codigo6clases

Amelia Martínez Sequera

codigo6clases

Amelia Martínez Sequera

13/5/2021

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

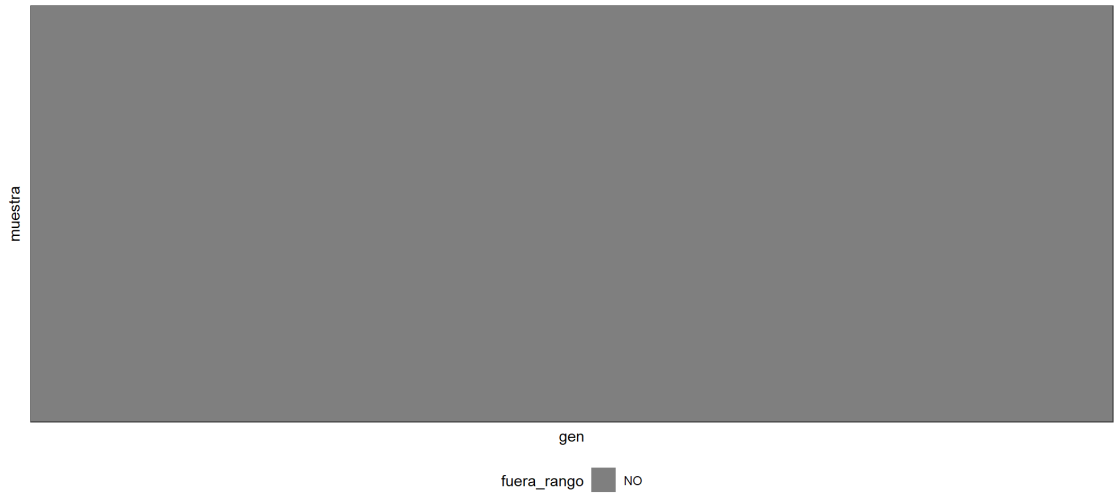
Se realizan una serie de modificaciones para almacenar la información en un dataframe en el que cada fila representa una muestra y las columnas contienen la información relativa a ellas (información descriptiva sobre el tipo de EM y la expresión de los genes).

```
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.

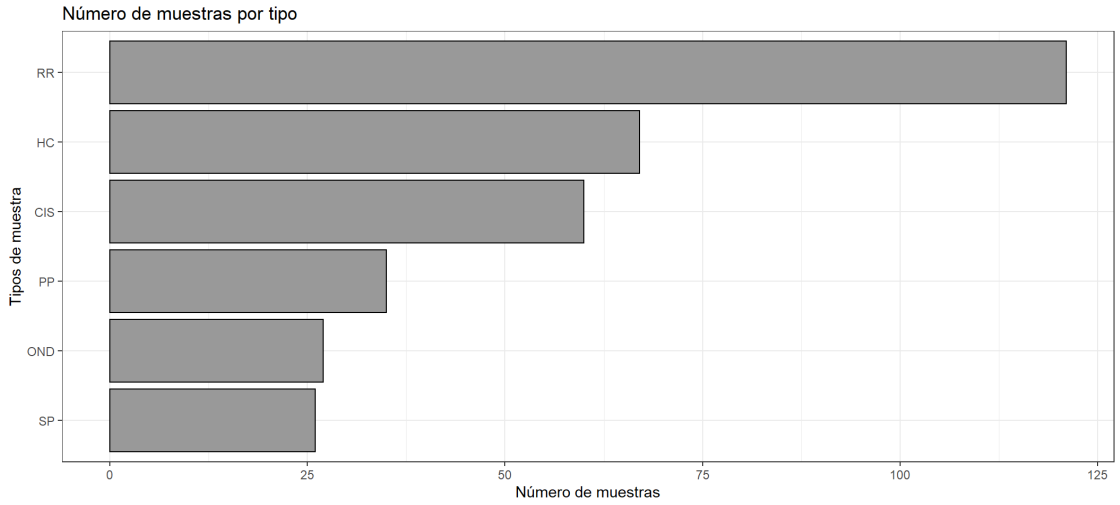
```
datos_long <- datos2 %>% gather(key= "gen",  value= "expresion", -
muestra) %>%
  mutate(fuera_rango = if_else(expresion< 10| expresion> 34,
                               "SI", "NO"))
# Proporción de valores por debajo del mínimo de detección
nrow(datos_long %>% filter(expresion < 10)) / nrow(datos_long)

## [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())
```



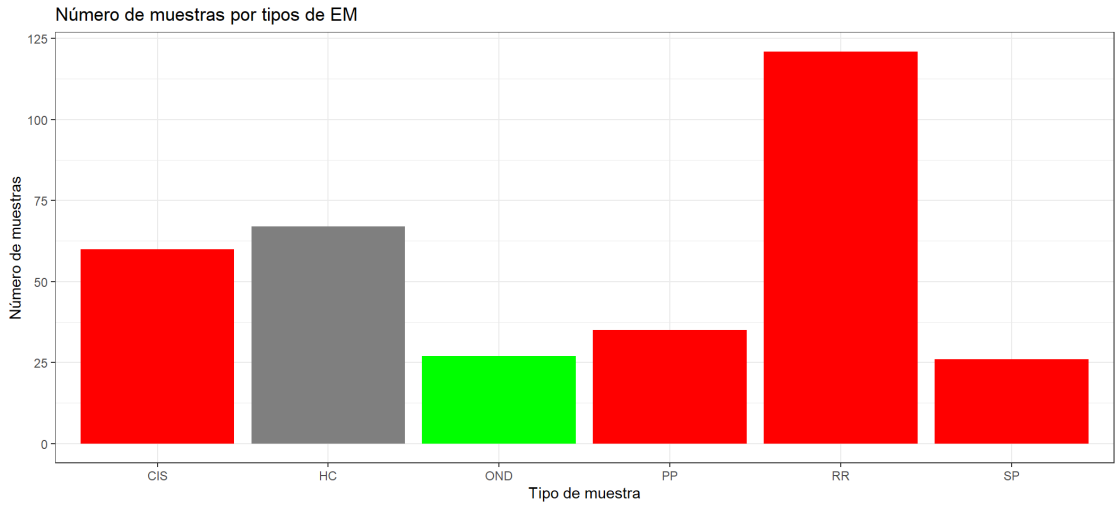
3.4. Exploración de los datos.

3.4.1. Cantidad y tipo de muestras.

```
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"

# 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 )
```

Este tipo de reparto estratificado asegura que el conjunto de entrenamiento y el de test sean similares en cuanto a la variable respuesta, sin embargo, no garantiza que ocurra lo mismo con los predictores. Es importante tenerlo en cuenta sobre todo cuando los predictores son cualitativos.

El tipo más frecuente es RR (36%), este es el porcentaje aproximado de aciertos que se espera si siempre se predice type = “RR”. Por lo tanto, este es el porcentaje de aciertos (accuracy) que deben ser capaces de superar los modelos predictivos para considerarse mínimamente útiles.

Aciertos si se emplea la clase mayoritaria como predictor:

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

```
## [1] 0.3592593
```

4.2. Preprocesado.

El preprocesado de datos engloba aquellas transformaciones hechas sobre los datos con la finalidad de que puedan ser aceptados por el algoritmo de machine learning o que mejoren sus resultados. Todo preprocesado de datos debe aprenderse de las observaciones de entrenamiento y luego aplicarse al conjunto de entrenamiento y al de test. Esto es muy importante para no violar la condición de que ninguna información procedente de las observaciones de test puede participar o influir en el ajuste del modelo.

4.2.1. Genes con varianza próxima a cero.

En la mayoría de análisis discriminantes (diferenciación de grupos), el número de observaciones disponibles es mucho mayor que el número de variables, sin embargo, los estudios de expresión genética suelen caracterizarse por justo lo contrario. Por lo general, se dispone de un número bajo de muestras en comparación a los varios miles de genes disponibles como predictores. Esto dificulta en gran medida la creación de modelos predictivos por dos razones. En primer lugar, algunos algoritmos de machine learning, por ejemplo el análisis discriminante lineal (LDA), no puede aplicarse si el número de observaciones es inferior al número de predictores. En segundo lugar, aun cuando todos los genes pueden incorporarse en el modelo (SVM), muchos de ellos no aportan más que ruido al modelo, lo que disminuye su capacidad predictiva cuando se aplica a nuevas observaciones (overfitting). A este problema se le conoce como "alta dimensionalidad".

Una forma de reducir este problema consiste en eliminar aquellos genes cuya expresión apenas varía en el conjunto de observaciones, y que, por lo tanto, no aportan información.

La función nearZeroVar() del paquete caret identifica como predictores potencialmente problemáticos aquellos que tienen un único valor (cero varianza) o que cumplen las dos siguientes condiciones:

- Ratio de frecuencias: ratio entre la frecuencia del valor más común y la frecuencia del segundo valor más común. Este ratio tiende a 1 si las frecuencias están equidistribuidas y a valores grandes cuando la frecuencia del valor mayoritario supera por mucho al resto (el denominador es un número decimal pequeño). Valor por defecto freqCut = 95/5.
- Porcentaje de valores únicos: número de valores únicos dividido entre el total de muestras (multiplicado por 100). Este porcentaje se aproxima a cero cuanto mayor es la variedad de valores. Valor por defecto uniqueCut = 10.

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

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

Entre los predictores incluidos en el modelo, no se detecta ninguno con varianza cero o próxima a cero.

Estos mismos genes identificados en el conjunto de entrenamiento, tendrían que ser excluidos también del conjunto de test.

Si bien la eliminación de predictores no informativos podría considerarse un paso propio del proceso de selección de predictores, dado que consiste en un filtrado por varianza, tiene que realizarse antes de estandarizar los datos, ya que después, todos los predictores tienen varianza 1.

4.2.2. Estandarización.

Cuando los predictores son numéricos, la escala en la que se miden, así como la magnitud de su varianza, pueden influir en gran medida en el modelo. Muchos algoritmos de machine learning (SVM, redes neuronales, lasso…) son sensibles a esto, de forma que, si no se igualan de alguna forma los predictores, aquellos que se midan en una escala mayor o que tengan más varianza, dominarán el modelo aunque no sean los que más relación tienen con la variable respuesta.

Existen principalmente 2 estrategias para evitarlo:

Centrado: consiste en restarle a cada valor la media del predictor al que pertenece. Si los datos están almacenados en un dataframe, el centrado se consigue restándole a cada valor la media de la columna en la que se encuentra. Como resultado de esta transformación, todos los predictores pasan a tener una media de cero, es decir, los valores se centran en torno al origen.

Normalización: consiste en transformar los datos de forma que todos los predictores estén aproximadamente en la misma escala. Se procede a normalizar la expresión de cada gen (columna) para que tengan media 0 y varianza 1.

```
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 Selección de genes y reducción de dimensionalidad

Es necesario aplicar una estrategia que permita identificar el subconjunto de genes que están realmente relacionados con la variable respuesta, es decir, genes que se expresan de forma diferente en una clase respecto al resto de clases. Incluir un exceso de variables suele conllevar una reducción de la capacidad predictiva del modelo cuando se expone a nuevos datos (overfitting).

Este problema a sido foco de investigación en el ámbito de la bioinformática durante años, lo que ha dado lugar a una amplia variedad de métodos conocidos como feature selection, cuyo objetivo es reducir el número de predictores para mejorar los modelos.

Métodos wrapper

Los métodos wrapper evalúan múltiples modelos, generados mediante la incorporación o eliminación de predictores, con la finalidad de identificar la combinación óptima que consigue maximizar la capacidad del modelo. Pueden entenderse como algoritmos de búsqueda que tratan a los predictores disponibles

como valores de entrada y utilizan una métrica del modelo, por ejemplo, su error de predicción, como objetivo de la optimización.

Métodos de filtrado

Los métodos basados en filtrado evalúan la relevancia de los predictores fuera del modelo para, posteriormente, incluir únicamente aquellos que pasan un determinado criterio. Se trata por lo tanto de analizar la relación que tiene cada predictor con la variable respuesta. Por ejemplo, en problemas de clasificación con predictores continuos, se puede aplicar un ANOVA a cada predictor para identificar aquellos que varían dependiendo de la variable respuesta. Finalmente, se incorporan al modelo aquellos predictores con un p-value inferior a un determinado límite o los n mejores.

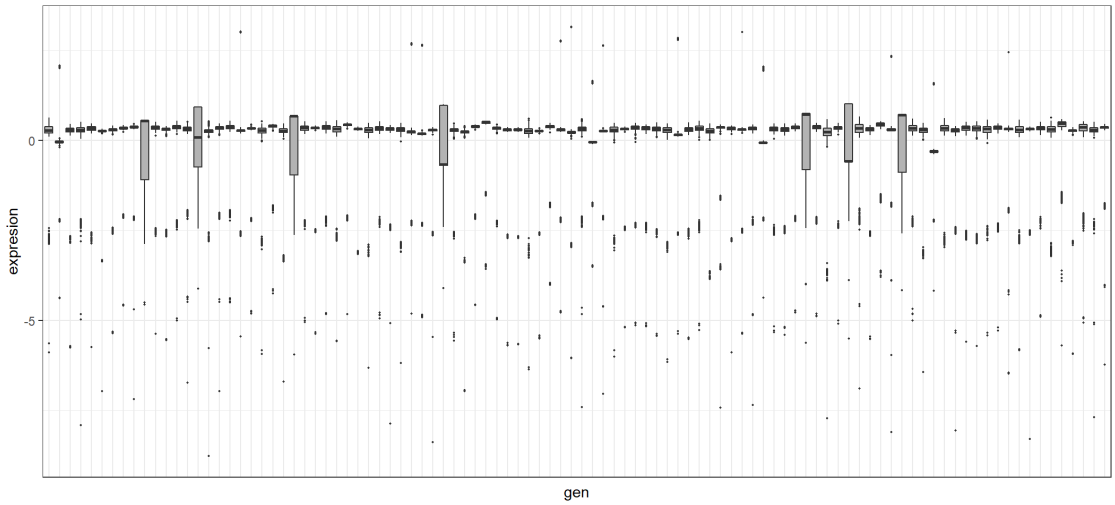
Ambas estrategias, wrapper y filtrado, tienen ventajas y desventajas. Los métodos de filtrado son computacionalmente más rápidos, por lo que suelen ser la opción factible cuando hay cientos o miles de predictores, sin embargo, el criterio de selección no está directamente relacionada con la efectividad del modelo. Además, en la mayoría de casos, los métodos de filtrado evalúan cada predictor de forma individual, por lo que no contemplan interacciones y pueden incorporar predictores redundantes (correlacionados). Los métodos wrapper, además de ser computacionalmente más costosos, para evitar overfitting, necesitan recurrir a validación cruzada o bootstrapping, por lo que requieren un número alto de observaciones. A pesar de ello, si se cumplen las condiciones, suelen conseguir una mejor selección.

En este caso, al existir varios miles de posibles predictores, se recurre a métodos de filtrado.

5.1. Anova p-value

El contraste de hipótesis ANOVA compara la media de una variable continua entre dos o más grupos. Para este estudio, la idea es que el ANOVA permita identificar aquellos genes cuya expresión varía significativamente entre los distintos tipos de EM. Dos de las condiciones para que este test de hipótesis sea válido son: que la variable respuesta (nivel de expresión génica) se distribuya de forma normal y que tenga varianza constante en todos los grupos.

```
# 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())
```



Un rápido análisis gráfico muestra que la expresión de los genes no se distribuye de forma normal ni con varianza constante. Esto significa que los resultados del ANOVA no son precisos, por lo que no se deben emplear los p-value para determinar significancia estadística. Sin embargo, sí pueden resultar útiles como criterio para ordenar los genes de mayor a menor potencial importancia.

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_1692511 ILMN_1767139 ILMN_1766171 ILMN_1814230 ILMN_1655935
ILMN_1753468
## 5.493650e-07 1.554090e-06 2.090130e-06 2.642613e-06 2.735263e-06
2.887171e-06
## ILMN_1678863 ILMN_1723007 ILMN_1786429 ILMN_1659786
## 3.437748e-06 4.238575e-06 6.620767e-06 1.018827e-05
```

Para evitar que la selección de genes esté excesivamente influenciada por los datos de entrenamiento, y así minimizar el riesgo de overfitting, se implementa un proceso de bootstraping. El algoritmo seguido es el siguiente:

1. Para cada iteración de bootstraping:

1.1 Se genera una nueva pseudo-muestra por muestreo repetido con reposición, del mismo tamaño que la muestra original.

1.2 Se calcula el p-value asociado a cada gen mediante un ANOVA.

2. Se calcula el p-value promedio de cada gen.

3. Se seleccionan los top n genes con menor p-value promedio.

VERSIÓN NO PARALELIZADA

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

```
# 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)

Otra forma de identificar genes característicos de un tipo de tumor es ordenándolos acorde al valor de estadístico Signal-to-Noise (S2N). Este estadístico se calcula con la siguiente ecuación:

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

Cuanto mayor es la diferencia entre la expresión promedio de un gen en un grupo respecto a los demás, mayor es el valor absoluto de S2N. Puede considerarse que, para un determinado grupo, los genes con un valor alto de S2N son buenos representantes.

El algoritmo seguido para calcular los valores S2N es:

Para cada grupo i:

- Se separan los valores del grupo i y los del resto de grupos en dos dataframe distintos.
- Se calcula la media y la desviación típica de cada gen en el grupo i.
- Se calcula la media y la desviación típica de cada gen en resto de grupos (de forma conjunta).
- Empleando las medias y desviaciones típicas calculadas en los dos puntos anteriores, se calcula el estadístico Signal-to-Noise de cada gen para el grupo i.

```
# 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
}
```

Como resultado de este algoritmo, se ha obtenido el valor del estadístico Signal-to-Noise para cada uno de los genes, en cada uno de los tipos de tumor. Los resultados se han almacenado en una lista. A continuación, se seleccionan los 10 genes con mayor valor absoluto en cada uno de los grupos.

```
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)
```

Si se cumple que el estadístico signal-to-noise es capaz de identificar en cada grupo genes cuya expresión es particularmente alta o baja en comparación al resto de grupos (genes representativos del estadío de la enfermedad), cabe esperar que, los genes con mayor valor absoluto signal-to-noise, sean distintos en cada tipo de tumor. Si esto es cierto, la intersección de los top 10 genes de los 6 grupos, debería contener aproximadamente 60 genes.

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

```
## [1] 60
```

```
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 (10x6=60), 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 (5x6 = 30):“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] 26
```

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

```
## [1] 14
```

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

5.4. Reducción de dimensionalidad

Los métodos de reducción de dimensionalidad permiten simplificar la complejidad de espacios muestrales con muchas dimensiones a la vez que conservan su información. Supóngase que existe una muestra con n individuos cada uno con p variables (X1, X2, …, Xp), es decir, el espacio muestral tiene p dimensiones. El objetivo de estos métodos es encontrar un número de factores subyacentes (z<p) que expliquen aproximadamente lo mismo que las p variables originales. Donde antes se necesitaban p valores para caracterizar a cada individuo, ahora bastan z valores. Dos de las técnicas más utilizadas son: PCA y t-SNE.

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 270 samples and 10161 variables
##
```

```
## Pre-processing:
##   - centered (10160)
##   - ignored (1)
##   - principal component signal extraction (10160)
##   - scaled (10160)
##
## PCA needed 251 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

En los siguientes apartados se entrenan diferentes modelos de machine learning con el objetivo de compararlos e identificar el que mejor resultado obtiene clasificando los diferentes tipos de EM. Además, se comparan los diferentes filtrados de genes. Los modelos se entrenan, optimizan y comparan empleando las funcionalidades que ofrece el paquete caret.

Diagrama de los modelos ajustados y los genes empleados

```
library(collapsibleTree)
analisis <- data.frame(
  modelo   = rep(c("SVM", "RandomForest", "Neural Network"), each = 6),
  filtrado = rep(c("Anova p-value 100", "Anova p-value 50", "Anova p-
value 25",
                   "S2N 140", "S2N 70", "PCA"), times = 3)
)

collapsibleTree(df = analisis,
                hierarchy = c("modelo", "filtrado"),
                collapsed = FALSE,
                zoomable = FALSE,
                fill = c("#cacfd2", "#d98880", "#7fb3d5", "#82e0aa",
                         rep(c("#d98880", "#7fb3d5", "#82e0aa"), each =
6)))
```

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:

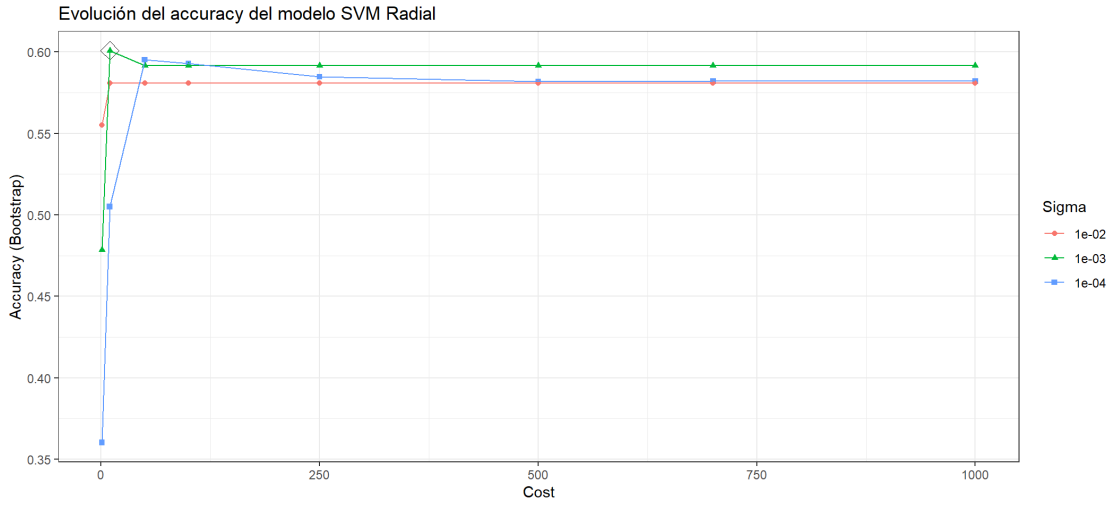
- sigma: coeficiente del kernel radial.
- C: penalización por violaciones del margen del hiperplano.

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
##
## 270 samples
## 100 predictors
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
```

```
##
##    sigma  C     Accuracy   Kappa
##    1e-04     1  0.3602428  0.0000000
##    1e-04    10  0.5051087  0.2780725
##    1e-04    50  0.5952747  0.4594879
##    1e-04   100  0.5928030  0.4644368
##    1e-04   250  0.5848321  0.4577336
##    1e-04   500  0.5818481  0.4539875
##    1e-04   700  0.5820609  0.4543033
##    1e-04  1000  0.5820609  0.4543033
##    1e-03     1  0.4785663  0.2296504
##    1e-03    10  0.6008727  0.4728531
##    1e-03    50  0.5916374  0.4650782
##    1e-03   100  0.5916374  0.4650782
##    1e-03   250  0.5916374  0.4650782
##    1e-03   500  0.5916374  0.4650782
##    1e-03   700  0.5916374  0.4650782
##    1e-03  1000  0.5916374  0.4650782
##    1e-02     1  0.5550270  0.3713299
##    1e-02    10  0.5807382  0.4186668
##    1e-02    50  0.5807382  0.4186668
##    1e-02   100  0.5807382  0.4186668
##    1e-02   250  0.5807382  0.4186668
##    1e-02   500  0.5807382  0.4186668
##    1e-02   700  0.5807382  0.4186668
##    1e-02  1000  0.5807382  0.4186668
##
## 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.
```

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



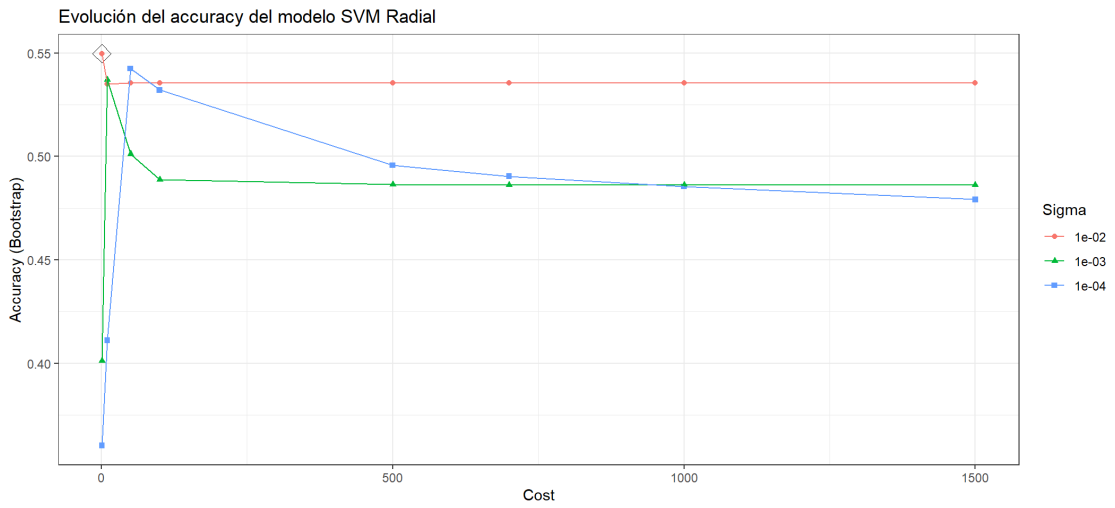
Mejor modelo: sigma = 0.001 C = 10 Accuracy = 0.6008727

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
##
## 270 samples
##  50 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
```

```
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04     1  0.3602428  0.00000000
##   1e-04    10  0.4111302  0.10598697
##   1e-04    50  0.5423603  0.37276089
##   1e-04   100  0.5320570  0.37536627
##   1e-04   500  0.4957256  0.34791063
##   1e-04   700  0.4904270  0.34360729
##   1e-04  1000  0.4853861  0.33920448
##   1e-04  1500  0.4791910  0.33088147
##   1e-03     1  0.4012227  0.08524703
##   1e-03    10  0.5370270  0.38062771
##   1e-03    50  0.5012200  0.35415905
##   1e-03   100  0.4888493  0.34168069
##   1e-03   500  0.4864321  0.33871077
##   1e-03   700  0.4862147  0.33845894
##   1e-03  1000  0.4862147  0.33845894
##   1e-03  1500  0.4862147  0.33845894
##   1e-02     1  0.5497199  0.37643074
##   1e-02    10  0.5350151  0.38669327
##   1e-02    50  0.5354529  0.38746950
##   1e-02   100  0.5354529  0.38746950
##   1e-02   500  0.5354529  0.38746950
##   1e-02   700  0.5354529  0.38746950
##   1e-02  1000  0.5354529  0.38746950
##   1e-02  1500  0.5354529  0.38746950
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 1.
```

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



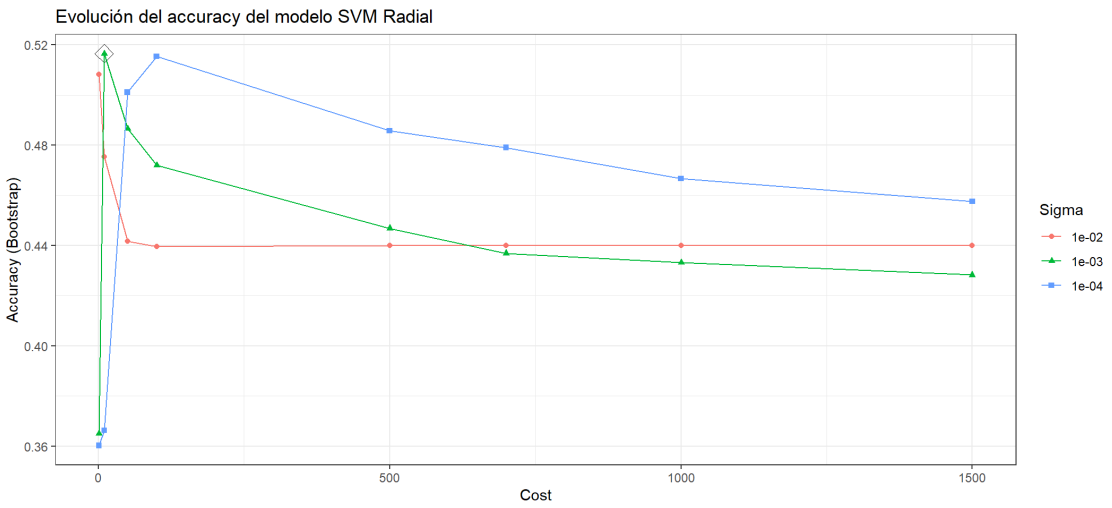
Mejor modelo: sigma = 0.01 C = 1 Accuracy = 0.5497199

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

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
##
## 270 samples
##  25 predictor
```

```
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04     1  0.3602428  0.00000000
##   1e-04    10  0.3662753  0.01442712
##   1e-04    50  0.5011066  0.29360340
##   1e-04   100  0.5153953  0.33335214
##   1e-04   500  0.4857426  0.32463031
##   1e-04   700  0.4790730  0.31925000
##   1e-04  1000  0.4666365  0.30617723
##   1e-04  1500  0.4575804  0.29754591
##   1e-03     1  0.3650338  0.01175423
##   1e-03    10  0.5165393  0.33363916
##   1e-03    50  0.4866490  0.32488232
##   1e-03   100  0.4718972  0.31266383
##   1e-03   500  0.4468684  0.28962477
##   1e-03   700  0.4369227  0.27734013
##   1e-03  1000  0.4331710  0.27453561
##   1e-03  1500  0.4282441  0.26893741
##   1e-02     1  0.5081371  0.31085184
##   1e-02    10  0.4752917  0.31221870
##   1e-02    50  0.4417683  0.27908995
##   1e-02   100  0.4396238  0.27786632
##   1e-02   500  0.4400383  0.27871011
##   1e-02   700  0.4400383  0.27871011
##   1e-02  1000  0.4400383  0.27871011
##   1e-02  1500  0.4400383  0.27871011
##
## 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_pvalue_25, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo SVM Radial") +
  theme_bw()
```



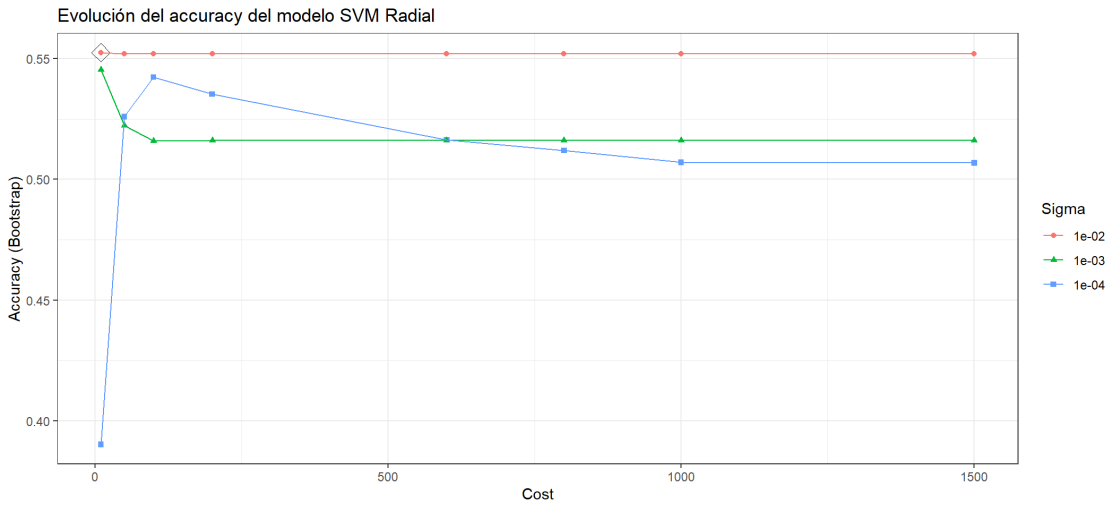
Mejor modelo: Sigma = 1e-03 C = 10
Accuracy = 0.5165393

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)

# 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
##
## 270 samples
##  60 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04    10  0.3901558  0.06483314
##   1e-04    50  0.5259894  0.36088893
##   1e-04   100  0.5422995  0.39991094
##   1e-04   200  0.5352775  0.40055830
##   1e-04   600  0.5163267  0.38130571
##   1e-04   800  0.5119561  0.37699074
##   1e-04  1000  0.5070532  0.37101118
##   1e-04  1500  0.5068353  0.37093457
##   1e-03    10  0.5454351  0.40213630
##   1e-03    50  0.5222715  0.38675061
##   1e-03   100  0.5159504  0.37973899
##   1e-03   200  0.5161513  0.37995483
##   1e-03   600  0.5161513  0.37997512
##   1e-03   800  0.5161513  0.37997512
##   1e-03  1000  0.5161513  0.37997512
##   1e-03  1500  0.5161513  0.37997512
##   1e-02    10  0.5525593  0.40722864
##   1e-02    50  0.5519296  0.40627691
##   1e-02   100  0.5519296  0.40627691
##   1e-02   200  0.5519296  0.40627691
##   1e-02   600  0.5519296  0.40627691
##   1e-02   800  0.5519296  0.40627691
##   1e-02  1000  0.5519296  0.40627691
##   1e-02  1500  0.5519296  0.40627691
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 10.
```

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



Mejor modelo: Sigma = 1e-02 C = 10 Accuracy = 0.5525593

6.1.5. 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(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)

# DEFINICIÓN DEL ENTRENAMIENTO

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

# AJUSTE DEL MODELO
```

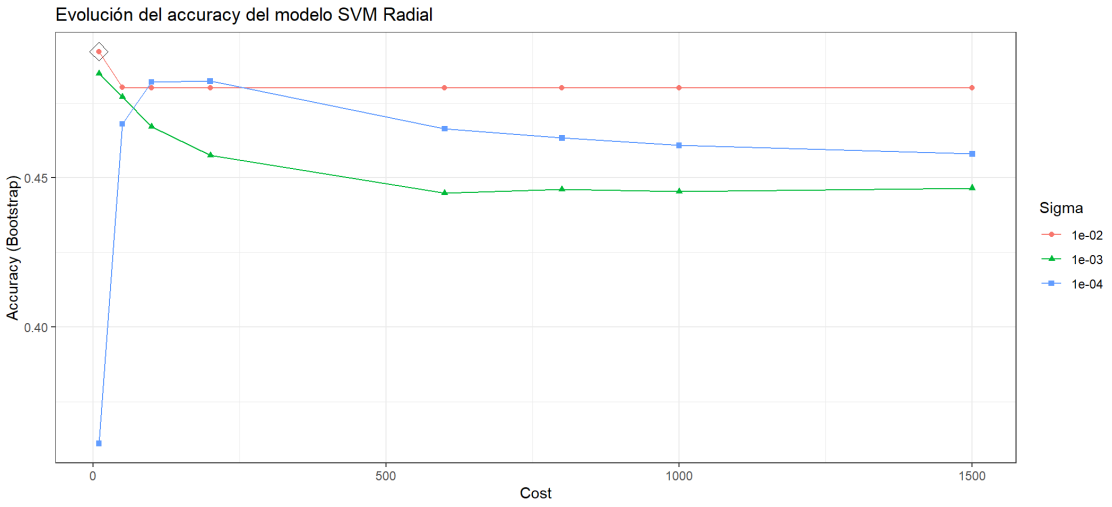
```r
set.seed(342)
svmrad_s2n_30 <- train(
                      form = type ~ .,
                      data = datos_train[c("type", filtrado_s2n_30)],
                      method = "svmRadial",
                      tuneGrid = hiperparametros,
                      metric = "Accuracy",
                      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
##
## 270 samples
##  30 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   1e-04    10  0.3610277  0.002970615
##   1e-04    50  0.4680512  0.242498920
##   1e-04   100  0.4820578  0.291256934
##   1e-04   200  0.4823744  0.314086185
##   1e-04   600  0.4663919  0.311985148
##   1e-04   800  0.4633352  0.309395863
##   1e-04  1000  0.4608344  0.307495059
##   1e-04  1500  0.4579943  0.305742544
##   1e-03    10  0.4848976  0.293610658
##   1e-03    50  0.4770690  0.323323953
##   1e-03   100  0.4671936  0.315353601
##   1e-03   200  0.4575371  0.305481891
##   1e-03   600  0.4449665  0.291863125
##   1e-03   800  0.4462205  0.292613809
##   1e-03  1000  0.4454411  0.291859280
##   1e-03  1500  0.4465895  0.293914466
##   1e-02    10  0.4922995  0.340658177
##   1e-02    50  0.4802920  0.326321831
##   1e-02   100  0.4801315  0.326330750
##   1e-02   200  0.4801315  0.326330750
##   1e-02   600  0.4801315  0.326330750
##   1e-02   800  0.4801315  0.326330750
##   1e-02  1000  0.4801315  0.326330750
##   1e-02  1500  0.4801315  0.326330750
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 10.
```

Mejor modelo: Sigma = 1e-02
C = 10
Accuracy = 0.5525593

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



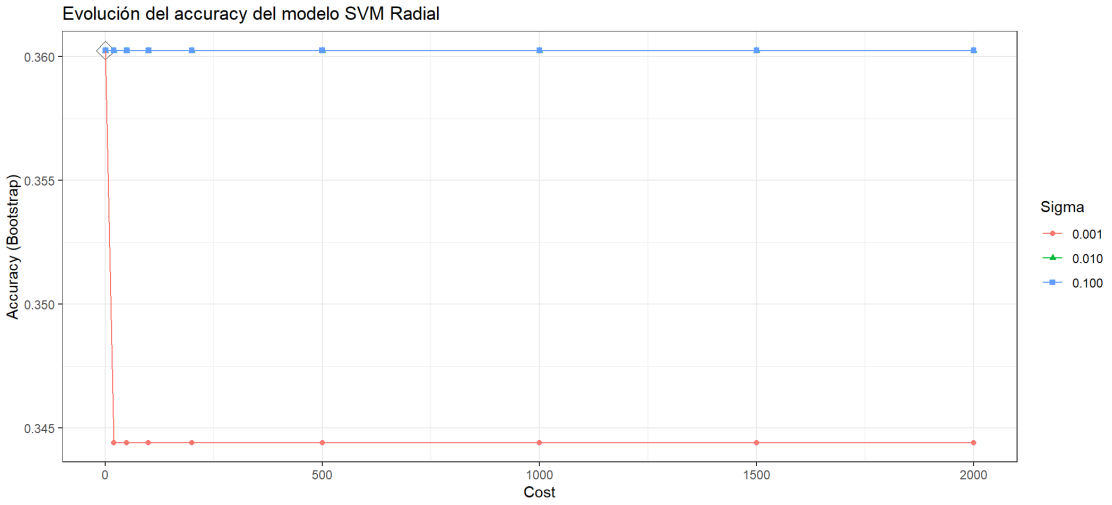
6.1.6. Reducción PCA

```
# 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))
}
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
##
## 270 samples
## 251 predictors
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   sigma  C     Accuracy   Kappa
##   0.001     1  0.3602428   0.0000000000
##   0.001    20  0.3444180  -0.0009278506
##   0.001    50  0.3444180  -0.0009278506
##   0.001   100  0.3444180  -0.0009278506
##   0.001   200  0.3444180  -0.0009278506
##   0.001   500  0.3444180  -0.0009278506
##   0.001  1000  0.3444180  -0.0009278506
##   0.001  1500  0.3444180  -0.0009278506
##   0.001  2000  0.3444180  -0.0009278506
##   0.010     1  0.3602428   0.0000000000
##   0.010    20  0.3602428   0.0000000000
##   0.010    50  0.3602428   0.0000000000
##   0.010   100  0.3602428   0.0000000000
##   0.010   200  0.3602428   0.0000000000
##   0.010   500  0.3602428   0.0000000000
##   0.010  1000  0.3602428   0.0000000000
##   0.010  1500  0.3602428   0.0000000000
##   0.010  2000  0.3602428   0.0000000000
##   0.100     1  0.3602428   0.0000000000
##   0.100    20  0.3602428   0.0000000000
##   0.100    50  0.3602428   0.0000000000
##   0.100   100  0.3602428   0.0000000000
##   0.100   200  0.3602428   0.0000000000
```

```
##   0.100   500  0.3602428   0.0000000000
##   0.100  1000  0.3602428   0.0000000000
##   0.100  1500  0.3602428   0.0000000000
##   0.100  2000  0.3602428   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()
```



Mejor modelo: sigma= 0.1 C = 1 Accuracy = 0.3602428

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.

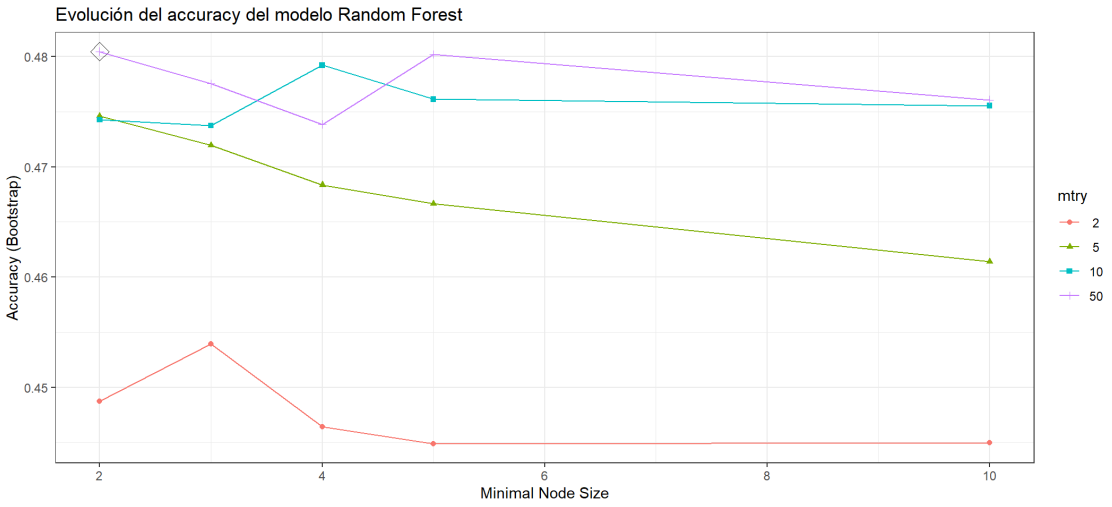
6.2.1. Filtrado por ANOVA p-value 100

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

# 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
##
## 270 samples
## 100 predictors
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    2     2             0.4487288  0.1795024
##    2     3             0.4539310  0.1882573
##    2     4             0.4464389  0.1749160
##    2     5             0.4449005  0.1730261
```

```
##     2    10              0.4449611  0.1721693
##     5     2              0.4746217  0.2291723
##     5     3              0.4719456  0.2253718
##     5     4              0.4683673  0.2198425
##     5     5              0.4666425  0.2170829
##     5    10              0.4614055  0.2087177
##    10     2              0.4742698  0.2348960
##    10     3              0.4737457  0.2345073
##    10     4              0.4792176  0.2439205
##    10     5              0.4761271  0.2388462
##    10    10              0.4755404  0.2362855
##    50     2              0.4804560  0.2580776
##    50     3              0.4775391  0.2534430
##    50     4              0.4738601  0.2498183
##    50     5              0.4801935  0.2579982
##    50    10              0.4760631  0.2505349
##
## 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 = 50, splitrule = gini
##  and min.node.size = 2.
```

```
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()
```



Mejor modelo: mtry = 50, splitrule = gini, min.node.size = 2, accuracy = 0.4804560

6.2.2. Filtrado por ANOVA p-value 50

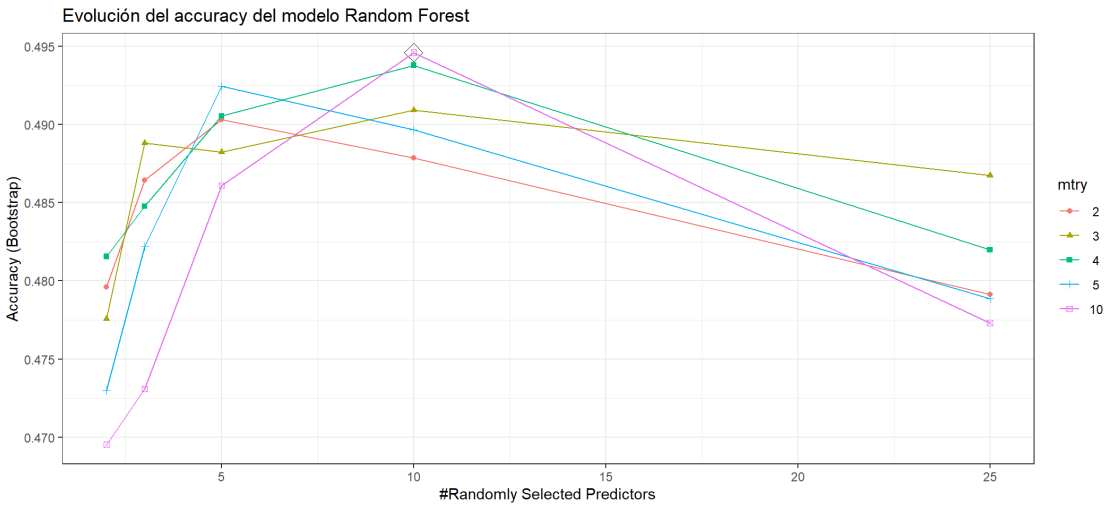
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

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

```r
repeticiones_boot <- 50

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(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
##
## 270 samples
##  50 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    2     2             0.4796146  0.2403289
```

```
##     2     3             0.4775646  0.2353074
##     2     4             0.4815527  0.2421383
##     2     5             0.4729819  0.2293845
##     2    10             0.4695339  0.2232398
##     3     2             0.4864222  0.2556045
##     3     3             0.4887963  0.2580302
##     3     4             0.4847597  0.2519135
##     3     5             0.4821823  0.2475498
##     3    10             0.4730910  0.2338890
##     5     2             0.4902979  0.2663986
##     5     3             0.4882326  0.2646928
##     5     4             0.4905440  0.2664100
##     5     5             0.4924292  0.2686974
##     5    10             0.4860960  0.2598744
##    10     2             0.4878581  0.2702159
##    10     3             0.4909115  0.2732648
##    10     4             0.4937811  0.2790000
##    10     5             0.4896533  0.2722846
##    10    10             0.4945952  0.2790295
##    25     2             0.4791353  0.2658944
##    25     3             0.4867258  0.2773218
##    25     4             0.4819767  0.2703386
##    25     5             0.4788634  0.2656521
##    25    10             0.4773057  0.2651993
##
## 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 = 10, splitrule = gini
##  and min.node.size = 10.
```

```
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()
```



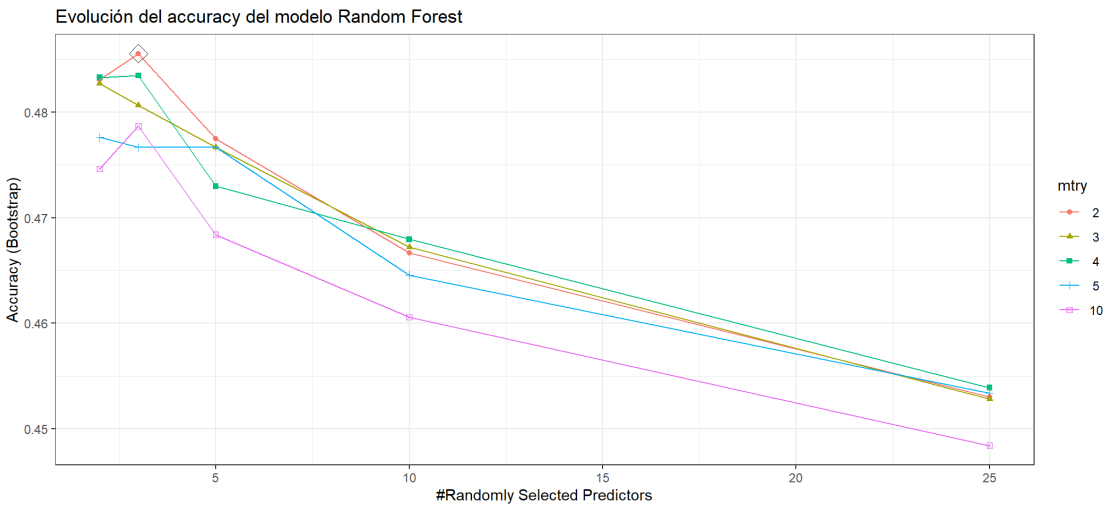
Mejor modelo: mtry = 10, splitrule = gini, min.node.size = 10, accuracy = 0.4945952.

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
##
## 270 samples
##  25 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
```

```
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    2     2             0.4830816  0.2617859
##    2     3             0.4827251  0.2598090
##    2     4             0.4832937  0.2624252
##    2     5             0.4776244  0.2525923
##    2    10             0.4746078  0.2481317
##    3     2             0.4855529  0.2699490
##    3     3             0.4806456  0.2635162
##    3     4             0.4834643  0.2672738
##    3     5             0.4766674  0.2573905
##    3    10             0.4786662  0.2588243
##    5     2             0.4774975  0.2654191
##    5     3             0.4766659  0.2640510
##    5     4             0.4729760  0.2580409
##    5     5             0.4766953  0.2644225
##    5    10             0.4683887  0.2512428
##   10     2             0.4666456  0.2579852
##   10     3             0.4671932  0.2574290
##   10     4             0.4679437  0.2598818
##   10     5             0.4645628  0.2547841
##   10    10             0.4605651  0.2504709
##   25     2             0.4529962  0.2475509
##   25     3             0.4528089  0.2480227
##   25     4             0.4538964  0.2492214
##   25     5             0.4533831  0.2487051
##   25    10             0.4484094  0.2419880
##
## 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 = 3, splitrule = gini
##  and min.node.size = 2.
```

```
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()
```



Mejor modelo: mtry = 3, splitrule = gini, min.node.size = 2, accuracy = 0.4855529.

6.2.4. Filtrado por S2N 60

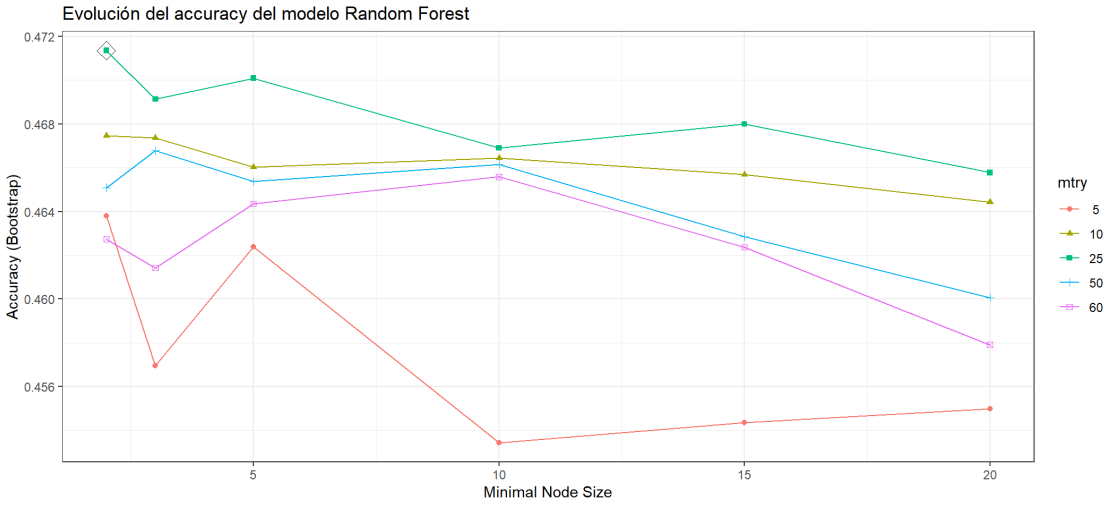
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

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

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(5, 10, 25, 50, 60),
                               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
##
## 270 samples
##  60 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    5     2             0.4637929  0.2211335
##    5     3             0.4569596  0.2099502
##    5     5             0.4623960  0.2177235
##    5    10             0.4534432  0.2023512
##    5    15             0.4543644  0.2055267
##    5    20             0.4549831  0.2058560
##   10     2             0.4674629  0.2348148
##   10     3             0.4673570  0.2344619
##   10     5             0.4660283  0.2307790
##   10    10             0.4664427  0.2316608
##   10    15             0.4656949  0.2330996
##   10    20             0.4644338  0.2336630
##   25     2             0.4713525  0.2514315
##   25     3             0.4691337  0.2478563
##   25     5             0.4700897  0.2499411
##   25    10             0.4669055  0.2448932
##   25    15             0.4679952  0.2477004
##   25    20             0.4657758  0.2469000
##   50     2             0.4650849  0.2494470
##   50     3             0.4667795  0.2512241
##   50     5             0.4653716  0.2493241
##   50    10             0.4661591  0.2514767
##   50    15             0.4628550  0.2473162
##   50    20             0.4600583  0.2456631
##   60     2             0.4627368  0.2482181
##   60     3             0.4614125  0.2462233
##   60     5             0.4643506  0.2506449
##   60    10             0.4655836  0.2528734
```

```
##    60    15             0.4623652  0.2490132
##    60    20             0.4579051  0.2448923
##
## 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 = 25, splitrule = gini
##  and min.node.size = 2.
```

```
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()
```



Mejor modelo: mtry = 25, splitrule = gini, min.node.size = 2, accuracy = 0.4713525

6.2.5. Filtrado por S2N 30

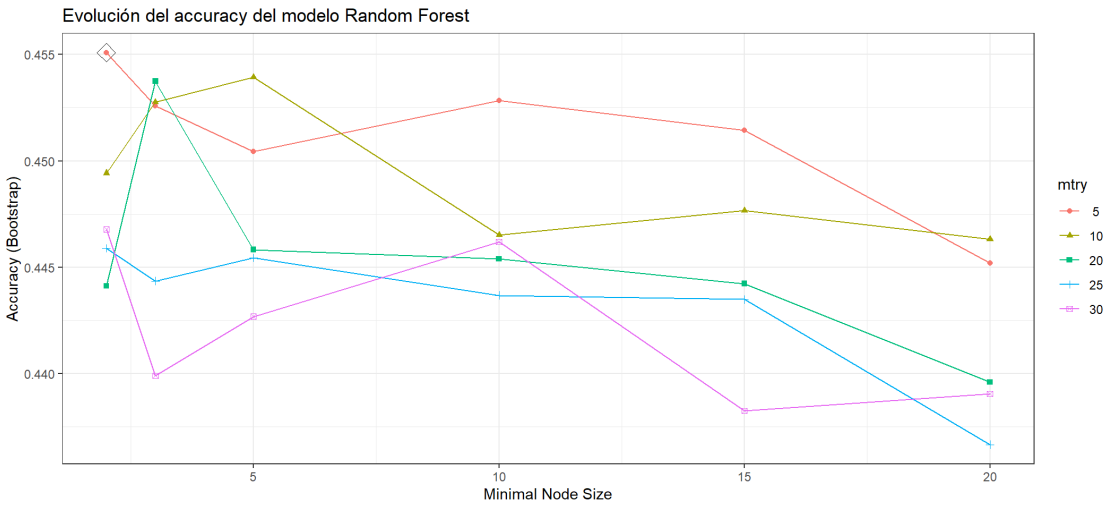
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

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

# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(5, 10, 20, 25, 30),
                               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)
```

```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)
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
##
## 270 samples
##  30 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    5     2             0.4550911  0.2231795
##    5     3             0.4525913  0.2190252
##    5     5             0.4504284  0.2154431
##    5    10             0.4528337  0.2180175
##    5    15             0.4514248  0.2158123
##    5    20             0.4451915  0.2076757
##   10     2             0.4494260  0.2252668
##   10     3             0.4527693  0.2289853
##   10     5             0.4539316  0.2309012
##   10    10             0.4465174  0.2184581
##   10    15             0.4476716  0.2213661
##   10    20             0.4463117  0.2220859
##   20     2             0.4441223  0.2269305
##   20     3             0.4537458  0.2396068
##   20     5             0.4458304  0.2272014
```

```
##    20    10             0.4453971  0.2282553
##    20    15             0.4442323  0.2266739
##    20    20             0.4396079  0.2203939
##    25     2             0.4458938  0.2297946
##    25     3             0.4443531  0.2289030
##    25     5             0.4454415  0.2296557
##    25    10             0.4436821  0.2265909
##    25    15             0.4434924  0.2282691
##    25    20             0.4366681  0.2186925
##    30     2             0.4467854  0.2331435
##    30     3             0.4399007  0.2243024
##    30     5             0.4426839  0.2268988
##    30    10             0.4461990  0.2315888
##    30    15             0.4382579  0.2220565
##    30    20             0.4390597  0.2246027
##
## 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.
```

```r
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()
```



Mejor modelo: mtry = 20, splitrule = gini, min.node.size = 2, accuracy = 0.4760024.

6.2.6. Reducción PCA

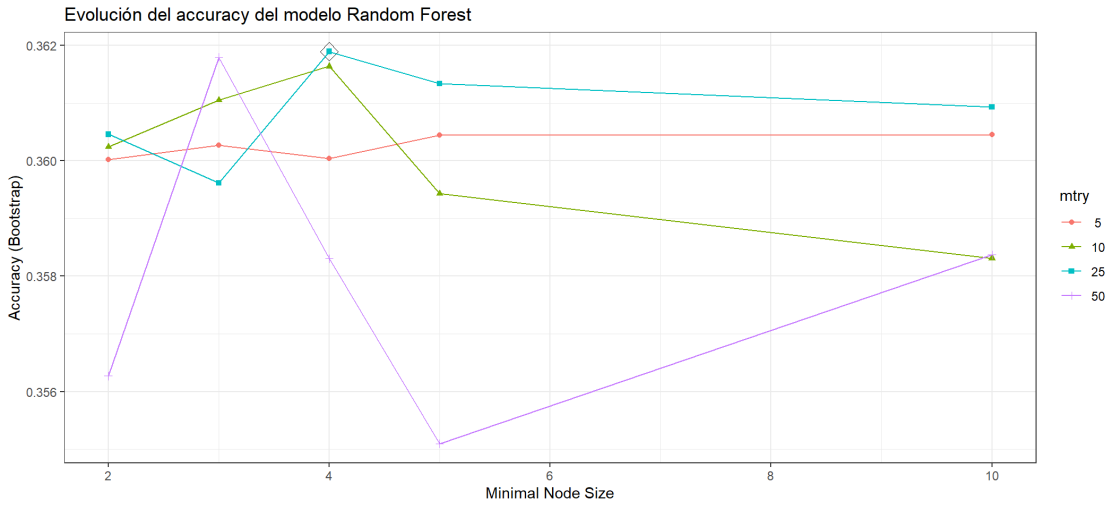
```r
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

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

```r
# Hiperparámetros
hiperparametros <- expand.grid(mtry = c(5, 10, 25, 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_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
##
## 270 samples
## 251 predictors
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  Accuracy   Kappa
##    5     2             0.3600193   0.0004831808
##    5     3             0.3602713   0.0012659163
##    5     4             0.3600360   0.0003413622
```

```
##     5     5             0.3604448   0.0012822341
##     5    10             0.3604524   0.0016453132
##    10     2             0.3602426   0.0023927788
##    10     3             0.3610516   0.0044255626
##    10     4             0.3616359   0.0047504884
##    10     5             0.3594323   0.0019453312
##    10    10             0.3583127  -0.0004488956
##    25     2             0.3604579   0.0076254812
##    25     3             0.3596128   0.0075123923
##    25     4             0.3618922   0.0101607251
##    25     5             0.3613391   0.0095917301
##    25    10             0.3609326   0.0086880542
##    50     2             0.3562713   0.0133189383
##    50     3             0.3617904   0.0196912025
##    50     4             0.3583070   0.0136566461
##    50     5             0.3551003   0.0117223998
##    50    10             0.3583708   0.0160144502
##
## 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 = 25, 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()
```



Mejor modelo: mtry = 25, splitrule = gini, min.node.size = 4, accuracy = 0.3618922

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.

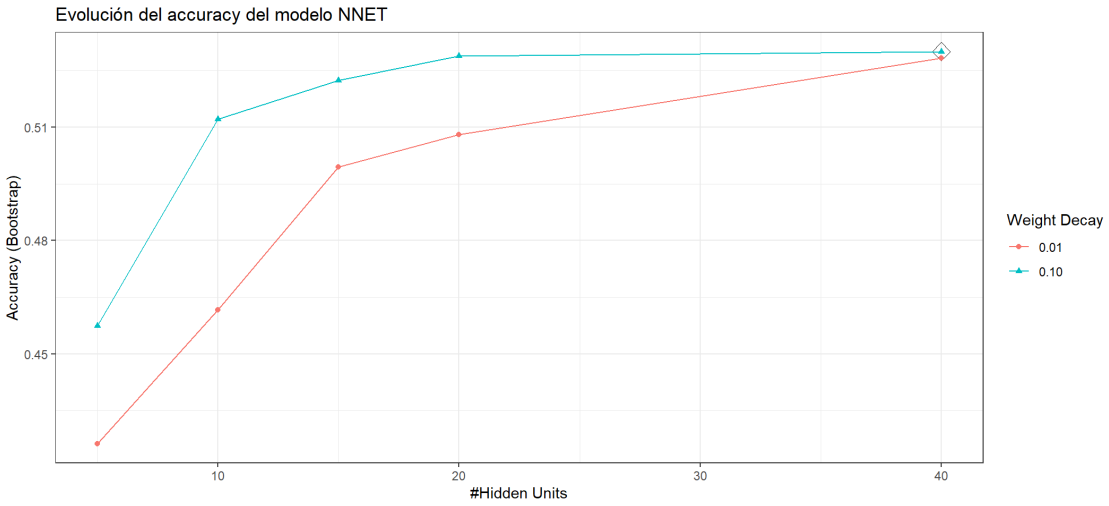
6.3.1. Filtrado por S2N 60

```
# 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
##
## 270 samples
##  60 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.4262684  0.2650201
##    5    0.10   0.4574041  0.3030057
##   10    0.01   0.4615682  0.3072501
##   10    0.10   0.5120407  0.3694175
##   15    0.01   0.4994776  0.3563600
##   15    0.10   0.5222954  0.3821879
##   20    0.01   0.5079832  0.3675462
##   20    0.10   0.5287912  0.3912607
##   40    0.01   0.5282009  0.3927231
##   40    0.10   0.5299830  0.3950993
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 40 and decay = 0.1.
```

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



Mejor modelo: size = 40, decay = 0.1, accuracy = 0.5299830

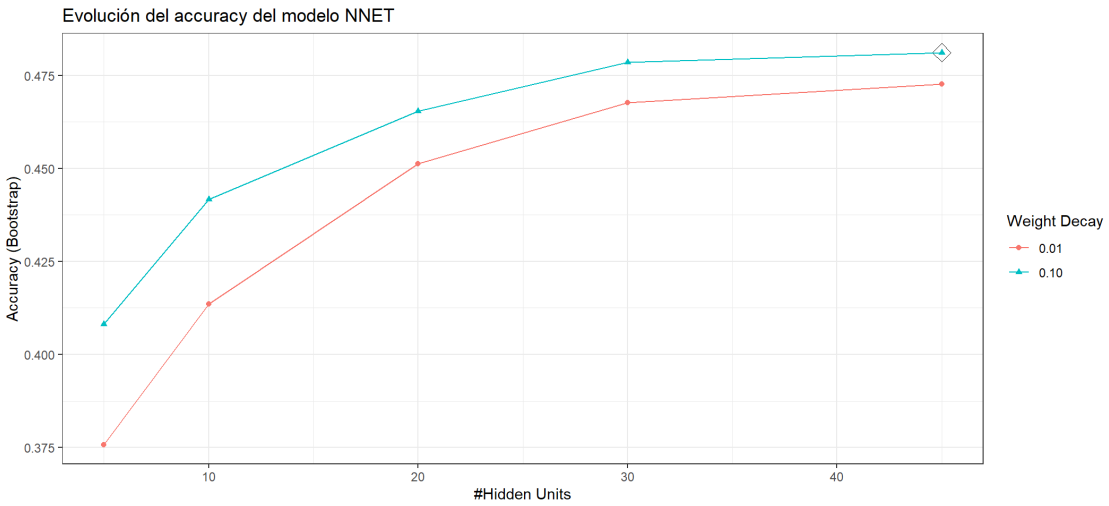
6.3.2. Filtrado por S2N 30

```
# 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
##
## 270 samples
##  30 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
```

```
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.3757995  0.2026701
##    5    0.10   0.4081692  0.2378656
##   10    0.01   0.4135330  0.2410005
##   10    0.10   0.4417153  0.2770219
##   20    0.01   0.4512525  0.2890515
##   20    0.10   0.4654131  0.3060371
##   30    0.01   0.4676573  0.3103278
##   30    0.10   0.4785124  0.3249307
##   45    0.01   0.4726634  0.3181591
##   45    0.10   0.4811533  0.3288367
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 45 and decay = 0.1.
```

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



Mejor modelo: size = 45, decay = 0.1, accuracy = 0.5351088

6.3.3. Filtrado por ANOVA p-value 100

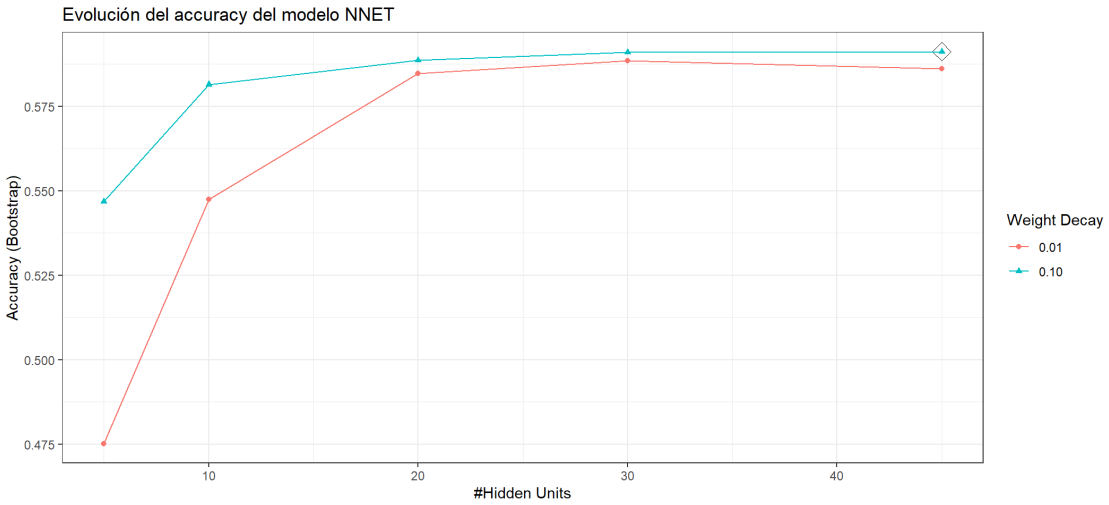
```
# PARALELIZACIÓN DE PROCESO
registerDoParallel(cores = 3)

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

# Hiperparámetros
```

```r
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_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
##
## 270 samples
## 100 predictors
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.4751625  0.3254923
```

```
##     5    0.10   0.5467786  0.4089357
##    10    0.01   0.5474042  0.4127573
##    10    0.10   0.5814581  0.4519679
##    20    0.01   0.5846101  0.4592027
##    20    0.10   0.5886516  0.4626063
##    30    0.01   0.5884037  0.4637620
##    30    0.10   0.5909606  0.4655922
##    45    0.01   0.5860408  0.4610434
##    45    0.10   0.5911956  0.4660712
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 45 and decay = 0.1.
```

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



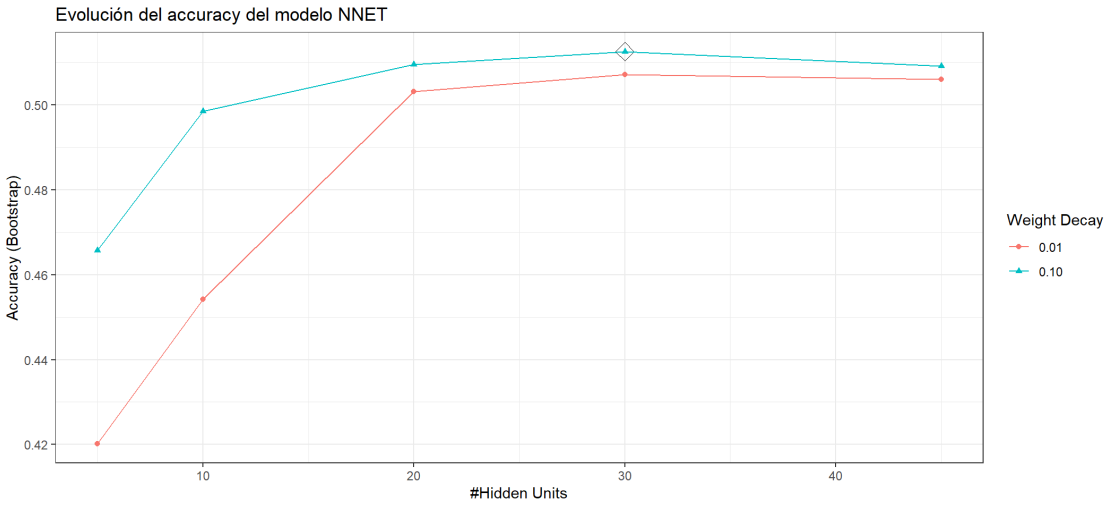
6.3.4. Filtrado por ANOVA p-value 50

```
# 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)
```

```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)
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
##
## 270 samples
##  50 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.4202271  0.2561079
##    5    0.10   0.4657299  0.3099334
##   10    0.01   0.4541963  0.2947754
##   10    0.10   0.4985322  0.3485582
##   20    0.01   0.5031407  0.3563769
##   20    0.10   0.5095571  0.3630907
##   30    0.01   0.5071134  0.3629428
##   30    0.10   0.5125797  0.3684155
##   45    0.01   0.5060629  0.3612184
##   45    0.10   0.5091215  0.3640913
```

```
##
## 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_pvalue_50, highlight = TRUE) +
  labs(title = "Evolución del accuracy del modelo NNET") +
  theme_bw()
```



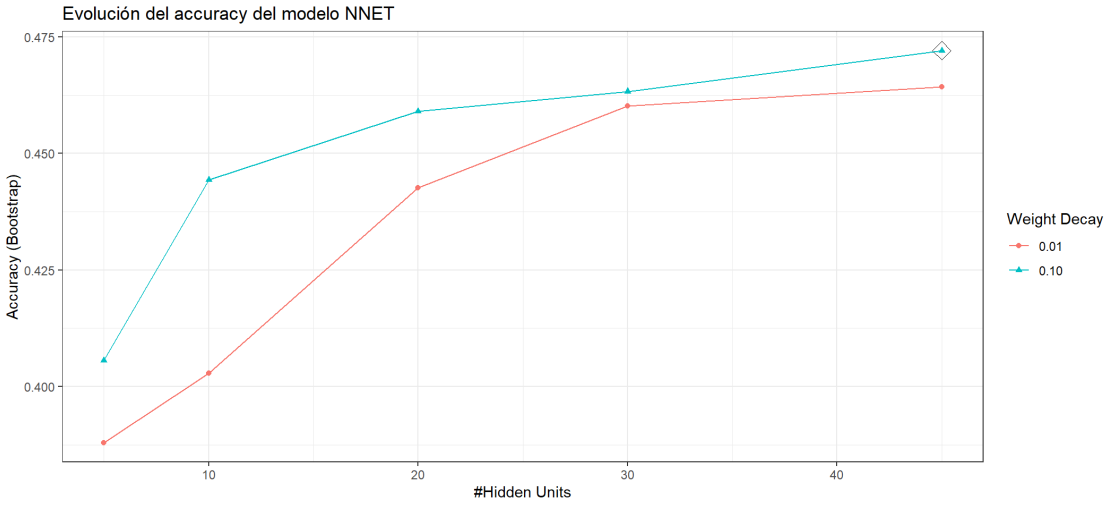
6.3.5. Filtrado por ANOVA p-value 25

```
# 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)
```

```r
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
##
## 270 samples
##  25 predictor
##   6 classes: 'CIS', 'HC', 'OND', 'PP', 'RR', 'SP'
##
## No pre-processing
## Resampling: Bootstrapped (50 reps)
## Summary of sample sizes: 270, 270, 270, 270, 270, 270, ...
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.01   0.3879672  0.2123508
##    5    0.10   0.4055673  0.2347690
##   10    0.01   0.4028316  0.2327504
##   10    0.10   0.4443131  0.2805997
##   20    0.01   0.4426236  0.2798421
##   20    0.10   0.4590862  0.2993045
##   30    0.01   0.4602110  0.3015357
##   30    0.10   0.4633268  0.3041165
##   45    0.01   0.4642611  0.3059412
##   45    0.10   0.4720984  0.3169544
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 45 and decay = 0.1.
```

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

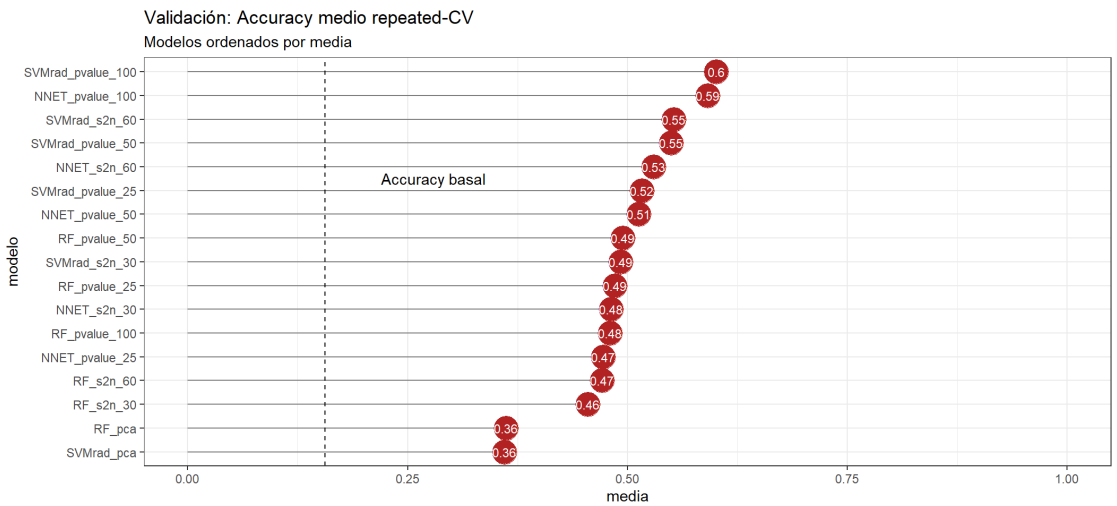


6.4. Comparación de modelos

6.4.1. Error de validación

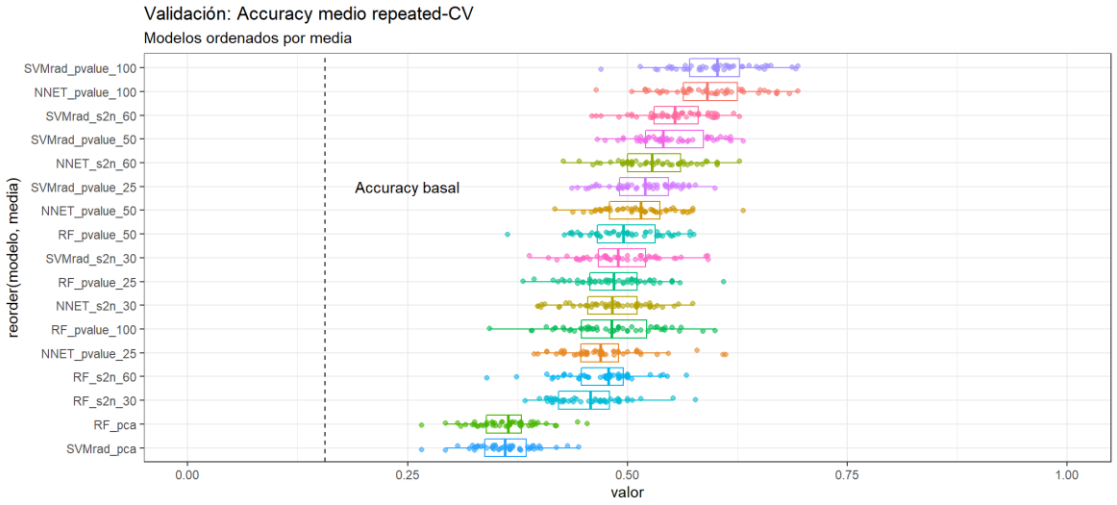
```
modelos <- list(
  SVMrad_pvalue_100 = svmrad_pvalue_100,
  SVMrad_pvalue_50  = svmrad_pvalue_50,
  SVMrad_pvalue_25  = svmrad_pvalue_25,
  SVMrad_s2n_60     = svmrad_s2n_60,
  SVMrad_s2n_30     = svmrad_s2n_30,
  SVMrad_pca        = svmrad_pca,
  RF_pvalue_100     = rf_pvalue_100,
  RF_pvalue_50      = 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")
```



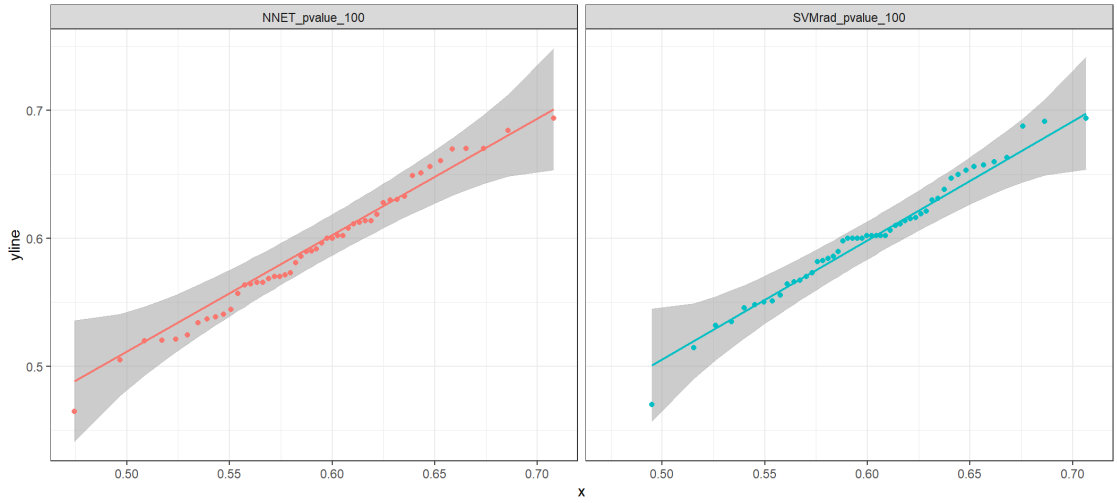
Una de las ventajas de los métodos de validación es que, si se han empleado exactamente los mismos datos en todos los modelos (en este caso es así gracias a las semillas), se pueden aplicar test de hipótesis que permitan determinar si las diferencias observadas entre modelos son significativas o si solo son debidas a variaciones aleatorias. Acorde a los errores de validación obtenidos por bootstrapping, los mejores modelos se consiguen con el algoritmo SVM radial y Neural Network, empleando los genes seleccionados por pvalue 100.

Se compara el mejor de cada uno para determinar si hay evidencias de que uno de ellos sea superior a los demás.

Si se cumple la condición de normalidad, se pueden aplicar un t-test de datos pareados para comparar el accuracy medio de cada modelo.

```
library(qqplotr)
metricas_resamples %>%
  filter(modelo %in% c("SVMrad_pvalue_100", "NNET_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 4 modelos.

```
metricas_ttest <- metricas_resamples %>%
  dplyr::filter(modelo %in% c("SVMrad_pvalue_100", "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
## SVMrad_pvalue_100 0.099
##
## P value adjustment method: none
```

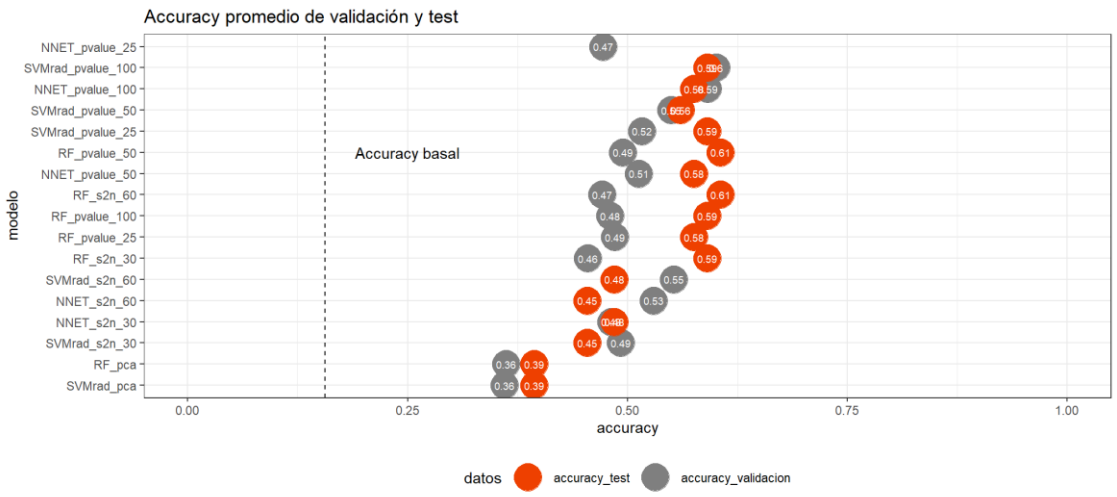
Para un nivel de significancia α=0.05, NO se encuentran evidencias para aceptar que el accuracy promedio de los modelos es distinto.

6.4.2. Error de test

```
predic_svmrad_pvalue_100 <- predict(object = svmrad_pvalue_100, newdata =
datos_test)
```

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

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")
```



Acorde al accuracy de test, el mejor modelo es RF_s2n_30 (0.6363636), seguido por RF_pvalue_50 y RF_s2n_60 (0.6060606).

```
predicciones %>% dplyr::select(RF_s2n_30, valor_real) %>%
dplyr::filter(RF_s2n_30 != valor_real)
```

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

```
confusionMatrix(data = predicciones$RF_s2n_30, reference =
as.factor(datos_test$type))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction CIS HC OND PP RR SP
##        CIS   3  1   1  2  0  0
##        HC    1  6   1  1  2  1
##        OND   0  0   0  0  0  0
##        PP    0  1   0  0  1  0
##        RR    8  5   3  4 21  4
##        SP    0  0   0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.4545
##                  95% CI : (0.3314, 0.5819)
##     No Information Rate : 0.3636
##     P-Value [Acc > NIR] : 0.08102
##
##                   Kappa : 0.2138
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
```

```
##
##                      Class: CIS Class: HC Class: OND Class: PP Class:
RR
## Sensitivity             0.25000   0.46154    0.00000    0.0000
0.8750
## Specificity             0.92593   0.88679    1.00000    0.9661
0.4286
## Pos Pred Value          0.42857   0.50000        NaN    0.0000
0.4667
## Neg Pred Value          0.84746   0.87037    0.92424    0.8906
0.8571
## Prevalence              0.18182   0.19697    0.07576    0.1061
0.3636
## Detection Rate          0.04545   0.09091    0.00000    0.0000
0.3182
## Detection Prevalence    0.10606   0.18182    0.00000    0.0303
0.6818
## Balanced Accuracy       0.58796   0.67417    0.50000    0.4831
0.6518
##                      Class: SP
## Sensitivity            0.00000
## Specificity            1.00000
## Pos Pred Value             NaN
## Neg Pred Value         0.92424
## Prevalence             0.07576
## Detection Rate         0.00000
## Detection Prevalence   0.00000
## Balanced Accuracy      0.50000
```

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

6.4.3. Mejor modelo

Para identificar cuál de todos es el mejor modelo, conviene tener en cuenta los dos tipos de error: el de validación, en este caso bootstrapping, y el de test. En vista de los resultados obtenidos, no puede afirmarse que, para este problema, exista un modelo que supere claramente a todos los demás, de ahí que, pequeñas diferencias, provoquen cambios en el orden de los modelos entre validación y de test. Tanto **SVMrad_pvalue_100, NNET_pvalue_100, RF_s2n_30, RF_s2n_60 y RF_pvalue_50** son buenos candidatos.

7. Model ensemble (stacking)

A modo general, el término model ensembling hace referencia a la combinación de las predicciones de dos o más modelos distintos, con el objetivo de mejorar las predicciones finales. Esta estrategia se basa en la asunción de que, distintos modelos entrenados independientemente, emplean distintos aspectos de los datos para realizar las predicciones, es decir, cada uno es capaz de identificar parte de la "verdad" pero no toda ella. Combinando la perspectiva de cada uno de ellos, se obtiene una descripción más detallada de la verdadera estructura subyacente en los datos.

A modo de analogía, imagínese un grupo de estudiantes que se enfrentan a un examen multidisciplinal. Aunque todos obtengan aproximadamente la misma nota, cada uno de ellos habrá conseguido más puntos con las preguntas que tratan sobre las disciplinas en las que destacan. Si en lugar de hacer el examen de forma individual, lo hacen en grupo, cada uno podrá contribuir en los aspectos que más domina, y el resultado final será probablemente superior a cualquiera de los resultados individuales.

La clave para que el ensembling consiga mejorar los resultados es la diversidad de los modelos. Si todos los modelos combinados son similares entre ellos, no podrán compensarse unos a otros. Por esta razón, se tiene que intentar combinar modelos que sean lo mejor posible a nivel individual y lo más diferentes entre ellos.

Las formas más simples de combinar las predicciones de varios modelos son: emplear la media para problemas de regresión y la moda para problemas de clasificación. También es posible ponderar estas agregaciones dando distinto peso a cada modelo, por ejemplo, en proporción al accuracy que han obtenido de forma individual.

```
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, RF_s2n_30, RF_s2n_60,
RF_pvalue_50) %>%
  mutate(moda = apply(X = dplyr::select(.data =
predicciones,SVMrad_pvalue_100, NNET_pvalue_100, RF_s2n_30, RF_s2n_60,
RF_pvalue_50),
                      MARGIN = 1,
                      FUN = moda,
                      indice_mejor_modelo = 1))

predicciones_ensemble %>% head()

mean(predicciones_ensemble$moda == datos_test$type)

## [1] 0.4848485
```

En este caso, el ensemble de los modelos no consigue una mejora.

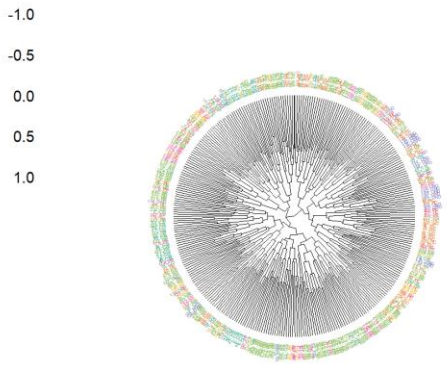
8. Clustering

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

Se procede a agrupar los tipos de EM mediante clustering aglomerativo empleando la selección de genes filtrado_s2n_60.

```
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")
```



El algoritmo de clustering no es capaz de diferenciar los 6 tipos de EM, lo que pone de manifiesto la dificultad de la clasificación.