2000  0.4841459   0.0000000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1 and C = 1.
```

```
ggplot(svmrad_pca, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo SVM Radial") +
  theme_bw()
```



6.2. Random Forest

El método ranger de caret emplea la función ranger() del paquete ranger. Este algoritmo tiene 3 hiperparámetros:

- mtry: número predictores seleccionados aleatoriamente en cada árbol.
- min.node.size: tamaño mínimo que tiene que tener un nodo para poder ser dividido.
- splitrule: criterio de división.

6.2.1. Filtrado por ANOVA p-value 100

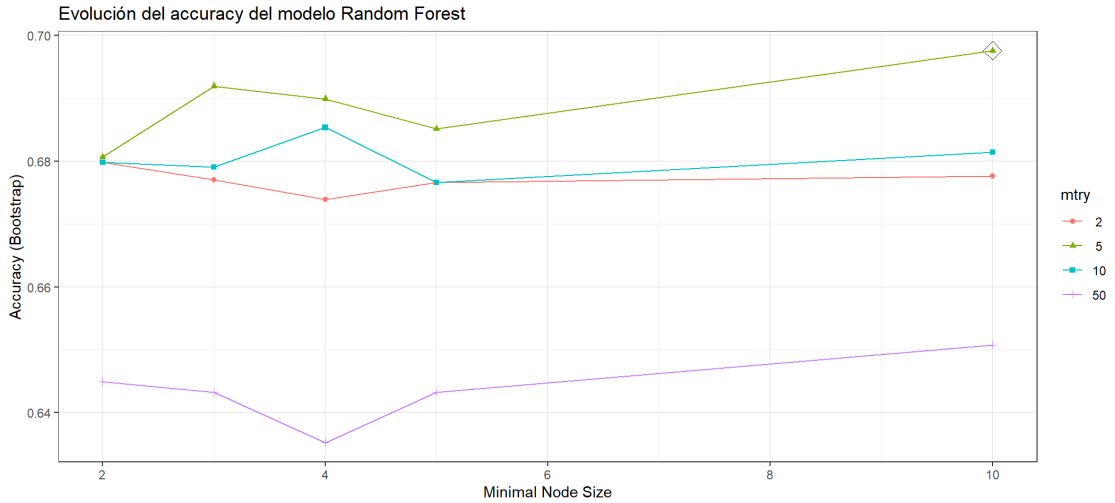
```
library(ranger)
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)
```

```r
# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(2, 5, 10, 50),
                               min.node.size = c(2, 3, 4, 5, 10),
                               splitrule = "gini")

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
rf_pvalue_100 <- train(
                  form = type ~ .,
                  data = datos_train[c("type",
filtrado_anova_pvalue_100)],
                  method = "ranger",
                  tuneGrid = hiperparametros,
                  metric = "Accuracy",
                  trControl = control_train,
                  # Número de árboles ajustados
                  num.trees = 500)

saveRDS(object = rf_pvalue_100, file = "rf_pvalue_100.rds")
registerDoParallel(cores = 1)

rf_pvalue_100
```

```
## Random Forest
##
##  49 samples
## 100 predictors
##   4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
```

```
##    mtry  min.node.size  Accuracy   Kappa
##     2     2             0.6798472  0.4300736
##     2     3             0.6770080  0.4256451
##     2     4             0.6739027  0.4204609
##     2     5             0.6766081  0.4270578
##     2    10             0.6776438  0.4270945
##     5     2             0.6806641  0.4298455
##     5     3             0.6919310  0.4570252
##     5     4             0.6899081  0.4493900
##     5     5             0.6851499  0.4395209
##     5    10             0.6975859  0.4667225
##    10     2             0.6798315  0.4336897
##    10     3             0.6790634  0.4345710
##    10     4             0.6853509  0.4387644
##    10     5             0.6765904  0.4208001
##    10    10             0.6814090  0.4385253
##    50     2             0.6449321  0.3800940
##    50     3             0.6432133  0.3694895
##    50     4             0.6352505  0.3600726
##    50     5             0.6432412  0.3724686
##    50    10             0.6507881  0.3947245
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, splitrule = gini
##  and min.node.size = 10.
```

```r
ggplot(rf_pvalue_100, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo Random Forest") +
  guides(color = guide_legend(title = "mtry"),
         shape = guide_legend(title = "mtry")) +
  theme_bw()
```

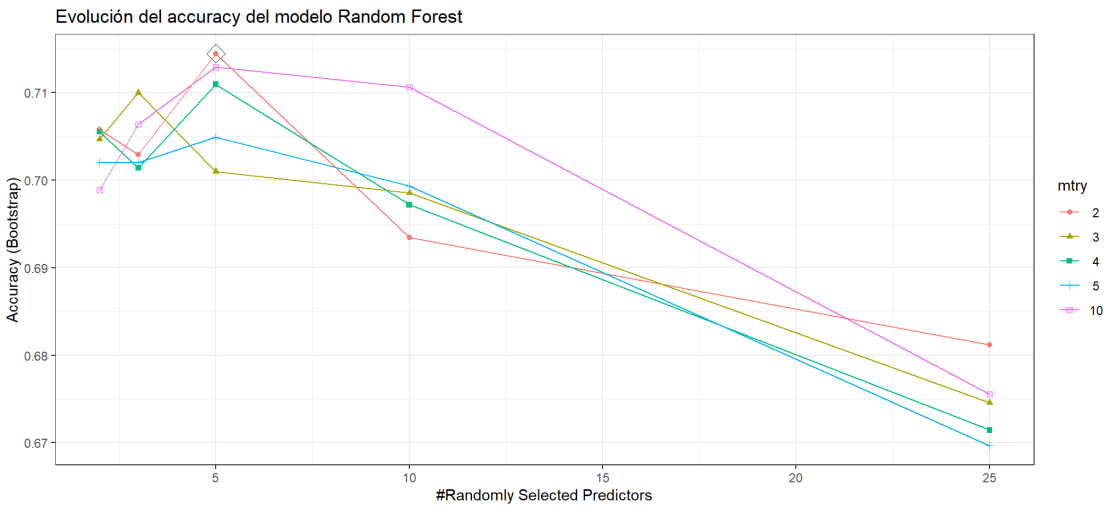


6.2.2. Filtrado por ANOVA p-value 50

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(2, 3, 5, 10, 25),
                               min.node.size = c(2, 3, 4, 5, 10),
                               splitrule = "gini")

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
rf_pvalue_50 <- train(
                  form = type ~ .,
                  data = datos_train[c("type",
filtrado_anova_pvalue_50)],
                  method = "ranger",
                  tuneGrid = hiperparametros,
                  metric = "Accuracy",
                  trControl = control_train,
                  # Número de árboles ajustados
                  num.trees = 500)

saveRDS(object = rf_pvalue_50, file = "rf_pvalue_50.rds")
registerDoParallel(cores = 1)
rf_pvalue_50

## Random Forest
##
## 49 samples
## 50 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
```

```
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    2     2             0.7057514  0.4794300
##    2     3             0.7046865  0.4785506
##    2     4             0.7055347  0.4784795
##    2     5             0.7020731  0.4737850
##    2    10             0.6988543  0.4620093
##    3     2             0.7029384  0.4750800
##    3     3             0.7099977  0.4912363
##    3     4             0.7014027  0.4699436
##    3     5             0.7020090  0.4742648
##    3    10             0.7063730  0.4819185
##    5     2             0.7144791  0.4988873
##    5     3             0.7010057  0.4722598
##    5     4             0.7109608  0.4870895
##    5     5             0.7048977  0.4857207
##    5    10             0.7129178  0.4963112
##   10     2             0.6934587  0.4566851
##   10     3             0.6985381  0.4705741
##   10     4             0.6972005  0.4639143
##   10     5             0.6993348  0.4752093
##   10    10             0.7106357  0.4924550
##   25     2             0.6811648  0.4349853
##   25     3             0.6745766  0.4292462
##   25     4             0.6714557  0.4163299
##   25     5             0.6696896  0.4172944
##   25    10             0.6755583  0.4349199
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, splitrule = gini
##  and min.node.size = 2.
```

```
ggplot(rf_pvalue_50, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo Random Forest") +
  guides(color = guide_legend(title = "mtry"),
         shape = guide_legend(title = "mtry")) +
  theme_bw()
```



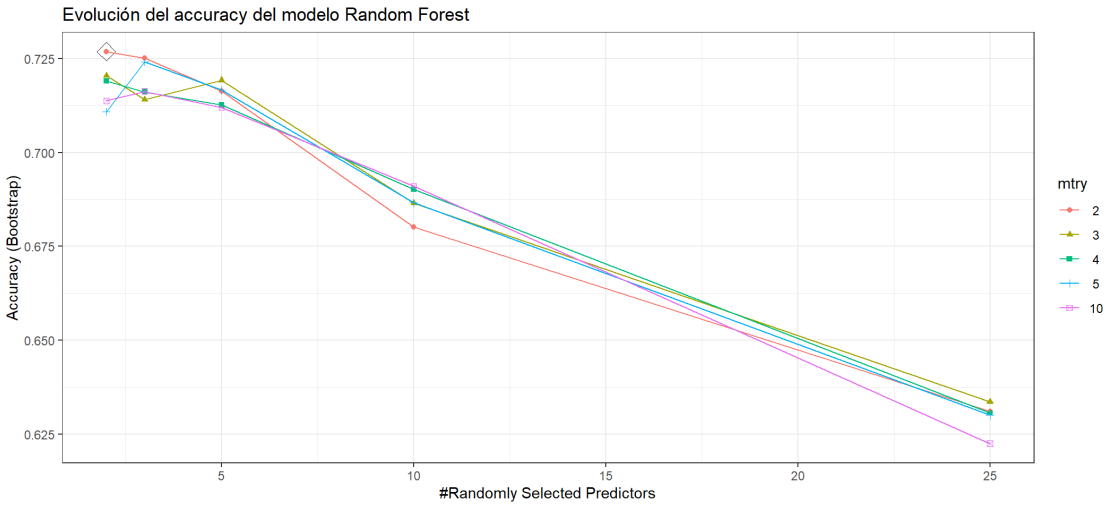
6.2.3. Filtrado por ANOVA p-value 25

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(2, 3, 5, 10, 25),
                               min.node.size = c(2, 3, 4, 5, 10),
                               splitrule = "gini")

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
rf_pvalue_25 <- train(
                  form = type ~ .,
                  data = datos_train[c("type",
filtrado_anova_pvalue_25)],
                  method = "ranger",
                  tuneGrid = hiperparametros,
                  metric = "Accuracy",
```

```
                  trControl = control_train,
                  # Número de árboles ajustados
                  num.trees = 500)

saveRDS(object = rf_pvalue_25, file = "rf_pvalue_25.rds")
registerDoParallel(cores = 1)
rf_pvalue_25
```

```
## Random Forest
##
## 49 samples
## 25 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    2     2             0.7268851  0.5186574
##    2     3             0.7204773  0.5081968
##    2     4             0.7190363  0.5030187
##    2     5             0.7108438  0.4909260
##    2    10             0.7137548  0.4965100
##    3     2             0.7251496  0.5185306
##    3     3             0.7140901  0.5009942
##    3     4             0.7161273  0.5033781
##    3     5             0.7241531  0.5168568
##    3    10             0.7162661  0.5041587
##    5     2             0.7163468  0.5069336
##    5     3             0.7192578  0.5156517
##    5     4             0.7126411  0.5016437
##    5     5             0.7165796  0.5093218
##    5    10             0.7120131  0.5029672
##   10     2             0.6801762  0.4548959
##   10     3             0.6865149  0.4668726
##   10     4             0.6901663  0.4688182
##   10     5             0.6866384  0.4714100
##   10    10             0.6910156  0.4807366
##   25     2             0.6310817  0.3790040
##   25     3             0.6335705  0.3844626
##   25     4             0.6306200  0.3803898
##   25     5             0.6300370  0.3819064
##   25    10             0.6225092  0.3696509
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = gini
##  and min.node.size = 2.
```

```r
ggplot(rf_pvalue_25, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo Random Forest") +
  guides(color = guide_legend(title = "mtry"),
         shape = guide_legend(title = "mtry")) +
  theme_bw()
```



6.2.4. Filtrado por S2N 60

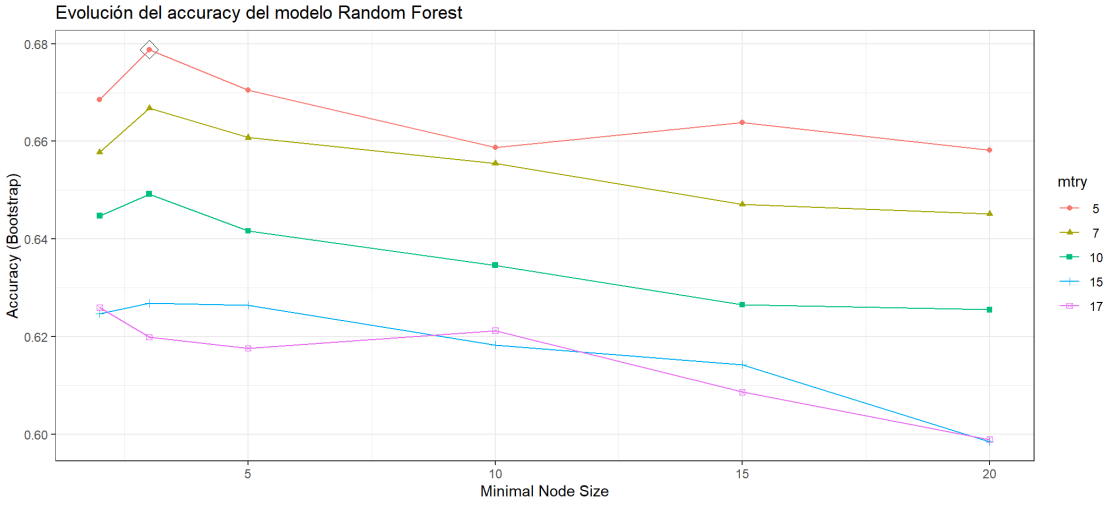
```r
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(5, 7, 10, 15, 17),
                               min.node.size = c(2, 3, 5, 10, 15, 20),
                               splitrule = "gini")

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
rf_s2n_60 <- train(
```

```
                form = type ~ .,
                data = datos_train[c("type", filtrado_s2n_60)],
                method = "ranger",
                tuneGrid = hiperparametros,
                metric = "Accuracy",
                trControl = control_train,
                # Número de árboles ajustados
                num.trees = 500
              )

saveRDS(object = rf_s2n_60, file = "rf_s2n_60.rds")
registerDoParallel(cores = 1)
rf_s2n_60
```

```
## Random Forest
##
## 49 samples
## 20 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    5     2             0.6685457  0.4409409
##    5     3             0.6788081  0.4618042
##    5     5             0.6705095  0.4473356
##    5    10             0.6587297  0.4355933
##    5    15             0.6638268  0.4399565
##    5    20             0.6581549  0.4314073
##    7     2             0.6577532  0.4337402
##    7     3             0.6668044  0.4447252
##    7     5             0.6607570  0.4361864
##    7    10             0.6555020  0.4349191
##    7    15             0.6470919  0.4211900
##    7    20             0.6451377  0.4199913
##   10     2             0.6447452  0.4136643
##   10     3             0.6492153  0.4247465
##   10     5             0.6416606  0.4103971
##   10    10             0.6346241  0.4032295
##   10    15             0.6264952  0.3965604
##   10    20             0.6255049  0.3966521
##   15     2             0.6246667  0.3864101
##   15     3             0.6268347  0.3874754
##   15     5             0.6264175  0.3881988
##   15    10             0.6182825  0.3852780
##   15    15             0.6142671  0.3858374
##   15    20             0.5984971  0.3595038
```

```
##   17     2             0.6259446  0.3844910
##   17     3             0.6198851  0.3768644
##   17     5             0.6175723  0.3721508
##   17    10             0.6212337  0.3864994
##   17    15             0.6086247  0.3758128
##   17    20             0.5989272  0.3601710
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, splitrule = gini
##  and min.node.size = 3.
```

```r
ggplot(rf_s2n_60, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo Random Forest") +
  guides(color = guide_legend(title = "mtry"),
         shape = guide_legend(title = "mtry")) +
  theme_bw()
```



6.2.5. Filtrado por S2N 30

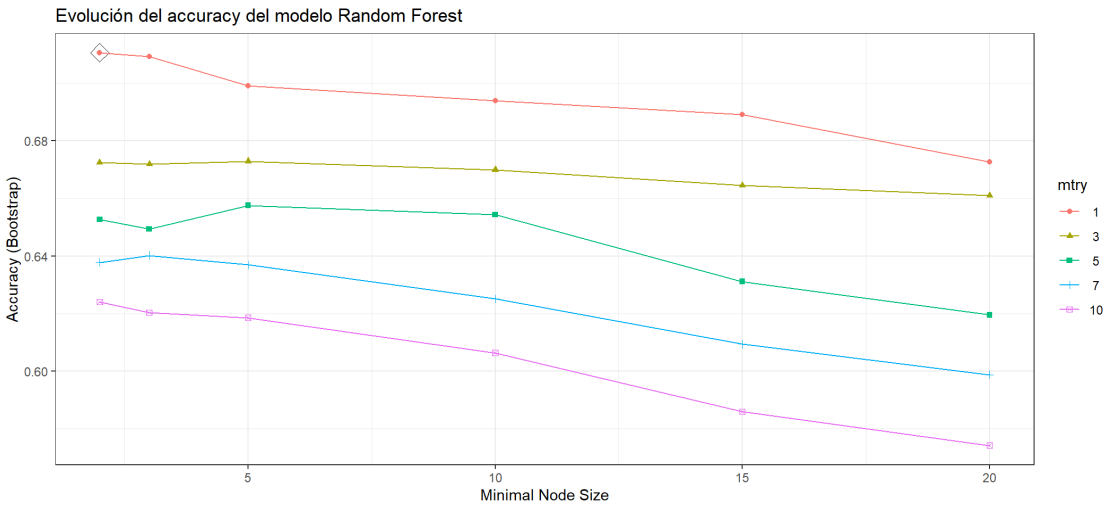
```r
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(1,3,5, 7, 10),
                               min.node.size = c(2, 3, 5, 10, 15, 20),
                               splitrule = "gini")

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
```

```
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
rf_s2n_30 <- train(
                form = type ~ .,
                data = datos_train[c("type", filtrado_s2n_30)],
                method = "ranger",
                tuneGrid = hiperparametros,
                metric = "Accuracy",
                trControl = control_train,
                # Número de árboles ajustados
                num.trees = 500
              )

saveRDS(object = rf_s2n_30, file = "rf_s2n_30.rds")
registerDoParallel(cores = 1)
rf_s2n_30
```

```
## Random Forest
##
## 49 samples
## 10 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    1     2             0.7106013  0.5089746
##    1     3             0.7091867  0.5056827
##    1     5             0.6991224  0.4871032
##    1    10             0.6939435  0.4773066
##    1    15             0.6891206  0.4630077
##    1    20             0.6726367  0.4310457
##    3     2             0.6724810  0.4529313
##    3     3             0.6719120  0.4486945
##    3     5             0.6729120  0.4480963
##    3    10             0.6699596  0.4424375
##    3    15             0.6645006  0.4366850
##    3    20             0.6610055  0.4284100
```

```
##     5     2             0.6526503  0.4243360
##     5     3             0.6493064  0.4201802
##     5     5             0.6575539  0.4308684
##     5    10             0.6543695  0.4248813
##     5    15             0.6311484  0.3929160
##     5    20             0.6195337  0.3763852
##     7     2             0.6377689  0.4048068
##     7     3             0.6402012  0.4071014
##     7     5             0.6368917  0.3994751
##     7    10             0.6251468  0.3823249
##     7    15             0.6093747  0.3656798
##     7    20             0.5987573  0.3523871
##    10     2             0.6240269  0.3831052
##    10     3             0.6202955  0.3747041
##    10     5             0.6185685  0.3726282
##    10    10             0.6063503  0.3593849
##    10    15             0.5860059  0.3352943
##    10    20             0.5742160  0.3199542
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 1, splitrule = gini
##  and min.node.size = 2.
```

```
ggplot(rf_s2n_30, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo Random Forest") +
  guides(color = guide_legend(title = "mtry"),
         shape = guide_legend(title = "mtry")) +
  theme_bw()
```



6.2.6. Reducción PCA

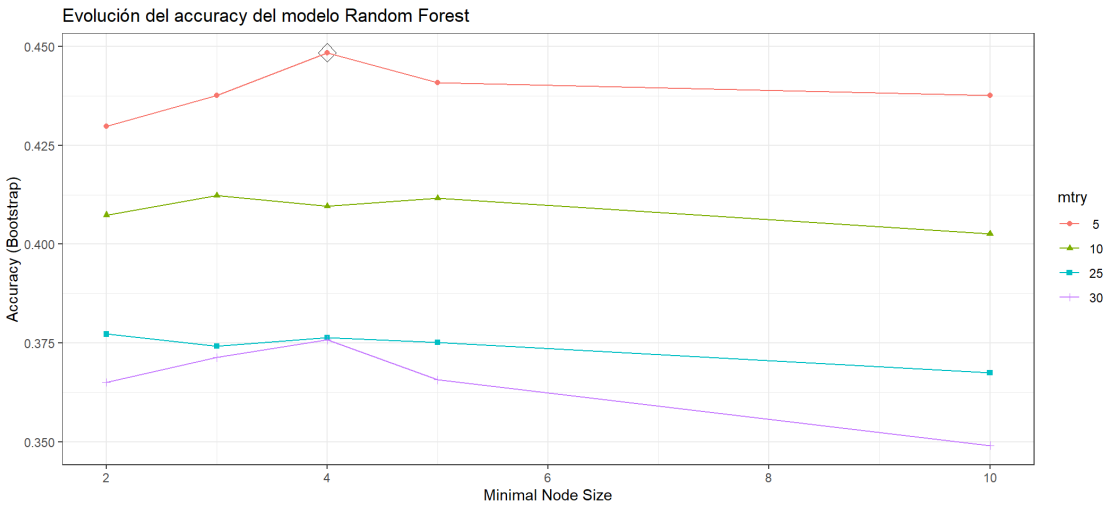
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
```

```r
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(5, 10, 25, 30),
                               min.node.size = c(2, 3, 4, 5, 10),
                               splitrule = "gini")

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
rf_pca <- train(
            form   = type ~ .,
            data   = datos_train_pca,
            method = "ranger",
            tuneGrid = hiperparametros,
            metric = "Accuracy",
            trControl = control_train,
            # Número de árboles ajustados
            num.trees = 500)

saveRDS(object = rf_pca, file = "rf_pca.rds")
registerDoParallel(cores = 1)
rf_pca
```

```
## Random Forest
##
## 49 samples
## 46 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    5     2             0.4297329  -0.04334187
##    5     3             0.4375212  -0.02665502
```

```
##     5     4            0.4483812  -0.01559800
##     5     5            0.4408445  -0.02258052
##     5    10            0.4375119  -0.02980272
##    10     2            0.4073832  -0.06659585
##    10     3            0.4123192  -0.05684472
##    10     4            0.4096022  -0.05900057
##    10     5            0.4115755  -0.05254642
##    10    10            0.4025760  -0.05266537
##    25     2            0.3772842  -0.07865664
##    25     3            0.3742560  -0.07932350
##    25     4            0.3763932  -0.07543047
##    25     5            0.3751794  -0.08186112
##    25    10            0.3675643  -0.07356611
##    30     2            0.3651116  -0.08504378
##    30     3            0.3713938  -0.07476173
##    30     4            0.3758135  -0.07031013
##    30     5            0.3657506  -0.08316182
##    30    10            0.3491711  -0.09337689
##
## Tuning parameter 'splitrule' was held constant at a value of gini
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 5, splitrule = gini
##  and min.node.size = 4.
```

```
ggplot(rf_pca, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo Random Forest") +
  guides(color = guide_legend(title = "mtry"),
         shape = guide_legend(title = "mtry")) +
  theme_bw()
```



6.3. Neural Network

El método nnet de caret emplea la función nnet() del paquete nnet para crear redes neuronales con una capa oculta. Este algoritmo tiene 2 hiperparámetros:

- size: número de neuronas en la capa oculta.
- decay: controla la regularización durante el entrenamiento de la red.

En vista de los resultados obtenidos con los algoritmos anteriores, no se empleará el filtrado por reducción PCA.

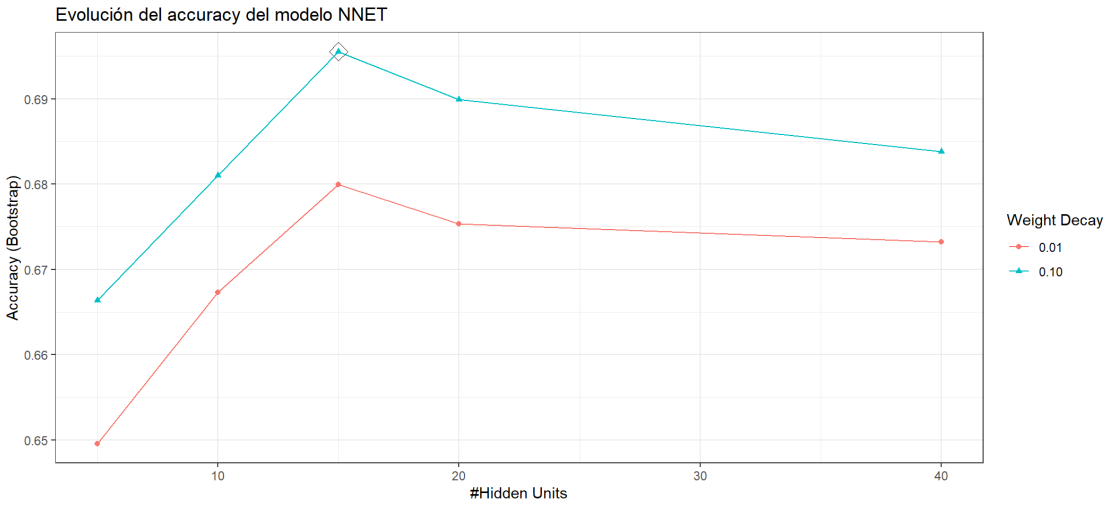
6.3.1. Filtrado por S2N 60

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(size = c(5, 10, 15, 20, 40),
                               decay = c(0.01, 0.1))

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
nnet_s2n_60 <- train(
                 form = type ~ .,
                 data = datos_train[c("type", filtrado_s2n_60)],
                 method = "nnet",
                 tuneGrid = hiperparametros,
                 metric = "Accuracy",
                 trControl = control_train,
                 # Rango de inicialización de los pesos
                 rang = c(-0.7, 0.7),
                 # Número máximo de pesos
                 MaxNWts = 10000,
                 # Para que no se muestre cada iteración por pantalla
                 trace = FALSE
               )

saveRDS(object = nnet_s2n_60, file = "nnet_s2n_60.rds")
```

```r
registerDoParallel(cores = 1)
nnet_s2n_60
```

```
## Neural Network
##
## 49 samples
## 20 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.6495678  0.4778446
##    5    0.10   0.6663865  0.4947147
##   10    0.01   0.6672967  0.5004936
##   10    0.10   0.6809899  0.5146938
##   15    0.01   0.6799743  0.5186047
##   15    0.10   0.6955528  0.5361675
##   20    0.01   0.6753407  0.5126087
##   20    0.10   0.6899188  0.5298255
##   40    0.01   0.6732396  0.5084195
##   40    0.10   0.6838120  0.5210241
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 15 and decay = 0.1.
```

```r
ggplot(nnet_s2n_60, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo NNET") +
  theme_bw()
```



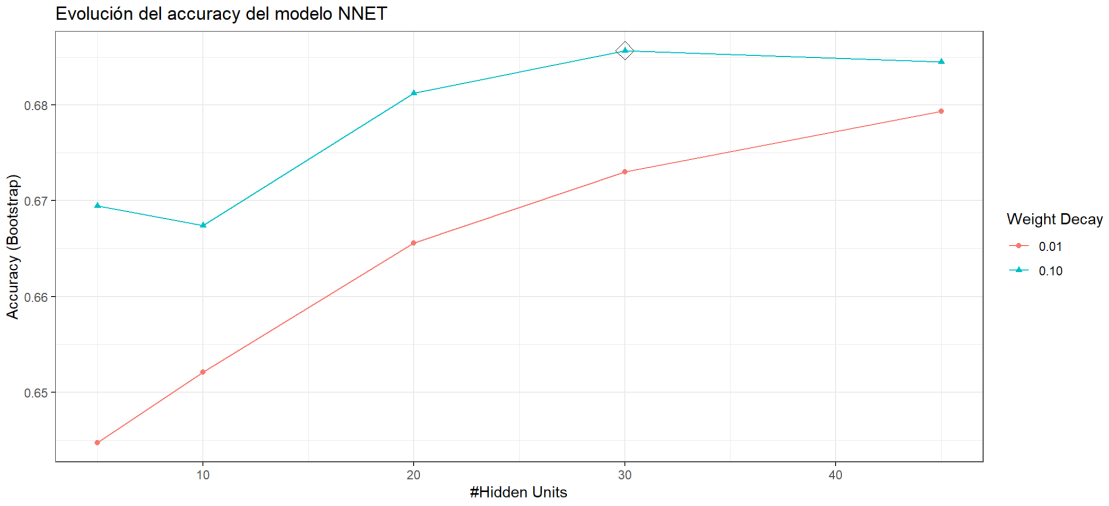
6.3.2. Filtrado por S2N 30

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(size = c(5, 10, 20, 30, 45),
                               decay = c(0.01, 0.1))

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
nnet_s2n_30 <- train(
                form = type ~ .,
                data = datos_train[c("type", filtrado_s2n_30)],
                method = "nnet",
                tuneGrid = hiperparametros,
                metric = "Accuracy",
                trControl = control_train,
                # Rango de inicialización de los pesos
                rang = c(-0.7, 0.7),
                # Número máximo de pesos
                MaxNWts = 10000,
                # Para que no se muestre cada iteración por pantalla
                trace = FALSE
              )

saveRDS(object = nnet_s2n_30, file = "nnet_s2n_30.rds")
registerDoParallel(cores = 1)
nnet_s2n_30
```

```
## Neural Network
##
## 49 samples
## 10 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
```

```
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.6447627  0.4574065
##    5    0.10   0.6694483  0.4883429
##   10    0.01   0.6521323  0.4698893
##   10    0.10   0.6674295  0.4873935
##   20    0.01   0.6655785  0.4891172
##   20    0.10   0.6812023  0.5113826
##   30    0.01   0.6730217  0.5028495
##   30    0.10   0.6856681  0.5166723
##   45    0.01   0.6793071  0.5134918
##   45    0.10   0.6844764  0.5136475
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 30 and decay = 0.1.
```

```
ggplot(nnet_s2n_30, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo NNET") +
  theme_bw()
```



6.3.3. Filtrado por ANOVA p-value 100

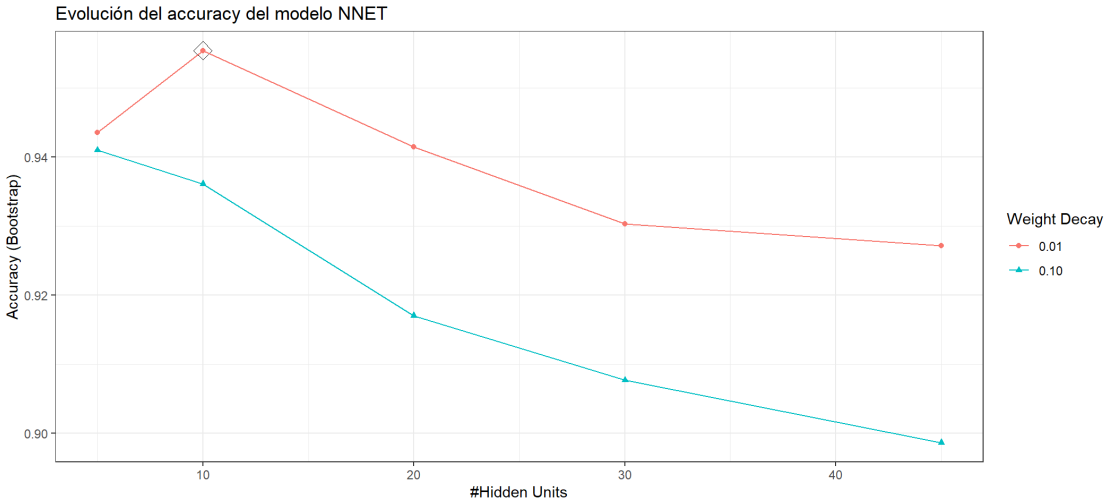
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(size = c(5, 10, 20, 30, 45),
                               decay = c(0.01, 0.1))
```

```r
set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
nnet_pvalue_100 <- train(
                 form = type ~ .,
                 data = datos_train[c("type", filtrado_anova_pvalue_100)],
                 method = "nnet",
                 tuneGrid = hiperparametros,
                 metric = "Accuracy",
                 trControl = control_train,
                 # Rango de inicialización de los pesos
                 rang = c(-0.7, 0.7),
                 # Número máximo de pesos
                 MaxNWts = 10000,
                 # Para que no se muestre cada iteración por pantalla
                 trace = FALSE
               )

saveRDS(object = nnet_pvalue_100, file = "nnet_pvalue_100.rds")
registerDoParallel(cores = 1)
nnet_pvalue_100
```

```
## Neural Network
##
##  49 samples
## 100 predictors
##   4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.9435589  0.9112923
##    5    0.10   0.9410170  0.9083058
##   10    0.01   0.9554087  0.9288216
```

```
##   10    0.10   0.9360976  0.8993654
##   20    0.01   0.9414546  0.9088895
##   20    0.10   0.9170129  0.8706031
##   30    0.01   0.9303129  0.8899313
##   30    0.10   0.9076494  0.8548389
##   45    0.01   0.9270981  0.8857888
##   45    0.10   0.8986172  0.8392153
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 10 and decay = 0.01.
```

```
ggplot(nnet_pvalue_100, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo NNET") +
  theme_bw()
```



6.3.4. Filtrado por ANOVA p-value 50

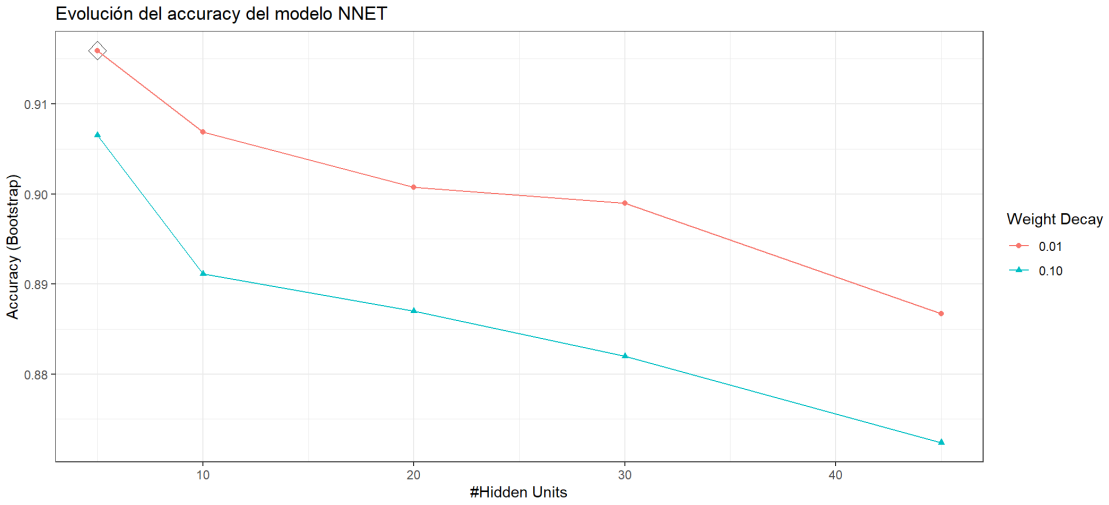
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(size = c(5, 10, 20, 30, 45),
                               decay = c(0.01, 0.1))

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
```

```r
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
nnet_pvalue_50 <- train(
                 form = type ~ .,
                 data = datos_train[c("type", filtrado_anova_pvalue_50)],
                 method = "nnet",
                 tuneGrid = hiperparametros,
                 metric = "Accuracy",
                 trControl = control_train,
                 # Rango de inicialización de los pesos
                 rang = c(-0.7, 0.7),
                 # Número máximo de pesos
                 MaxNWts = 10000,
                 # Para que no se muestre cada iteración por pantalla
                 trace = FALSE
               )

saveRDS(object = nnet_pvalue_50, file = "nnet_pvalue_50.rds")
registerDoParallel(cores = 1)
nnet_pvalue_50
```

```
## Neural Network
##
## 49 samples
## 50 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.9159172  0.8692827
##    5    0.10   0.9065225  0.8498716
##   10    0.01   0.9068488  0.8509746
##   10    0.10   0.8911099  0.8229043
##   20    0.01   0.9007177  0.8410253
##   20    0.10   0.8869928  0.8144026
##   30    0.01   0.8989804  0.8362181
##   30    0.10   0.8819866  0.8057649
##   45    0.01   0.8866979  0.8159156
##   45    0.10   0.8724008  0.7893556
##
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 5 and decay = 0.01.
```

```
ggplot(nnet_pvalue_50, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo NNET") +
  theme_bw()
```



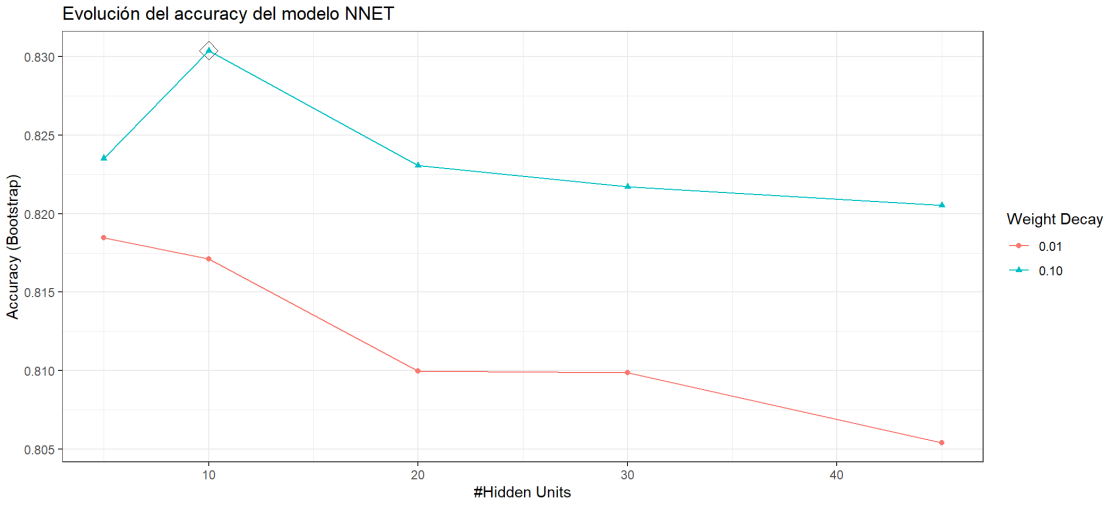
6.3.5. Filtrado por ANOVA p-value 25

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(size = c(5, 10, 20, 30, 45),
                               decay = c(0.01, 0.1))

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
nnet_pvalue_25 <- train(
```

```
                form = type ~ .,
                data = datos_train[c("type", filtrado_anova_pvalue_25)],
                method = "nnet",
                tuneGrid = hiperparametros,
                metric = "Accuracy",
                trControl = control_train,
                # Rango de inicialización de los pesos
                rang = c(-0.7, 0.7),
                # Número máximo de pesos
                MaxNWts = 10000,
                # Para que no se muestre cada iteración por pantalla
                trace = FALSE
              )

saveRDS(object = nnet_pvalue_25, file = "nnet_pvalue_25.rds")
registerDoParallel(cores = 1)
nnet_pvalue_25
```

```
## Neural Network
##
## 49 samples
## 25 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.8184713  0.7014756
##    5    0.10   0.8235256  0.7093696
##   10    0.01   0.8171236  0.6990311
##   10    0.10   0.8303917  0.7199855
##   20    0.01   0.8099706  0.6865294
##   20    0.10   0.8230821  0.7069075
##   30    0.01   0.8098601  0.6859726
##   30    0.10   0.8217098  0.7046837
##   45    0.01   0.8054195  0.6778669
##   45    0.10   0.8205257  0.7023326
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 10 and decay = 0.1.
```

```
ggplot(nnet_pvalue_25, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo NNET") +
  theme_bw()
```



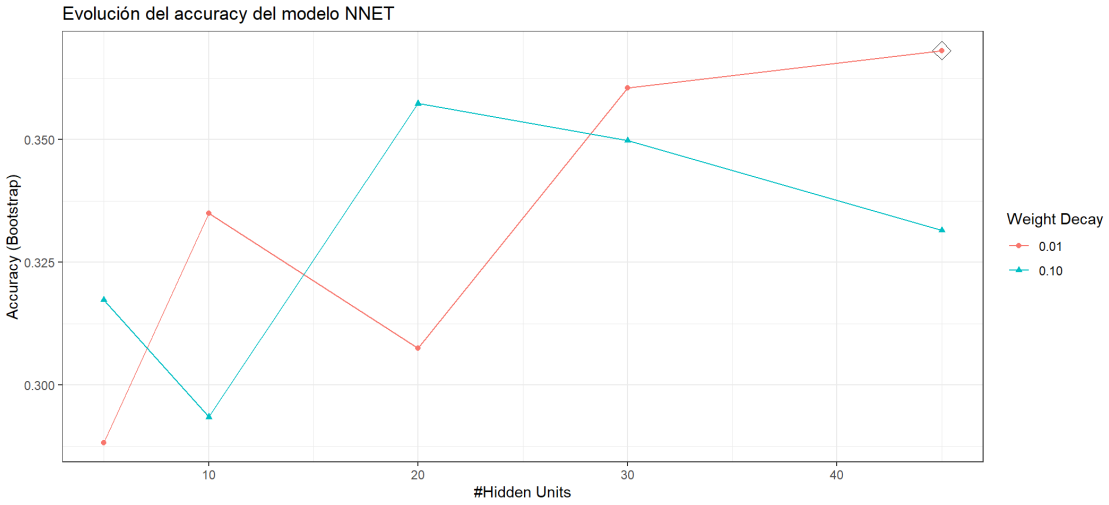
6.3.6. PCA

```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

# HIPERPARÁMETROS, NÚMERO DE REPETICIONES Y SEMILLAS PARA CADA REPETICIÓN
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(size = c(5, 10, 20, 30, 45),
                               decay = c(0.01, 0.1))

set.seed(123)
seeds <- vector(mode = "list", length = repeticiones_boot + 1)
for (i in 1:repeticiones_boot) {
  seeds[[i]] <- sample.int(1000, nrow(hiperparametros))
}
seeds[[repeticiones_boot + 1]] <- sample.int(1000, 1)

# DEFINICIÓN DEL ENTRENAMIENTO
control_train <- trainControl(method = "boot", number =
repeticiones_boot,
                              seeds = seeds, returnResamp = "final",
                              verboseIter = FALSE, allowParallel = TRUE)

# AJUSTE DEL MODELO
set.seed(342)
nnet_pca <- train(
                form = type ~ .,
                data = datos_train_pca,
                method = "nnet",
                tuneGrid = hiperparametros,
                metric = "Accuracy",
                trControl = control_train,
                # Rango de inicialización de los pesos
```

```
                  rang = c(-0.7, 0.7),
                  # Número máximo de pesos
                  MaxNWts = 10000,
                  # Para que no se muestre cada iteración por pantalla
                  trace = FALSE
                )

saveRDS(object = nnet_pca, file = "nnet_pca.rds")
registerDoParallel(cores = 1)
nnet_pca
```

```
## Neural Network
##
## 49 samples
## 46 predictors
##  4 classes: 'CIS', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 49, 49, 49, 49, 49, 49, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.2883108  -0.01194768
##    5    0.10   0.3173675   0.02224167
##   10    0.01   0.3349564   0.03659622
##   10    0.10   0.2935103   0.01291432
##   20    0.01   0.3075311   0.01252888
##   20    0.10   0.3573166   0.05606175
##   30    0.01   0.3604769   0.06449293
##   30    0.10   0.3498149   0.05640796
##   45    0.01   0.3681189   0.05983224
##   45    0.10   0.3315225   0.03642839
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 45 and decay = 0.01.
```

```
ggplot(nnet_pca, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo NNET") +
  theme_bw()
```

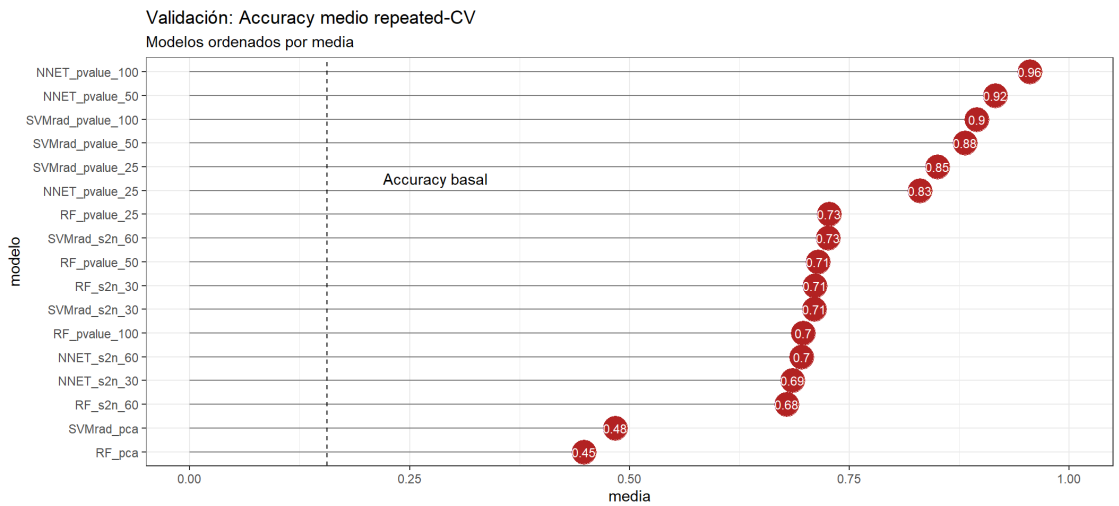


6.4. Comparación de modelos

6.4.1. Error de validación

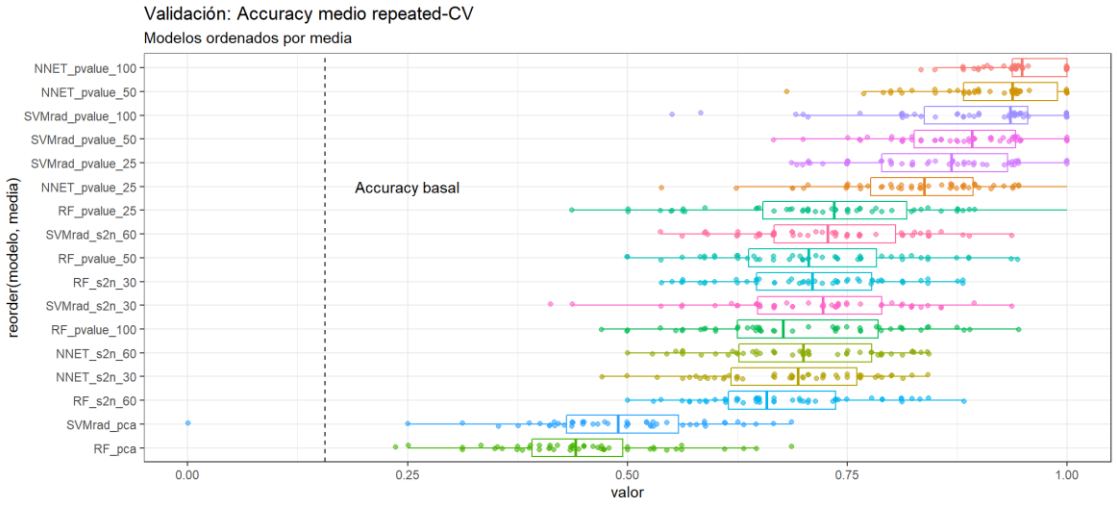
```
modelos <- list(
  SVMrad_pvalue_100 = svmrad_pvalue_100,
  SVMrad_pvalue_50  = svmrad_pvalue_50,
  SVMrad_pvalue_25  = svmrad_pvalue_25,
  SVMrad_s2n_60     = svmrad_s2n_60,
  SVMrad_s2n_30     = svmrad_s2n_30,
  SVMrad_pca        = svmrad_pca,
  RF_pvalue_100     = rf_pvalue_100,
  RF_pvalue_50      = rf_pvalue_50,
  RF_pvalue_25      = rf_pvalue_25,
  RF_s2n_60         = rf_s2n_60,
  RF_s2n_30         = rf_s2n_30,
  RF_pca            = rf_pca,
  NNET_pvalue_100   = nnet_pvalue_100,
  NNET_pvalue_50    = nnet_pvalue_50,
  NNET_pvalue_25    = nnet_pvalue_25,
  NNET_s2n_60       = nnet_s2n_60,
  NNET_s2n_30       = nnet_s2n_30
  )

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
```

```r
promedio_metricas_resamples <- metricas_resamples %>%
  group_by(modelo, metrica) %>%
  summarise(media = mean(valor)) %>%
  spread(key = metrica, value = media) %>%
  arrange(desc(Accuracy))

promedio_metricas_resamples

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 = 12.5, label = "Accuracy basal")
+
    labs(title = "Validación: Accuracy medio repeated-CV",
         subtitle = "Modelos ordenados por media",
         x = "modelo") +
    coord_flip() +
    theme_bw()
```

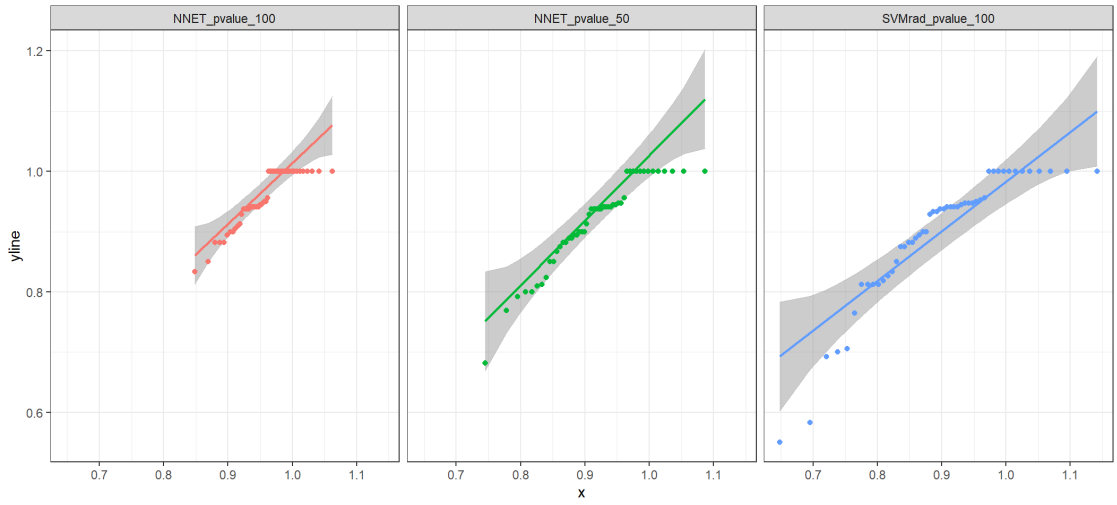



```r
metricas_resamples %>% filter(metrica == "Accuracy") %>%
  group_by(modelo) %>%
  mutate(media = mean(valor)) %>%
  ungroup() %>%
  ggplot(aes(x = reorder(modelo, media), y = valor, color = modelo)) +
```

```
    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 = 0.25, x = 12, label = "Accuracy basal") +
    theme_bw() +
    labs(title = "Validación: Accuracy medio repeated-CV",
         subtitle = "Modelos ordenados por media") +
    coord_flip() +
    theme(legend.position = "none")
```




```
library(qqplotr)
metricas_resamples %>%
  filter(modelo %in% c("NNET_pvalue_100", "NNET_pvalue_50",
"SVMrad_pvalue_100") &
           metrica == "Accuracy") %>%
  ggplot(aes(sample = valor, color = modelo)) +
    stat_qq_band(alpha = 0.5, color = "gray") +
    stat_qq_line() +
    stat_qq_point() +
    theme_bw() +
  theme(legend.position = "none") +
    facet_wrap(~modelo)
```



El análisis gráfico no muestra grandes desviaciones de la normal, además, dado que se dispone de más de 30 valores por grupo, el t-test tiene cierta robustez. Se procede a comparar los modelos.

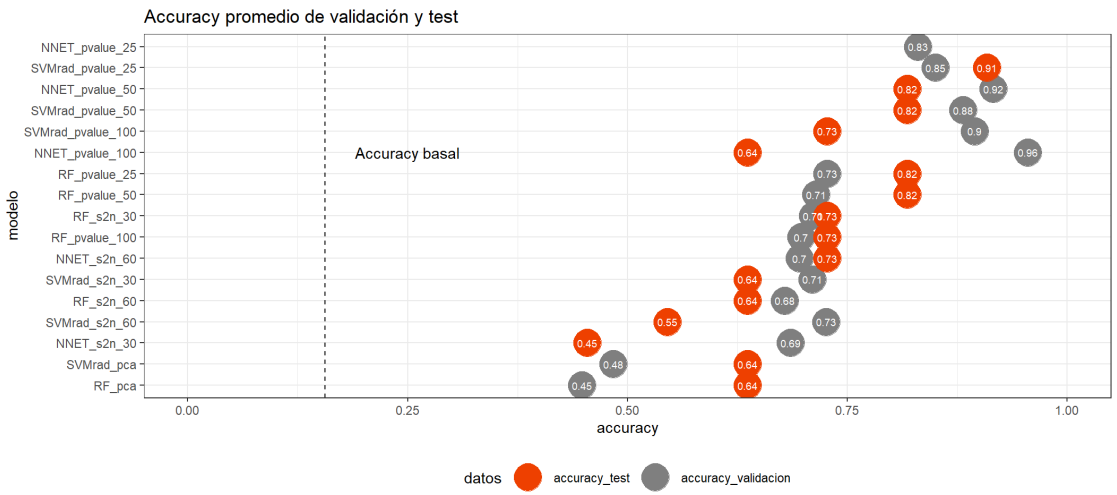
```
metricas_ttest <- metricas_resamples %>%
  dplyr::filter(modelo %in% c("SVMrad_pvalue_100","NNET_pvalue_50",
"NNET_pvalue_100") &
                  metrica == "Accuracy") %>%
  dplyr::select(-metrica)

pairwise.t.test(x = metricas_ttest$valor,
                g = metricas_ttest$modelo,
                paired = TRUE,
                # Al ser solo 2 comparaciones, no se añade ajuste de
p.value
                p.adjust.method = "none")

##
##  Pairwise comparisons using paired t tests
##
## data:  metricas_ttest$valor and metricas_ttest$modelo
##
##                   NNET_pvalue_100 NNET_pvalue_50
## NNET_pvalue_50    0.00042         -
## SVMrad_pvalue_100 3.4e-05         0.13647
##
## P value adjustment method: none
```

6.4.2. Error de test

```
predic_svmrad_pvalue_100 <- predict(object = svmrad_pvalue_100, newdata =
datos_test)
predic_svmrad_pvalue_50  <- predict(object = svmrad_pvalue_50, newdata =
datos_test)
predic_svmrad_pvalue_25  <- predict(object = svmrad_pvalue_25, newdata =
datos_test)
```

```r
predic_svmrad_s2n_60    <- predict(object = svmrad_s2n_60, newdata =
datos_test)
predic_svmrad_s2n_30     <- predict(object = svmrad_s2n_30, newdata =
datos_test)
predic_svmrad_pca        <- predict(object = svmrad_pca, newdata =
datos_test_pca)
predic_rf_pvalue_100     <- predict(object = rf_pvalue_100, newdata =
datos_test)
predic_rf_pvalue_50      <- predict(object = rf_pvalue_50, newdata =
datos_test)
predic_rf_pvalue_25      <- predict(object = rf_pvalue_25, newdata =
datos_test)
predic_rf_s2n_60        <- predict(object = rf_s2n_60, newdata =
datos_test)
predic_rf_s2n_30         <- predict(object = rf_s2n_30, newdata =
datos_test)
predic_rf_pca            <- predict(object = rf_pca, newdata =
datos_test_pca)
predic_nnet_s2n_60      <- predict(object = nnet_s2n_60, newdata =
datos_test)
predic_nnet_s2n_30       <- predict(object = nnet_s2n_30, newdata =
datos_test)
predic_nnet_pvalue_100   <- predict(object = nnet_pvalue_100, newdata =
datos_test)
predic_nnet_pvalue_50   <- predict(object = nnet_pvalue_50, newdata =
datos_test)
predic_nnet_pvalue_25   <- predict(object = nnet_pvalue_25, newdata =
datos_test)

predicciones <- data.frame(
      SVMrad_pvalue_100 =predic_svmrad_pvalue_100,
      SVMrad_pvalue_50  = predic_svmrad_pvalue_50,
      SVMrad_pvalue_25  = predic_svmrad_pvalue_25,
      SVMrad_s2n_60     = predic_svmrad_s2n_60,
      SVMrad_s2n_30     = predic_svmrad_s2n_30,
      SVMrad_pca        = predic_svmrad_pca,
      RF_pvalue_100     = predic_rf_pvalue_100,
      RF_pvalue_50      = predic_rf_pvalue_50,
      RF_pvalue_25      = predic_rf_pvalue_25,
      RF_s2n_60         = predic_rf_s2n_60,
      RF_s2n_30         = predic_rf_s2n_30,
      RF_pca            = predic_rf_pca,
      NNET_s2n_60       = predic_nnet_s2n_60,
      NNET_s2n_30       = predic_nnet_s2n_30,
      NNET_pvalue_100   = predic_nnet_pvalue_100,
      NNET_pvalue_50    = predic_nnet_pvalue_50,
      NNET_pvalue_25    = predic_nnet_pvalue_25,
      valor_real        = datos_test$type
    )
```

```r
predicciones %>% head()

calculo_accuracy <- function(x, y){
  return(mean(x == y))
}

accuracy_test <- map_dbl(.x = predicciones[, -17],
                         .f = calculo_accuracy,
                         y = predicciones[, 17]) %>%
                 as.data.frame() %>%
                 dplyr::rename(accuracy_test = ".") %>%
                 tibble::rownames_to_column(var = "modelo") %>%
                 arrange(desc(accuracy_test))

metricas_resamples %>%
  dplyr::group_by(modelo, metrica) %>%
  summarise(media = mean(valor)) %>%
  spread(key = metrica, value = media) %>%
  dplyr::select(accuracy_validacion = Accuracy) %>%
  left_join(accuracy_test, by = "modelo") %>%
  arrange(desc(accuracy_test))

metricas_resamples %>%
  dplyr::group_by(modelo, metrica) %>%
  summarise(media = mean(valor)) %>%
  spread(key = metrica, value = media) %>%
  dplyr::select(accuracy_validacion = Accuracy) %>%
  left_join(accuracy_test, by = "modelo") %>%
  gather(key = "datos", value = "accuracy", -modelo) %>%
  ggplot(aes(x = reorder(modelo, accuracy), y = accuracy,
             color = datos, label = round(accuracy, 2))) +
  geom_point(size = 9) +
  ylim(0, 1) +
  scale_color_manual(values = c("orangered2", "gray50")) +
  geom_text(color = "white", size = 2.5) +
  # Accuracy basal
  geom_hline(yintercept = 0.156, linetype = "dashed") +
  annotate(geom = "text", y = 0.25, x = 12, label = "Accuracy basal") +
  coord_flip() +
  labs(title = "Accuracy promedio de validación y test",
       x = "modelo") +
  theme_bw() +
  theme(legend.position = "bottom")
```




```
predicciones %>% dplyr::select(SVMrad_pvalue_25, valor_real) %>%
dplyr::filter(SVMrad_pvalue_25 != valor_real)
```

```
predicciones %>% dplyr::select(NNET_pvalue_50, valor_real) %>%
dplyr::filter(NNET_pvalue_50 != valor_real)
```

```
predicciones %>% dplyr::select(RF_pvalue_50, valor_real) %>%
dplyr::filter(RF_pvalue_50 != valor_real)
```

```
confusionMatrix(data = predicciones$SVMrad_pvalue_25, reference =
as.factor(datos_test$type))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIS PP RR SP
##        CIS   2  0  1  0
##        PP    0  0  0  0
##        RR    1  1  5  1
##        SP    0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.6364
##                  95% CI : (0.3079, 0.8907)
##     No Information Rate : 0.5455
##     P-Value [Acc > NIR] : 0.3853
##
##                   Kappa : 0.3125
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: CIS Class: PP Class: RR Class: SP
```

```
## Sensitivity              0.6667   0.00000    0.8333   0.00000
## Specificity              0.8750   1.00000    0.4000   1.00000
## Pos Pred Value           0.6667       NaN    0.6250       NaN
## Neg Pred Value           0.8750   0.90909    0.6667   0.90909
## Prevalence               0.2727   0.09091    0.5455   0.09091
## Detection Rate           0.1818   0.00000    0.4545   0.00000
## Detection Prevalence     0.2727   0.00000    0.7273   0.00000
## Balanced Accuracy        0.7708   0.50000    0.6167   0.50000
```

La matriz de confusión muestra que el modelo clasifica peor unos tipos que otros.

```
confusionMatrix(data = predicciones$NNET_pvalue_50, reference =
as.factor(datos_test$type))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIS PP RR SP
##        CIS   2  0  0  0
##        PP    0  0  1  0
##        RR    1  1  5  1
##        SP    0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.6364
##                  95% CI : (0.3079, 0.8907)
##     No Information Rate : 0.5455
##     P-Value [Acc > NIR] : 0.3853
##
##                   Kappa : 0.3333
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: CIS Class: PP Class: RR Class: SP
## Sensitivity              0.6667   0.00000    0.8333   0.00000
## Specificity              1.0000   0.90000    0.4000   1.00000
## Pos Pred Value           1.0000   0.00000    0.6250       NaN
## Neg Pred Value           0.8889   0.90000    0.6667   0.90909
## Prevalence               0.2727   0.09091    0.5455   0.09091
## Detection Rate           0.1818   0.00000    0.4545   0.00000
## Detection Prevalence     0.1818   0.09091    0.7273   0.00000
## Balanced Accuracy        0.8333   0.45000    0.6167   0.50000
```

```
confusionMatrix(data = predicciones$RF_pvalue_50, reference =
as.factor(datos_test$type))
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction CIS PP RR SP
##        CIS   1  0  1  0
##        PP    0  0  0  0
##        RR    2  1  5  1
##        SP    0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5455
##                  95% CI : (0.2338, 0.8325)
##     No Information Rate : 0.5455
##     P-Value [Acc > NIR] : 0.6214
##
##                   Kappa : 0.0984
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: CIS Class: PP Class: RR Class: SP
## Sensitivity             0.33333   0.00000    0.8333   0.00000
## Specificity             0.87500   1.00000    0.2000   1.00000
## Pos Pred Value          0.50000       NaN    0.5556       NaN
## Neg Pred Value          0.77778   0.90909    0.5000   0.90909
## Prevalence              0.27273   0.09091    0.5455   0.09091
## Detection Rate          0.09091   0.00000    0.4545   0.00000
## Detection Prevalence    0.18182   0.00000    0.8182   0.00000
## Balanced Accuracy       0.60417   0.50000    0.5167   0.50000
```

6.4.3. Mejor modelo

7. Model ensemble (stacking)

```
moda <- function(x, indice_mejor_modelo){
  tabla_freq <- table(x)
  freq_maxima <- max(tabla_freq)
  if(sum(tabla_freq == freq_maxima) > 1) {
    # En caso de empate, se devuelve la predicción que ocupa el índice
del mejor modelo
    return(x[indice_mejor_modelo])
  }
  return(names(which.max(table(x))))
}

predicciones_ensemble <- predicciones %>%
  dplyr::select(SVMrad_pvalue_100, NNET_pvalue_100, NNET_pvalue_50,
RF_pvalue_50, RF_pvalue_25) %>%
  mutate(moda = apply(X = dplyr::select(.data =
predicciones,SVMrad_pvalue_100, NNET_pvalue_100, NNET_pvalue_50,
RF_pvalue_50, RF_pvalue_25),
```

```
                        MARGIN = 1,
                        FUN = moda,
                        indice_mejor_modelo = 1))

predicciones_ensemble %>% head()

mean(predicciones_ensemble$moda == datos_test$type)
```

```
## [1] 0.6363636
```

8. Clustering

```
library(factoextra)

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

# 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$type, sep = "_")

# Se emplean únicamente los genes filtrados
datos_clustering <- datos_clustering %>% dplyr::select(type,
filtrado_anova_pvalue_100)

# 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 DENDOGRAMA

# 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 6 colores, uno para cada tipo
colores <- colores[1:6]
```

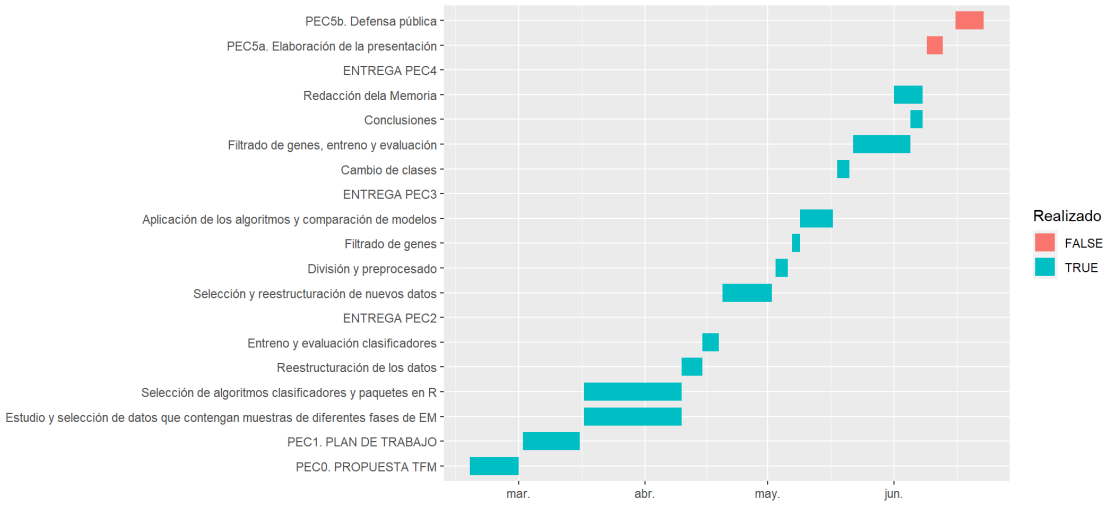
```r
# Se asigna a cada tipo de tumor uno de los colores. Para conseguirlo de
forma
# rápida, se convierte la variable tipo_tumor en factor y se emplea su
codificación
# numérica interna para asignar los colores.
colores <- colores[as.factor(datos_clustering$type)]

# 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 = 0.2,
          lwd = 0.1,
          main = "Linkage completo",
          type = "circular")
```

Actualización del cronograma.

La nueva planificación queda así:

Tarea	Inicio	Fin	Dificultad
PEC0. PROPUESTA TFM	2021-02-17	2021-03-01	7
PEC1. PLAN DE TRABAJO	2021-03-02	2021-03-16	7
Estudio y selección de datos que contengan muestras de diferentes fases de EM	2021-03-17	2021-04-10	6
Selección de algoritmos clasificadores y paquetes	2021-	2021-	5

en R	03-17	04-10	
Reestructuración de los datos	2021-04-10	2021-04-15	6
Entreno y evaluación clasificadores	2021-04-15	2021-04-19	8
ENTREGA PEC2	2021-04-19	2021-04-19	NA
Selección y reestructuración de nuevos datos	2021-04-20	2021-05-02	9
División y preprocesado	2021-05-03	2021-05-06	6
Filtrado de genes	2021-05-07	2021-05-09	8
Aplicación de los algoritmos y comparación de modelos	2021-05-09	2021-05-17	8
ENTREGA PEC3	2021-05-17	2021-05-17	NA
Cambio de clases	2021-05-18	2021-05-21	5
Filtrado de genes, entreno y evaluación	2021-05-22	2021-06-05	6
Conclusiones	2021-06-05	2021-06-08	8
Redacción dela Memoria	2021-06-01	2021-06-08	7
ENTREGA PEC4	2021-06-08	2021-06-08	NA
PEC5a. Elaboración de la presentación	2021-06-9	2021-06-13	NA
PEC5b. Defensa pública	2021-06-16	2021-06-23	NA


PEC5b. Defensa pública
PEC5a. Elaboración de la presentación
ENTREGA PEC4
Redacción dela Memoria
Conclusiones
Filtrado de genes, entreno y evaluación
Cambio de clases
ENTREGA PEC3
Aplicación de los algoritmos y comparación de modelos
Filtrado de genes
División y preprocesado
Selección y reestructuración de nuevos datos
ENTREGA PEC2
Entreno y evaluación clasificadores
Reestructuración de los datos
Selección de algoritmos clasificadores y paquetes en R
Estudio y selección de datos que contengan muestras de diferentes fases de EM
PEC1. PLAN DE TRABAJO
PEC0. PROPUESTA TFM
mar.
abr.
may.
jun.
Realizado
FALSE
TRUE