Código en R: Estudio de la expresión genética del cáncer renal para predecir la supervivencia

Alejandro Floriano Pardal

6 de enero, 2023

Contents

INTRODUCCIÓN

Para mejorar el entendimiento del proceso seguido el documento ha sido dividido en cuatro etapas bien diferenciadas. En cada etapa se explica el objetivo de la misma así como los *outputs* intermedios obtenidos que forman parte del *input* de la siguiente entrada:

- **ETAPA 1**: Recopilación de la información necesaria del TCGA y análisis preliminar de los datos. Posterior a ello se estructuran los dos dataframes (el clínico y el de número de cuentas en bruto de la expresión genética) para identificar aquellos pacientes que poseen información en ambos conjuntos de datos y así seleccionarlos. A su vez cada dataframe quedará estructurado de manera conveniente para su uso posterior: en el caso del df con información clínica, cada paciente será una fila del mismo (y las columnas corresponderán a algunas variables clínicas y demográficas recopiladas) y en el caso del df con las cuentas de la expresión genética cada paciente será una columna y las filas corresponderán a los genes. Por tanto el *output* de esta etapa serán dos dataframes cuya estructura es la necesaria para los pasos siguientes.
- **ETAPA 2**: En este apartado vamos a crear los dos dataframe base que usaremos en los modelos Cox PH a generar y en la evaluación de los mismos. Para ello agruparemos previamente la información clínica y de expresión genética en un único dataframe. El conjunto *train* nos servirá para el entrenamiento de los 3 modelos a realizar y el conjunto *test* para evaluar la bondad de dichos modelos en datos no vistos previamente por el modelo. Durante el proceso normalizaremos los datos de expresión genética y obtendremos los *differential expressed genes* (DEGs) para los pacientes con y sin metástatis en el momento del diagnóstico. El *output* de esta etapa serán 3 dataframes: uno del conjunto *train*, otro del conjunto *test*, y un dataframe con el listado de DEGs obtenidos que son los que usaremos en la creación de los modelos Cox PH.
- **ETAPA 3**: Usando el dataframe del conjunto *train* vamos a generar tres modelos Cox PH diferentes: uno sólo con información de expresión genética a partir de los DEGs calculados, otro sólo con información clínica y un último donde se combina información clínica y de expresión genética. A su vez crearemos un modelo clínico usando la idea que subyace en algunos índices usados clínicamente: asignar valores de score en función de algunas variables clínicas que están presentes y que vendrán a reflejar el grado de riesgo asociado. El *output* de esta etapa son cuatro modelos: tres de ellos generados a partir de modelos Cox PH y uno último creado a partir de valores clínicos. Los modelos Cox PH serán comparados en esta etapa a partir del C-index (índice que se asemeja al área bajo la curva ROC y da una idea de la capacidad de ordenar los resultados a partir del modelo generado)
- **ETAPA 4**: Por último en esta etapa se evaluan los 3 modelos Cox PH sobre 3 conjuntos de datos. Los tres conjuntos de datos sobre los que se analizará la bondad de los modelos son: el conjunto *train*, para asegurar que el modelo funciona sobre los propios datos que generaron el modelo; el conjunto *test*, para evaluar cuánto de bueno son nuestros modelos en datos nuevos que no han sido vistos nunca; y elmismo conjunto *test* pero sólo de los pacientes con estadío I y II, ya que uno de los objetivos es identificar si específicamente para aquellos pacientes que han sido diagnosticados en estadíos tempranos, podemos obtener mejores predicciones de su sobrevida que mejoren la toma de decisiones de tratamiento posterior. A su vez veremos sobre el conjunto *test* los resultados obtenidos aplicando la metodología score con el modelo Cox PH de información genética obtenido.
- **CONCLUSIONES**: En este apartado se realizará un breve resumen de los resultados arrojados por el código R generado y que formarán la base del TFM final que será entregado..
- **ANEXOS**:
  - Anexo 1: Análisis de la influencia de la partición en los DEGs obtenidos.
  - Anexo 2: Ubicación en github.

ETAPA 1. RECOPILACIÓN, ANÁLISIS EXPLORATORIO Y PREPROCESADO DE DATOS

0. Carga de librerías en R:

- Se cargan las siguientes librerías en R:

```
library(TCGAbiolinks)
#library(MultiAssayExperiment)
#library(maftools)
library(dplyr)
#library(ComplexHeatmap)

library(tidyverse)
library(readxl)
library(writexl)
library(lubridate)

#library(SummarizedExperiment)
#library(DT)
library(survival)

library("DESeq2")
library(survivalAnalysis)
```

1. Descarga de los archivos desde el TCGA:

- Este proceso sólo será realizado una vez, ya que descargaremos los datos que necesitamos sobre el proyecto KIRC y los guardaremos en archivos .csv que serán los que usaremos para importar los datos.

1.1 Descarga de los datos clínicos (CLINICAL_DATA):

- Descargamos la información clínica del proyecto KIRC.

```
# descargamos información:
clinical <- GDCquery_clinic("TCGA-KIRC")
head(clinical)

# guardamos a .csv
write.csv2(clinical,"clinical_KIRC.csv")
```

1.2 Descarga de los datos de transcription (RAW_DATA):

- Descargamos la información de expresión genética del proyecto KIRC.

```
# descargamos información del proyecto KIRC
query.TFM <- GDCquery(project = "TCGA-KIRC",
                  data.category = "Transcriptome Profiling",
                  data.type = "Gene Expression Quantification",
                  workflow.type = "STAR - Counts")
GDCdownload(query.TMF)
raw.counts <- GDCprepare(query = query.TFM, summarizedExperiment = FALSE)

# guardamos a .csv
write.csv2(raw.counts,"raw_KIRC.csv")
```

2. Análisis preliminar de los archivos y proceso ETL básico:

2.1 Carga a través de los .csv guardados en el apartado anterior:

- Cargamos los archivos que hemos descargado desde la web.

```
# df_clínico:
df_clinical <- read.csv2("clinical_KIRC.csv")

# df_expresión_genética:
df_raw <- read.csv2("raw_KIRC.csv")
```

2.2 Análisis exploratorio de los datos y algunas modificaciones de los dataframes:

2.2.1 Dataframe clínico (df_clinical):

- Datos básicos:

```
nrow(df_clinical)
```

```
## [1] 537
```

```
ncol(df_clinical)
```

```
## [1] 72
```

- Tenemos 537 observaciones: cada observación corresponde con la información clínica de un paciente con cáncer de riñón.
- Tenemos 71 variables relacionadas con cada observación.
- El identificador único del paciente es la columa **submitter_id**
- Algunos pacientes se encuentran fallecidos y otros vivos.
- Se realizarán algunas modificaciones en el dataframe:
  - Crearemos una columna (*status*) que especifique si el paciente está vivo (valor 0), o fallecido (valor 1)
  - Crearemos una columna donde esté el tiempo (t) desde el diagnóstico hasta el último evento ocurrido (bien el fallecimiento, bien la última revisión).
  - Vamos a crear una nueva columna que indique si tiene o no metástasis en base a la información que contiene actualmente el dataframe. Para aquellos pacientes donde no tengamos información de si es metástasis o no (es decir, donde no quede identificado si es M1 o M0 bien porque aparezca MX bien porque esté en blanco) seguiremos el siguiente criterio:
    * Si es Stage I,II y III asignaremos M0.
    * Si es Stage IV asignaremos M1.
    * Existe un caso donde tenemos MX y el Stage es desconocido. Eliminaremos ese registro al no poseer información.
- Veamos la tabla de frecuencias donde se identifica el M del paciente:

```
tab_ajcc_pathologic_m <- table(df_clinical$ajcc_pathologic_m,df_clinical$ajcc_pathologic_stage,useNA="i
tab_ajcc_pathologic_m
```

```
##
##        Stage I Stage II Stage III Stage IV <NA>
##   M0       245       54       123        3    1
##   M1         0        0         0       78    1
##   MX        22        3         2        2    1
##   <NA>       2        0         0        0    0
```

- Creamos las tres columnas mencionadas:

```
# indicación de metástatsis o no en función de la variuable ajcc_pathologic_m:
df_clinical2 <- df_clinical%>%
  mutate(metastasis_a_fecha_diagnostico=case_when(
    ajcc_pathologic_m == "M1" ~ "con_metastasis",
    ajcc_pathologic_m == "M0" ~ "sin_metastasis",
    (ajcc_pathologic_m == "MX" & ajcc_pathologic_stage =="Stage I") ~ "sin_metastasis",
        (ajcc_pathologic_m == "MX" & ajcc_pathologic_stage =="Stage II") ~ "sin_metastasis",
        (ajcc_pathologic_m == "MX" & ajcc_pathologic_stage =="Stage III") ~ "sin_metastasis",
            (ajcc_pathologic_m == "MX" & ajcc_pathologic_stage =="Stage IV") ~ "con_metastasis",
    is.na(ajcc_pathologic_m) ~ "sin_metastasis",
    is.na(ajcc_pathologic_stage) ~ "ELIMINAR"
  ))

# tiempo desde el diagnóstico hasta el último evento ocurrido:
df_clinical2 <- df_clinical2 %>%
  mutate(t = case_when(
    vital_status=="Alive" ~ days_to_last_follow_up,
    vital_status=="Dead" ~ days_to_death
  ))

# estatus del paciente:
df_clinical2 <- df_clinical2 %>%
  mutate(status=case_when(vital_status=="Alive" ~ 0,
                   vital_status=="Dead" ~ 1))
```

- Tabla de frecuencia para la nueva columna creada:

```
tab_ajcc_pathologic_m_new <- table(df_clinical2$metastasis_a_fecha_diagnostico,useNA="ifany" )
tab_ajcc_pathologic_m_new
```

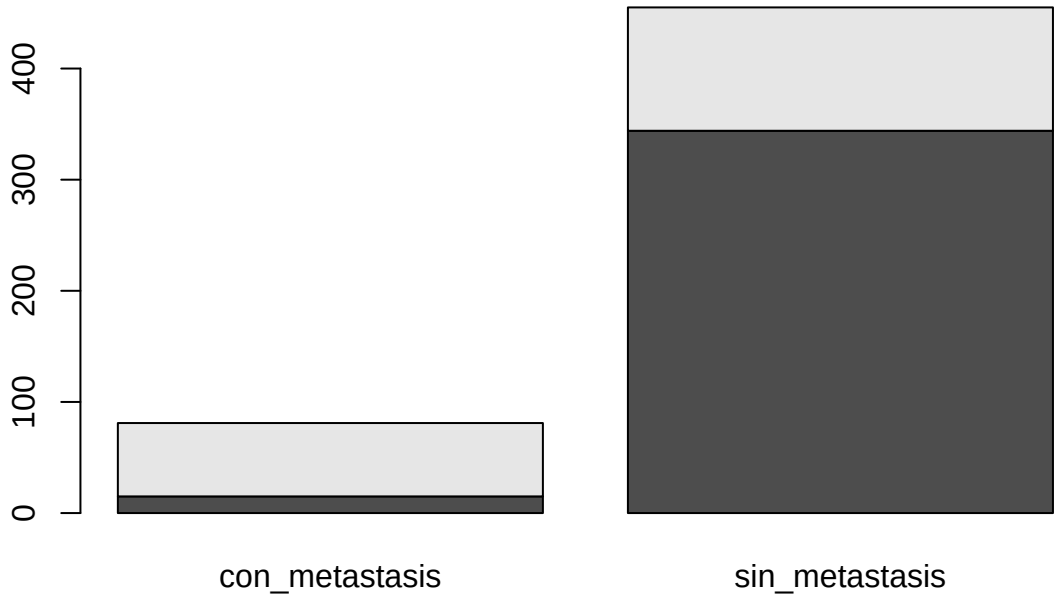
```
##
## con_metastasis       ELIMINAR sin_metastasis
##             81              1            455
```

- Eliminamos el registro del que no sabemos situación sobre su metástasis:

```
df_clinical2 <- df_clinical2 %>%
  filter(metastasis_a_fecha_diagnostico != "ELIMINAR")
```

- Veamos varios gráficos muy básicos sobre cómo están repartidos los pacientes en relación a las metástasis y el status:

```
tab_clinical <- table(df_clinical2$vital_status, df_clinical2$metastasis_a_fecha_diagnostico)
barplot(tab_clinical)
```



- Muchas de las variables clínicas que aparecen no están completas o no son relevantes para nuestro estudio, creamos un dataframe reducido del dataframe clínico con aquellas variables que resultan de interés para el proceso:

```
# seleccionamos algunas variables de interés que nos servirán posteriormente en el cox análisis:
df_clinical3 <- df_clinical2 %>%
  select(3,5,20,23,24,39,42,73,41,74,75)

# el rownames de este dataframe será el código del paciente:
rownames(df_clinical3) <- df_clinical3$submitter_id

# ordenamos las FILAS en orden alfabético en función del código del paciente
df_clinical3 <- df_clinical3[ order(row.names(df_clinical3)), ]
```

- Borramos df intermedios:

```
rm(df_clinical2)
```

2.2.2 Dataframe expresión genética (df_raw):

- Tenemos 60664 observaciones: cada observación corresponde a las cuentas en bruto de la expresión genética de un gen determinado.
- Tenemos 3681 variables relacionadas con cada gen. Aparte de las 3 primeras variables (que describen el nombre del gen y el tipo), el resto de columnas son las cuentas para cada gen en función de la muestra de cada paciente.
- Un paciente puede tener varias muestras (lo que significa que posee varias columnas de datos).
- Queremos los datos en bruto, que corresponden a las columnas nombradas como **unstranted_**+ el código asociado a la muestra:
- El identificador único del paciente es la columa *submitter_id* (hasta la columna 617 del dataframe)
- Extraemos a un nuevo dataframe esas columnas **unstranted_** (son 613 en total):

```
df_raw2 <- df_raw %>%
  dplyr::select(5:617)
```

- Queremos identificar a qué paciente pertenece cada una de las muestras (para asociarlos al *id_submitter*), por lo que realizamos las modificaciones de los nombres de las columnas para que queden en el mismo formato que tienen los códigos en el dataframe clínico:

```
# substraemos la parte inicial:
colnames(df_raw2) <- sub("unstranded_", "", colnames(df_raw2))

# nos quedamos con la parte de la codificación que identifica a un paciente:
colnames(df_raw2) <- substr(colnames(df_raw2),1,12)

# sustituimos el . por el -. Así tenemos el mismo formato:
colnames(df_raw2) <- gsub("\\.", "-", colnames(df_raw2))

# El rownames de este dataframe será el nombre de los genes del dataframe cargado:
rownames(df_raw2) <- df_raw$gene_id

# ordenamos las COLUMNAS del dataframe nuevo en orden alfabético:
df_raw3 <- df_raw2[,order(colnames(df_raw2))]
```

NOTA IMPORTANTE:Como ocurre que algunos pacientes tienen varias muestras y R no permite que dos columnas se llamen igual, el software R le añade un .# cuando esto ocurre. Que compartan una parte del código de paciente común es lo que nos va a permitir el siguiente apartado, ya que lo que haremos en identificar estos casos y calcular para ellos el valor promedio de las lecturas de dichas columnas.

- Borramos df intermedios:

```
rm(df_raw2)
```

2.3 Identificación de los datos comunes en los dos dataframes:

- En el conjunto de datos descargado ocurre que tenemos algunos pacientes con datos clínicos pero que no poseen muestras de expresión genética y también tenemos algún código de muestra del que no tenemos ningún dato clínico. Por ello debemos identificar aquellos códigos que son comunes en ambos dataframes (el clínico y el de cuentas):

```
id_comunes <- intersect(colnames(df_raw3),rownames(df_clinical3))
length(id_comunes)
```

```
## [1] 531
```

- Tenemos por tanto 531 códigos de los que tenemos tanto información clínica como muestras de datos. Este conjunto de id_comunes nos servirá más adelante para filtrar la información deseada.

3. Preparación de los 2 dataframes para estructura necesaria en DESEq2:

- Para que podamos realizar el análisis necesitamos modificar las estructuras de ambos dataframes.
- La primera modificación es en el df_raw, ya que para cada paciente sólo puede haber una única muestra de cuentas de expresión genética. En aquellos en los que hay más de una, calcularemos el promedio de cuentas.
- Vamos a generar una lista de dataframes. Cada conjunto de la lista contendrá aquellas columnas de muestras de un mismo paciente:

```
nombres <- as.vector(rownames(df_clinical3))

dfList_counts <-list()

for (i in 1:length(nombres)){
  df <- select(df_raw3,contains(nombres[i]))
  dfList_counts[[i]] <- df
}
```

- Ahora podemos calcular el valor medio de cada elemento de la lista (es el valor promedio fila a fila). Cada nuevo dataframe de la lista tendrá ya un único elemento, que será el valor promedio de las muestras tomadas

```
dfList_counts_Mean <- lapply(dfList_counts,as.data.frame(rowMeans))

# le asignamos a cada dataframe el código que le corresponde:
names(dfList_counts_Mean) <- nombres

# le asignamos a la columna generada el código del que procede:
for (i in 1:length(nombres)){
  colnames(dfList_counts_Mean[[i]])[1] = nombres[i]
}
```

- Por último debemos agrupar la información en un único dataframe a partir de todos estos dataframes generados que contienen la información promedio de las lecturas asociadas a cada paciente. El df que contendrá una columna por paciente es *matriz_gen*:

```
# inicializamos la matriz con el primer dataframe:
matriz_gen <- dfList_counts_Mean[[1]]

# vamos haciendo merge para cada dataframe hasta llegar al último:
for (i in 1:535){
  matriz_gen <- merge(matriz_gen,dfList_counts_Mean[[i+1]], by = 'row.names')
  rownames(matriz_gen) <- matriz_gen[,1]
  matriz_gen <- matriz_gen %>% select(-1)
}
```

- Ahora nos quedamos de estos dos dataframes con aquellos identificadores que son comunes a ambos (recordemos que los identificadores comunes están en la lista **id_comunes**)

```
# Identificamos las filas comunes del dataframe clínico:
df_clinical_final <- df_clinical3 %>%
  filter(rownames(df_clinical3) %in% id_comunes)

# Identificamos las columnas comunes del la matriz de cuentas:
names.use <- names(df_raw3)[(names(df_raw3) %in% id_comunes)]
df_raw_final <- matriz_gen[, names.use]
```

- Borramos df intermedios:

```
rm(df)
rm(matriz_gen)
rm(df_clinical3)
rm(df_raw3)
```

- El resultado final de este apartado es que tenemos dos dataframes (el clínico con 531 filas y el de cuentas con 531 columnas con el mismo identificador) con la estructura necesaria para usarlos en DESEq2 e identificar así los DEGs entre distintos estados, que es lo que veremos en el apartado siguiente.

4. Inclusión de score en el datafrme clínico (según metodología score tipo GRANT/Leibovich):

- Vamos a crear un dataframe clínico calculando el riesgo siguiendo la metodología *score*. Por tanto, vamos a añadir en este apartado variables categóricas que asignan un valor 0 ó 1 en función de las variables clínicas que poseemos. A su vez, dejaremos calculado el score final numérico y la categorización de riesgo clínico asociado. Las variables que formarán parte de nuestro score son:
  - ajcc_pathologic_t
  - ajcc_pathologic_n
  - age_at_index
  - ajcc_pathologic_stage
- El *score_clinical* se obtiene a partir de la suma de los valores previos. Si el resultado del *score* es mayor que 2, se considerará alto riesgo, y si es menor, se considerará bajo riesgo según esta metodología.

```
df_clinical_final_score <- df_clinical_final
df_clinical_final_score <- df_clinical_final_score %>%
  mutate(ajcc_pathologic_t_cat = case_when(
    (ajcc_pathologic_t=="T3b" | ajcc_pathologic_t=="T3c"| ajcc_pathologic_t=="T4") ~ 1,
    TRUE ~ 0
  ))

df_clinical_final_score <- df_clinical_final_score %>%
  mutate(ajcc_pathologic_n_cat = case_when(
    ajcc_pathologic_n=="N1" ~ 1,
    TRUE ~ 0
  ))

df_clinical_final_score <- df_clinical_final_score %>%
  mutate(age_at_index_cat = case_when(
    age_at_index>60 ~ 1,
    TRUE ~ 0
  ))

df_clinical_final_score <- df_clinical_final_score %>%
  mutate(ajcc_pathologic_stage_cat = case_when(
    (ajcc_pathologic_stage=="Stage III" | ajcc_pathologic_stage=="Stage IV") ~ 1,
    TRUE ~ 0
  ))

df_clinical_final_score$score_clinical <- df_clinical_final_score$ajcc_pathologic_t_cat+
                                          df_clinical_final_score$ajcc_pathologic_n_cat+
                                          df_clinical_final_score$age_at_index_cat+
                                          df_clinical_final_score$ajcc_pathologic_stage_cat

df_clinical_final_score <- df_clinical_final_score %>%
  mutate(risk_clinical_score = case_when(
    score_clinical >=2 ~ "riego clinico alto",
    TRUE ~ "riesgo clinico bajo"
  ))
```

Resumen output ETAPA1:

En esta etapa tenemos tres dataframes como *output*:

- **df_clinical_final**: Dataframe con la información clínica relevante para 531 pacientes. Todos ellos poseen una muestra genética asociada. A su vez tienen asignada un tiempo entre el diagnóstico y el último evento, un estatus numérico sobre si están vivos o no y una marca sobre si tenían metástasis en el momento del diagnóstico.
- **df_clinical_final_score**: Dataframe idéntico al anterior al que se le han unido algunas variables categorizadas según metología GRANT/Leibovich para crear el modelo clínico que usaremos.
- **df_raw_final**: Dataframe con la información de las cuentas de expresión genética sin normalizar para 60664 genes en los 531 pacientes de los que se posee información clínica. En aquellos pacientes que tenían varias muestras se calculó el valor promedio de las cuentas y se asignó este valor.

ETAPA 2. PROCESADO DE DATOS: GRUPOS TRAIN/TEST, NORMALIZACIÓN VALORES Y CÁLCULO DE DEGS

4. Creación de subgrupos train y test:

4.1. Separación del df clínico en dos: clinical_train y clinical_test

- En el grupo *train* estarán todos aquellos pacientes con metástasis y el 80% de los pacientes sin metástasis. El grupo *test* estará formado por el 20% de los pacientes sin metástasis.
- Separamos los pacientes con y sin metástasis:

```
df_clinical_metas <- df_clinical_final_score %>%
  filter(metastasis_a_fecha_diagnostico == "con_metastasis")

df_clinical_no_metas <- df_clinical_final_score %>%
  filter(metastasis_a_fecha_diagnostico == "sin_metastasis")

nrow(df_clinical_metas)
```

```
## [1] 81
```

```
nrow(df_clinical_no_metas)
```

```
## [1] 450
```

- Separamos el grupo que **no tiene metástasis** en dos grupos. Uno con el 80% para obtener los DEGs (en combinación con los pacientes que sí tienen metástatasis) y el 20% para evaluación posterior:

```
#seleccionamos una semilla para reproducibilidad del proceso:
set.seed(42)

# hacemos la separación en los dos grupos señalados:
sample <- sample(c(TRUE, FALSE), nrow(df_clinical_no_metas), replace=TRUE, prob=c(0.8,0.2))

train  <- df_clinical_no_metas[sample, ]
test   <- df_clinical_no_metas[!sample, ]
```

- Vemos cómo está repartido en train en relación al status de vida:

```
tab_TRAIN <- table(train$vital_status)
tab_TRAIN
```

```
##
## Alive  Dead
##   274    87
```

- Vemos cómo está repartido en test en relación al status de vida:

```
tab_TEST <- table(test$vital_status)
tab_TEST
```

```
##
## Alive  Dead
##    67    22
```

Vemos que en ambos grupos alrededor del 70% de los pacientes se encuentran con vida y el 30% fallecidos.

- Creamos el grupo clinical_train (combinando los pacientes con y sin metástasis) y clinical_test (sólo renombrado) a partir de los anterior:

```
# unimos el train creado con aquellos pacientes con metástasis para consolidad el conjunto final
# de entrenamiento:
clinical_train <- train %>%
  rbind(df_clinical_metas)
clinical_train <- clinical_train[ order(row.names(clinical_train)), ]

# renombramos el grupo test
clinical_test <- test
```

- Borramos algunos df intermedios:

```
rm(dfList_counts)
rm(dfList_counts_Mean)
```

Por tanto ya tenemos separado el dataframe información clínica en dos grupos, el de entrenamiento y el de testeo.

4.2. Separación del df_raw_final en dos: raw_train y raw_test

- En base a los códigos de los pacientes que están en cada uno de los dos grupos anteriores, procedemos a la separación del dataframe de la cuentas de la expresión genética en los mismos dos subgrupos:

```
# Obtenemos el df de raw_train:
id_comunes_train <- intersect(colnames(df_raw_final),rownames(clinical_train))
names.use_train <- names(df_raw_final)[(names(df_raw_final) %in% id_comunes_train)]
raw_train <- df_raw_final[, names.use_train]


# Obtenemos el df de raw_test:
id_comunes_test <- intersect(colnames(df_raw_final),rownames(clinical_test))
names.use_test <- names(df_raw_final)[(names(df_raw_final) %in% id_comunes_test)]
raw_test <- df_raw_final[, names.use_test]
```

5. Normalización de los valores raw del conjunto TEST:

5.1. Creación del dds (sólo para uso de valores normalizados):

- Tenemos que normalizar los valores test que usaremos posteriormente. Para ello realizarmos un cálculo con DESeq que sólo usaremos para extraer los valores normalizados del grupo test en el apartado 5.2:

```
clinical_completo_new <- df_clinical_final_score
raw_completo_new <- df_raw_final

# en el raw data:
raw_completo_new <- mutate_all(raw_completo_new, function(x) as.integer(x))

# en el clinical data:
clinical_completo_new$metastasis_a_fecha_diagnostico <- as.factor(clinical_completo_new$metastasis_a_fe

dds_total <- DESeqDataSetFromMatrix(countData = raw_completo_new,
                                    colData = clinical_completo_new,
                                    design = ~ metastasis_a_fecha_diagnostico)

dds_total <- DESeq(dds_total)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

## -- replacing outliers and refitting for 8907 genes
## -- DESeq argument 'minReplicatesForReplace' = 7
## -- original counts are preserved in counts(dds)

## estimating dispersions

## fitting model and testing
```

5.2. Normalización de valores TEST:

- A partir de los valores normalizados en el paso anterior, nos quedamos con aquellos que forman parte del conjunto *test*.

```
# obtenemos los valores normalizados del conjunto completo
df_normalizado_final <- data.frame(counts(dds_total, normalized=TRUE))

colnames(df_normalizado_final) <- colnames(raw_completo_new)

# nos quedamos con los valores normalizados del subgrupo test:
norm_test <- df_normalizado_final[,names.use_test]
```

- Borramos df que no nos interesan:

```
rm(clinical_completo_new)
rm(raw_completo_new)
```

6. Normalización de valores TRAIN:

6.1 Creación del dds:

- El cálculo de los DEGs lo realizarmos sólo con los datos del conjunto *clinical_train*:

```
# renombramos datos del conjunto train:
df_clinical_new <- clinical_train
df_matriz_gen_new <- raw_train

# Modificaciones previas para que funcione la función:
# en el raw data:
df_matriz_gen_new <- mutate_all(df_matriz_gen_new, function(x) as.integer(x))

# en el clinical data:
df_clinical_new$metastasis_a_fecha_diagnostico <- as.factor(df_clinical_new$metastasis_a_fecha_diagnosti
```

- Creamos el dds con los datos de train:

```
dds <- DESeqDataSetFromMatrix(countData = df_matriz_gen_new,
                              colData = df_clinical_new,
                              design = ~ metastasis_a_fecha_diagnostico)
```

- Aplicamos la función DESeq y obtenemos sus resultados. A partir de estos resultados normalizaremos los datos del conjunto *clinical_train* e identicaremos los DEGs que usaremos en los apartados subsiguientes:

```
dds2 <- DESeq(dds)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

## -- replacing outliers and refitting for 6033 genes
## -- DESeq argument 'minReplicatesForReplace' = 7
## -- original counts are preserved in counts(dds)

## estimating dispersions

## fitting model and testing
```

```
res <- results(dds2)
```

6.2. Normalización de TRAIN:

- Creamos el dataframe de valores normalizados a partir del dds2 (con el que hemos hecho el cálculo de los DEGs):

```
# valores normalizados para el conjunto de datos clinical_train:
norm_train <- data.frame(counts(dds2, normalized=TRUE))
colnames(norm_train) <- colnames(raw_train)
```

7. Selección de DEGs (en función de parámetros de log2FoldGhange y p.adjust):

- En este paso debemos decidir qué vamos a considerar como DEG, que vendrá determinado por la combinación de dos parámetros: el valor absoluto del log2FoldChange y el valor del padj.
- Valores seleccionados para ambos parámetros (estos corresponden a los usados por Yang et.al y parece un buen compromiso en la cantidad de DEGs identificados inicialmente):

```
hp_abs_log2FoldChange <- 1.000
hp_res.padj <- 0.001
```

- Filtramos los DEGS en función de dichos valores:

```
# Creamos df con nombre de genes y valores de los resultados:
analisis_m <- data.frame(df_raw$gene_id,df_raw$gene_name,df_raw$gene_type, res$baseMean,res$log2FoldChan

# creamos variable de valor absoluto de log2FoldChange:
analisis_m$abs_log2FoldChange <- abs(analisis_m$res.log2FoldChange)

# Filtramos los DEGs que queremos:
DEGs_metas <- analisis_m %>%
  filter(abs_log2FoldChange > hp_abs_log2FoldChange  & res.padj < hp_res.padj)

DEGs_metas <- DEGs_metas %>%
  arrange(df_raw.gene_name)

# Número de DEGs seleccionados:
nrow(DEGs_metas)
```

```
## [1] 350
```

Con este criterio tenemos 350 DEGs identificados. El listado de los mismos se encuentra en el dataframe **DEGs_metas**.

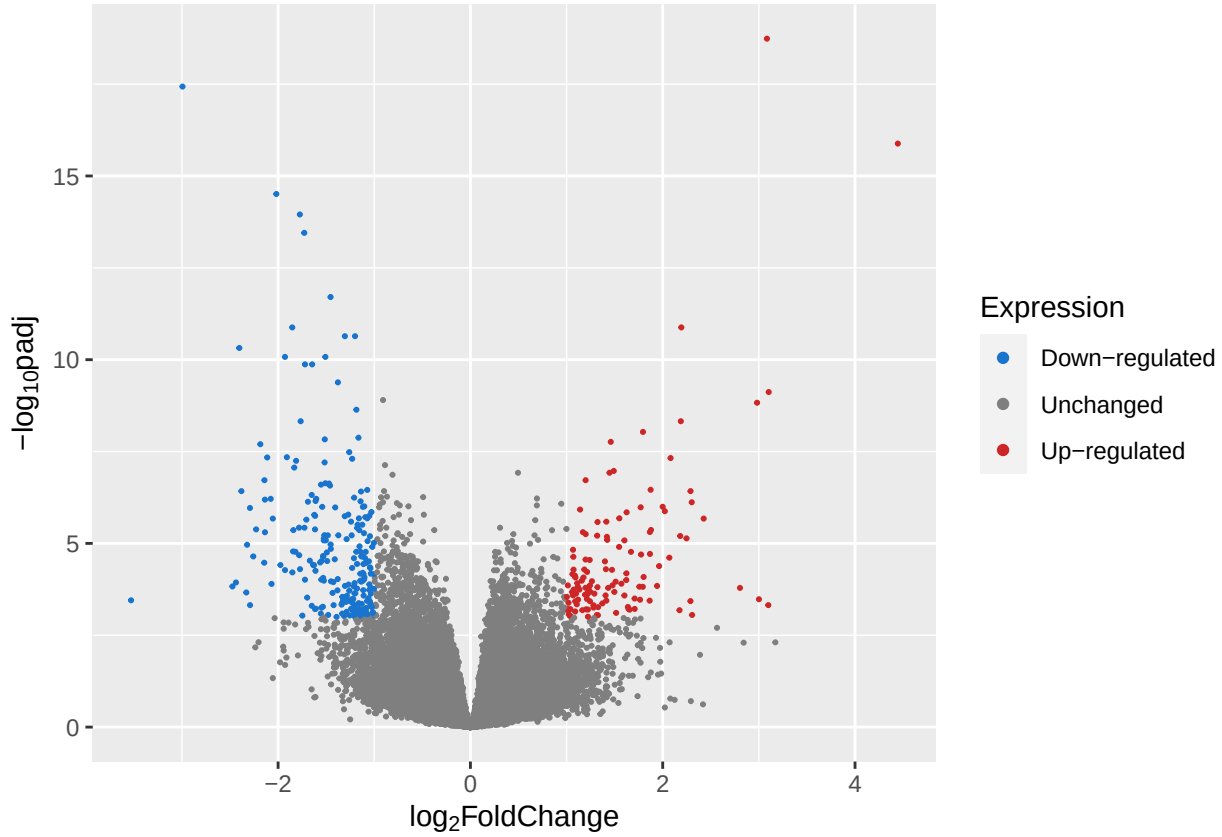
- Borramos df intermedios:

```
rm(df_clinical_new)
rm(df_matriz_gen_new)
```

- Podemos visualizar gráficamente los DEGs seleccionados a partir del gráfico Volcano:

```
analisis_m_volcano <- analisis_m %>%
  mutate(
    Expression = case_when(res.log2FoldChange > 1 & res.padj <= 0.001 ~ "Up-regulated",
                           res.log2FoldChange <= -1 & res.padj <= 0.001 ~ "Down-regulated",
                           TRUE ~ "Unchanged")
    )
p1 <- ggplot(analisis_m_volcano, aes(res.log2FoldChange, -log(res.padj,10))) + # -log10 conversion
  geom_point(aes(color = Expression), size = 2/5) +
  xlab(expression("log"[2]*"FoldChange")) +
  ylab(expression("-log"[10]*"padj")) +
  scale_color_manual(values = c("dodgerblue3", "gray50", "firebrick3")) +
  guides(colour = guide_legend(override.aes = list(size=1.5)))
p1
```

```
## Warning: Removed 19562 rows containing missing values (`geom_point()`).
```



Este gráfico nos permite identificar qué genes son los que poseen expresión genética distinta entre los pacientes que tienen metástasis y aquellos que no la tienen.

8. Creación de dataframes bases COMBINADOS (datos clínicos + expresión genética):

Por último vamos a crear el dataframe con la información combinada tanto clínica como de expresión genética del conjunto train y del conjunto test que serán la base para la creación y evaluación de los modelos COX PH de la siguiente etapa.

8.1. Creación del dataframe COMBINADO para el conjunto TRAIN (para entrenamiento):

_ Transponemos el dataframe de cuentas normalizadas y lo unimos al clínico. El valor de cuentas de los genes serán ahora una variable más (columna) de mi dataframe:

```
# transposición de valores:
norm_train_cox_T <-as.data.frame(t(norm_train))

# combinación de ambos df:
df_cox_model_train <- merge(clinical_train, norm_train_cox_T, by=0, all=TRUE)
rownames(df_cox_model_train) <- df_cox_model_train$Row.names

names.use_train <- rownames(df_cox_model_train)
```

8.2. Creación del dataframe COMBINADO para el conjunto TEST (para testear completo):

_ Transponemos el dataframe de cuentas y lo unimos al clínico. El valor de cuentas de los genes serán ahora una variable más (columna) de mi dataframe:

```
# transposición de valores:
norm_test_cox_T <-as.data.frame(t(norm_test))

# combinación de ambos df:
df_cox_model_test <- merge(clinical_test, norm_test_cox_T, by=0, all=TRUE)
rownames(df_cox_model_test) <- df_cox_model_test$Row.names

names.use_test <- rownames(df_cox_model_test)
```

8.3. Creación del dataframe de evaluación en TEST (sólo STAGE I y II):

- Creamos el dataframe reducido que usaremos en este apartado (conjunto TEST (sólo stage I y II)):

```
df_cox_model_test_stage <- df_cox_model_test %>%
  filter(df_cox_model_test$ajcc_pathologic_stage=="Stage I" |
           df_cox_model_test$ajcc_pathologic_stage=="Stage II" )

names.use_test_stage <- rownames(df_cox_model_test_stage)
```

8.4. Creación del dataframe unificado:

- Dataframe único con toda la información tanto clínica como de expresión genética normalizada para las 531 muestras que forman parte del estudio:

```r
df_expresion_clinico_total <- rbind(df_cox_model_train,df_cox_model_test)
names.use_expresion_clinico_total <- rownames(df_expresion_clinico_total)
```

Resumen output ETAPA2:

En esta etapa tenemos cinco dataframes como *output*:

- **df_cox_model_train**: Dataframe combinado completo con la información clínica relevante y las cuentas de genes normalizados. Este es el dataframe base que será usado para entrenar cualquiera de los modelos COX posteriores, dependiendo del filtrado de información que se realice (por ejemplo habrá que seleccionar de este dataframe los DEGs que consideremos oportunos). Tenemos 442 filas que corresponden a códigos de pacientes.
- **df_cox_model_test**: Dataframe combinado completo con la información clínica relevante y las cuentas de genes normalizados. Este es el dataframe base que será usado para testear el modelo COX que se genere. No será necesario filtrar nada, ya que cuando se haga la predicción correspondiente la propia función identifica las columnas que deben ser usada. Contiene 89 filas, que corresponden a códigos de pacientes.
- **df_expresion_clinico_total**: Dataframe combinado completo con la información clínica relevante y las cuentas de genes normalizados para los 531 pacientes, es decir, la combinación de los dos dataframes previos
- **df_cox_model_test_stage**: Dataframe test pero sólo de aquellos pacientes que son stage I y II
- **DEGs_metas**: Dataframe con el listado de los genes seleccionados según el procedimiento empleado y los datos de corte tanto de p.adjust como de log2FoldChange. En total son 350, pero no todos los DEGs serán usados para la creación del modelo Cox PH. Lo que tenemos es el conjunto inicial del que eligiremos en el apartado siguiente los DEGs que formarán parte del modelo final.

A su vez tenemos cuatro listas de muestras:

- **names.use_expresion_clinico_total**: listado de todos los códigos identificativos de pacientes que tenían tanto datos clínicos como datos de expresión genética.
- **names.use_train**: listado de los códigos identificativos de los pacientes usados para entrenar
- **names.use_test**: listado de los códigos identificativos de los pacientes usados para testear (completo)
- **names.use_test_stage**: listado de los códigos identificativos de los pacientes de Stage I y II usados para testear.

ETAPA 3. CREACIÓN DE MODELOS COX PH: EXPRESIÓN GENÉTICA, CLÍNICO Y COMBINADO

En este apartado vamos a generar los tres modelos Cox PH que vamos a evaluar en la etapa última. Pasamos a describir la metodología de creación de cada uno de ellos así como las características de cada modelo:

10. Creación modelo COX PH EXPRESIÓN GENÉTICA: res.cox_DEGs1_final

10.1 Selección de DEGs para creación del modelo:

- Del conjunto de genes que han sido identificados como DEGs en el apartado anterior vamos a seleccionar los 50 con mayor valor absoluto del *log2FoldChange*.
- Parámetro sobre el número de DEGs que vamos a utilizar:

```
n_selec_degs <- 50
```

- Nos quedamos con los 50 DEGs con mayor *log2FoldChange*:

```
# ordenamos de mayor a menor por valor de abs_log2FoldChange:
DEGs_metas_cox <- DEGs_metas %>%
  arrange(desc(abs_log2FoldChange))

# selecciomaos y guardamos en nuevo df:
DEGs_metas_cox_seleccionados <- DEGs_metas_cox[1:n_selec_degs,]
```

10.2 Creación del modelo:

- La creación del modelo genético final lo vamos a realizar en dos pasos: primero generamos un modelo con todos los DEGs seleccionados del apartado 10.1, nos quedamos con los estadísticamente significativos y volvemos a crear un modelo con un número menor de DEGs, que será el modelo final asociado a la expresión genética.
- Hacemos un modelo inicial para esos genes seleccionados y a su vez elejimos los genes significativos para un segundo modelo cox:

```
# Elegimos los DEGs y las variables t y status:
name.use_seleccionados <- c(DEGs_metas_cox_seleccionados$df_raw.gene_id,"t","status")

df_cox_model_train_s  <- df_cox_model_train[, name.use_seleccionados]

matriz_cox <- df_cox_model_train_s
```

- Realizamos el modelo temporal primero:

```
res.cox_DEGs1 <- coxph(Surv(t, status) ~ ., data =  matriz_cox,iter.max=500 )
```

- Parámetro del valor de p para filtrar los DEGs que formarán parte del modelo final:

```
p_significativo_cox <- 0.05
```

- Nos quedamos con aquellos DEGs que han salido significativos en el modelo previo:

```
# convertimos a df los resultados del modelo Cox:
result_temporal <- cox_as_data_frame(res.cox_DEGs1)

# filtramos aquellos estadísticamente significativos:
result_temporal2 <- result_temporal %>%
  filter(p < p_significativo_cox)
nrow(result_temporal2)
```

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

```
# como los nombres de los genes quedan modificados en el proceso, se arregla aquí:
result_temporal2$gen_name <- paste("E",result_temporal2$factor.value)
result_temporal2$gen_name <-  str_replace_all(string=result_temporal2$gen_name, pattern=" ", repl="")

# Elegimos los DEGs para crear el nuevo modelo
names.use2 <- c(result_temporal2$gen_name,"t","status")
matriz_cox_v2 <- matriz_cox[, names.use2]
```

- Usamos el conjunto de DEGs seleccionados tras el primer proceso para crear el modelo Cox PH de expresión genética final:

```
# creación del modelo:
res.cox_DEGs1_final <- coxph(Surv(t, status) ~ ., data =  matriz_cox_v2,iter.max=500 )

# revisión de resultados:
res.cox_DEGs1_final
```

```
## Call:
## coxph(formula = Surv(t, status) ~ ., data = matriz_cox_v2, iter.max = 500)
##
##                          coef  exp(coef)   se(coef)      z        p
## ENSG00000234844.1  -0.0211040  0.9791172  0.0087970 -2.399  0.01644
## ENSG00000235207.1   0.0128658  1.0129489  0.0042926  2.997  0.00272
## ENSG00000189001.11  0.0008728  1.0008732  0.0002079  4.198 2.70e-05
## ENSG00000248227.1   0.0369861  1.0376786  0.0075568  4.894 9.86e-07
## ENSG00000286016.1   0.0121941  1.0122687  0.0027439  4.444 8.83e-06
## ENSG00000155269.12  0.0301647  1.0306242  0.0067156  4.492 7.06e-06
## ENSG00000170044.8  -0.0005692  0.9994309  0.0001433 -3.972 7.13e-05
##
## Likelihood ratio test=64.66  on 7 df, p=1.763e-11
## n= 442, number of events= 153
```

- Construimos un dataframe con los resultados de los genes y los valores del modelo creado:

```r
# creación de df con los resultados del modelo cox ph:
result_final_degs <- cox_as_data_frame(res.cox_DEGs1_final)

# proceso para rehacer el nombre de los genes que son DEGs
result_final_degs$gen_name <- paste("E",result_final_degs$factor.value)
result_final_degs$gen_name <-  str_replace_all(string=result_final_degs$gen_name, pattern=" ", repl="")
result_final_degs <- result_final_degs %>%
  select(-(1:3))

result_final_degs
```

```
##          HR  Lower_CI  Upper_CI    Inv_HR Inv_Lower_CI Inv_Upper_CI
## 1 0.9791172 0.9623801 0.9961453 1.0213282    1.0038696    1.0390905
## 2 1.0129489 1.0044623 1.0215072 0.9872166    0.9789456    0.9955575
## 3 1.0008732 1.0004654 1.0012812 0.9991275    0.9987205    0.9995348
## 4 1.0376786 1.0224228 1.0531620 0.9636896    0.9495216    0.9780690
## 5 1.0122687 1.0068394 1.0177274 0.9878800    0.9825814    0.9932071
## 6 1.0306242 1.0171478 1.0442792 0.9702857    0.9575983    0.9831413
## 7 0.9994309 0.9991502 0.9997117 1.0005694    1.0002884    1.0008505
##              p           gen_name
## 1 1.644051e-02  ENSG00000234844.1
## 2 2.724873e-03  ENSG00000235207.1
## 3 2.696133e-05 ENSG00000189001.11
## 4 9.858847e-07  ENSG00000248227.1
## 5 8.828176e-06  ENSG00000286016.1
## 6 7.063565e-06 ENSG00000155269.12
## 7 7.128501e-05  ENSG00000170044.8
```

```r
write.csv2(result_final_degs,"OUTPUT_DEGS_COX/result_final_degs_modelo.csv")
```

- Como podemos observar los 7 DEGs aparecen como estadísticamente significativos.

10.3 Cálculo del índice de concordancia:

En los modelos Cox tenemos un índice que evalua la concordancia en el ordenamiento de los eventos sucedidos que da una idea de lo bueno/malo que es un modelo creado y que viene a asemejarse al área bajo la curva ROC en modelos logísticos.

Para más información, se dejan las referencias:

Ref1: https://www.mayo.edu/research/documents/bsi-techreport-85/doc-20433003 Ref2: https://cran.r-project.org/web/packages/survival/vignettes/concordance.pdf

- Calculamos el valor para este modelo creado sólo con información de DEGs:

```r
cindex_expr_gen <- concordance(res.cox_DEGs1_final)
cindex_expr_gen[1]
```

```
## $concordance
## [1] 0.6602719
```

11. Creación modelo COX PH CLÍNICO: res.z_clinico_cox_final

Ahora vamos a construir un modelo con algunas variables clínicas. El modelo COx PH creado será en base a las variables ya categorizadas (valor 0 ó 1) según metodología de score.

11.1. Selección de variables clínicas para creación del modelo:

Las variables elegidas han sido: stage, T, N y edad del paciente ya categorizadas en riesgos según artículos de referencia. Ver:

Ref: https://onlinelibrary.wiley.com/doi/full/10.1111/iju.14859

```
# selección de variables clínicas relevantes:
z_clinico_cox_train <- df_cox_model_train %>%
  select(11:16)
rownames(z_clinico_cox_train) <- df_cox_model_train$Row.names

colnames(z_clinico_cox_train)
```

```
## [1] "t"                          "status"
## [3] "ajcc_pathologic_t_cat"      "ajcc_pathologic_n_cat"
## [5] "age_at_index_cat"           "ajcc_pathologic_stage_cat"
```

11.2 Creación del modelo:

- Con estas variables seleccionadas se crea un modelo clínico:

```
res.z_clinico_cox_final <- coxph(Surv(t, status) ~ ., data =  z_clinico_cox_train,iter.max=500 )
res.z_clinico_cox_final
```

```
## Call:
## coxph(formula = Surv(t, status) ~ ., data = z_clinico_cox_train,
##     iter.max = 500)
##
##                               coef exp(coef) se(coef)     z        p
## ajcc_pathologic_t_cat     0.03128   1.03177  0.21652 0.144  0.88515
## ajcc_pathologic_n_cat     0.99823   2.71348  0.35819 2.787  0.00532
## age_at_index_cat          0.46479   1.59168  0.16700 2.783  0.00538
## ajcc_pathologic_stage_cat 1.16419   3.20333  0.18586 6.264 3.76e-10
##
## Likelihood ratio test=70.83  on 4 df, p=1.515e-14
## n= 442, number of events= 153
```

11.3 Cálculo del índice de concordancia:

- Calculamos el índice de concordancia para este modelo basado en información exclusivamente clínica:

```
cindex_clinico <- concordance(res.z_clinico_cox_final)
cindex_clinico[1]
```

```
## $concordance
## [1] 0.7225999
```

12. Creación del modelo COX PH CLÍNICO + EXPRESIÓN GENÉTICA (COMBINADO): res.z_completo_cox_final

12.1. Selección de variables clínicas + expresión genéticas:

- El proceso es algo más complejo ya que requiere identificat tanto las variables clínicas como los DEGs usados en el mdelo del apartado 10 y estructurarlo en un único df:

```
# Esto es un proceso algo más complicado porque tengo que construir el dataframe
# con el que voy a entrenar al cox modelo. Primero extrayendo los datos clínicos
z_clinico_gen_cox_train <- df_cox_model_train %>%
  select(11:16)

# ahora extrayendo los genes que queremos:
temp <- matriz_cox_v2 %>%
  select(1:nrow(result_temporal2))

rownames(temp) <- df_cox_model_train$Row.names
rownames(z_clinico_gen_cox_train) <- df_cox_model_train$Row.names

# generando la unión de ambos
z_clinico_gen_cox_train2 <- merge(z_clinico_gen_cox_train,temp, by=0)

rownames(z_clinico_gen_cox_train2) <- df_cox_model_train$Row.names
z_clinico_gen_cox_train2 <- z_clinico_gen_cox_train2 %>%
  select(-1)
```

12.2 Creación del modelo:

- Con el df anterior, genero el modelo:

```
# aquí creo el modelo:
res.z_clinico_gen_cox_final <- coxph(Surv(t, status) ~ ., data =  z_clinico_gen_cox_train2,iter.max=500
res.z_clinico_gen_cox_final
```

```
## Call:
## coxph(formula = Surv(t, status) ~ ., data = z_clinico_gen_cox_train2,
##     iter.max = 500)
##
##                               coef  exp(coef)   se(coef)      z        p
## ajcc_pathologic_t_cat     -0.0415755  0.9592769  0.2286161 -0.182 0.855695
## ajcc_pathologic_n_cat      0.9420726  2.5652928  0.4109318  2.293 0.021875
## age_at_index_cat           0.4178067  1.5186270  0.1726905  2.419 0.015546
## ajcc_pathologic_stage_cat  1.0496068  2.8565277  0.1880892  5.580 2.40e-08
## ENSG00000234844.1         -0.0200873  0.9801132  0.0089639 -2.241 0.025032
## ENSG00000235207.1          0.0116703  1.0117387  0.0043894  2.659 0.007843
## ENSG00000189001.11         0.0007168  1.0007170  0.0002208  3.247 0.001167
## ENSG00000248227.1          0.0312169  1.0317093  0.0074782  4.174 2.99e-05
## ENSG00000286016.1          0.0101888  1.0102409  0.0029744  3.425 0.000614
## ENSG00000155269.12         0.0283041  1.0287085  0.0067387  4.200 2.67e-05
## ENSG00000170044.8         -0.0006239  0.9993762  0.0001531 -4.075 4.60e-05
##
```

```
## Likelihood ratio test=115.6  on 11 df, p=< 2.2e-16
## n= 442, number of events= 153
```

12.3. Cálculo del índice de concordancia:

- Calculamos el índice de concordancia para este nuevo modelo que es combinación de información genética y clínica:

```
cindex_combinado <- concordance(res.z_clinico_gen_cox_final)
cindex_combinado[1]
```

```
## $concordance
## [1] 0.7427057
```

Resumen output ETAPA3:

En esta etapa hemos generado tres modelos diferentes usando siempre el conjunto train (a partir del dataframe con información clínica y de expresión genética de *df_cox_model_train*):

- **res.cox_DEGs1_final**: Un modelo Cox PH exclusivo con información de la expresión genética, usando para ello un subgrupo de las DEGs obtenidas inicialmente que se identifican en el propio proceso de creación del modelo como aquellos que aparecen como estadísticamente significativos.
- **res.z_clinico_cox_final**: Un modelo Cox PH exclusivo con información clínica a partir de algunas variables del paciente que han sido previamente categorizadas siguiendo la filosofía de asignación de score (al estilo GRANT o Leibovich).
- **res.z_completo_cox_final**: Un modelo Cox PH donde se combina la información clínica con la información de la expresión genética.

En este apartado a su vez hemos calculado el c-index para cada uno de los modelos Cox PH creados. A través del índice de concordancia de los modelos Cox PH creados podemos tener una estimación de cuánto de buenos son nuestros modelos para ordenar los datos del propio conjunto train con el que son creados. Es el equivalente al área bajo la curva ROC. Como podemos ver, hemos tenido una mejora en los modelos una vez incluída la parte de expresión genética al modelo clínico base.

ETAPA 4. EVALUACIÓN DE MODELOS COX A PARTIR DE CURVAS KAPLAN-MEIER: EXPRESIÓN GENÉTICA, CLÍNICO Y COMBINADO

1) Para evaluar los **modelos Cox PH** vamos a seguir la siguiente medtodología (apartados 13 a 15):

- Elegimos el grupo de datos sobre el que queremos evaluar el modelo.
- Pasamos dicho conjunto de datos por la función predict() sobre el modelo a evaluar.
- Asignamos una etiqueta de riesgo en función del resultado del predict():
  - Se asigna la etiqueta RIESGO ALTO a aquellos pacientes cuyo valor predictivo en el modelo COX haya salido mayor que 1.
  - Se asigna la etiqueta RIESGO BAJO a aquellos pacientes cuyo valor predictivo en el modelo COX haya salido menor o igual a 1.

A la hora de estructurar esta etapa en el documento se ha decidido agrupar los apartados de las evaluaciones de los modelos no por cada modelo en sí, sino por el grupo de datos sobre el que se está evaluando dicho modelo, entendiendo que así se facilita la capacidad de comparación de resultados entre distintos modelos sobre el mismo conjunto de datos.

2) Para evaluar el **modelo SCORE clínico** y su posible mejora al añadir un SCORE de expresión genética vamos a usar la siguiente metodología (apartado 16):

- Usaremos exclusivamente el grupo test.
- A partir de los resultados del aparatado 14, asignamos un score de riesgo asociado exclusivamente a la expresión genética que lo añadiremos al score asociado a variables clínicas.
- La forma de combinar ambos scores es la siguiente: si alguno de los dos es score alto, el paciente quedao como riesgo alto.

13. Evaluación de los modelos COX sobre el conjunto TRAIN:

13.1. Evaluación en TRAIN modelo COX PH EXPRESIÓN GENÉTICA: res.cox_DEGs1_final:

- Pasamos el conjunto TRAIN por el predict de este modelo y añadimos etiqueta de riesgo según modelo:

```
df_predict_train <- data.frame(resultado=predict(res.cox_DEGs1_final,matriz_cox_v2, type = "risk"))
df_predict_train <- df_predict_train %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))

table(df_predict_train$risk)
```

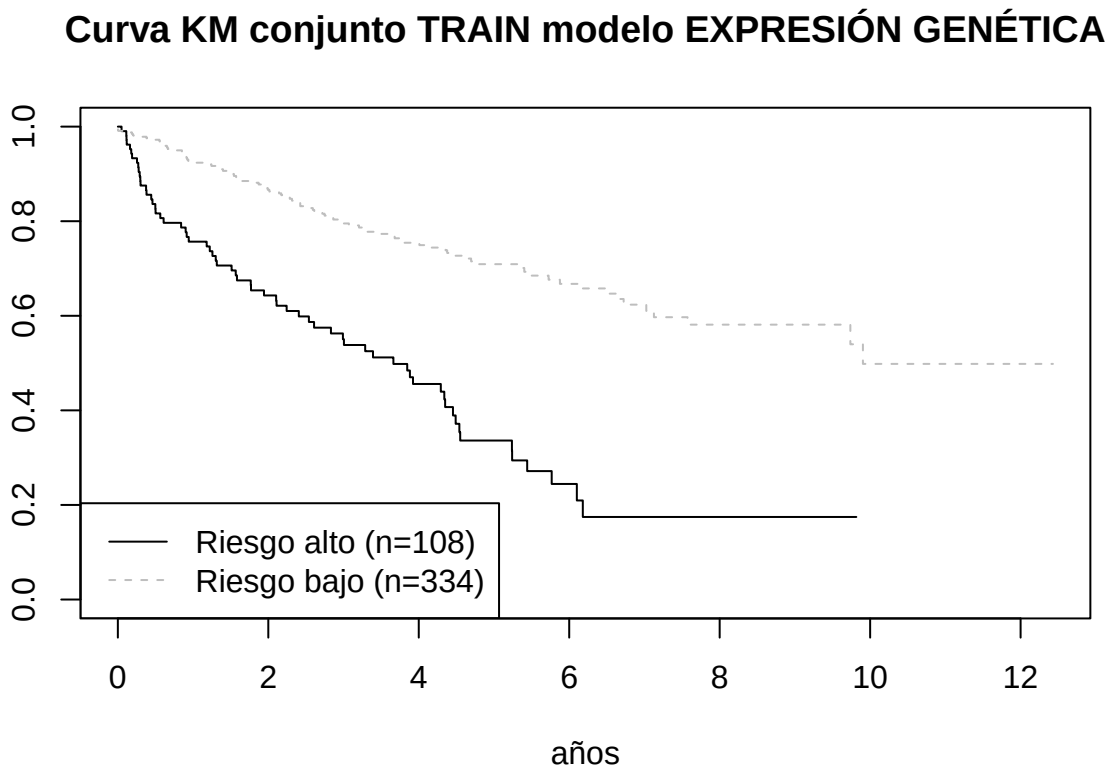
```
##
## alto bajo
##  108  334
```

```r
# Unimos las predicciones al conjunto de datos del que procede y graficamos curva KM:

union_train_gen <- merge(matriz_cox_v2,df_predict_train, by=0)

km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_train_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=108)", "Riesgo bajo (n=334)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa. Para ello usamos el método log-rank:

Ref: https://iagolast.github.io/blog/2019/01/13/kaplan-meier.html

```r
# método log-rank:

logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_train_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 3.619327e-13
```

13.2 Evaluación en TRAIN modelo COX PH CLÍNICO: res.z_clinico_cox_final

- Pasamos el conjunto TRAIN por el predict de este modelo:

```
z_predict_clinical_train <- data.frame(resultado=predict(res.z_clinico_cox_final,z_clinico_cox_train, ty
z_predict_clinical_train <- z_predict_clinical_train %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))
rownames(z_predict_clinical_train) <- df_cox_model_train$Row.names

table(z_predict_clinical_train$risk)
```

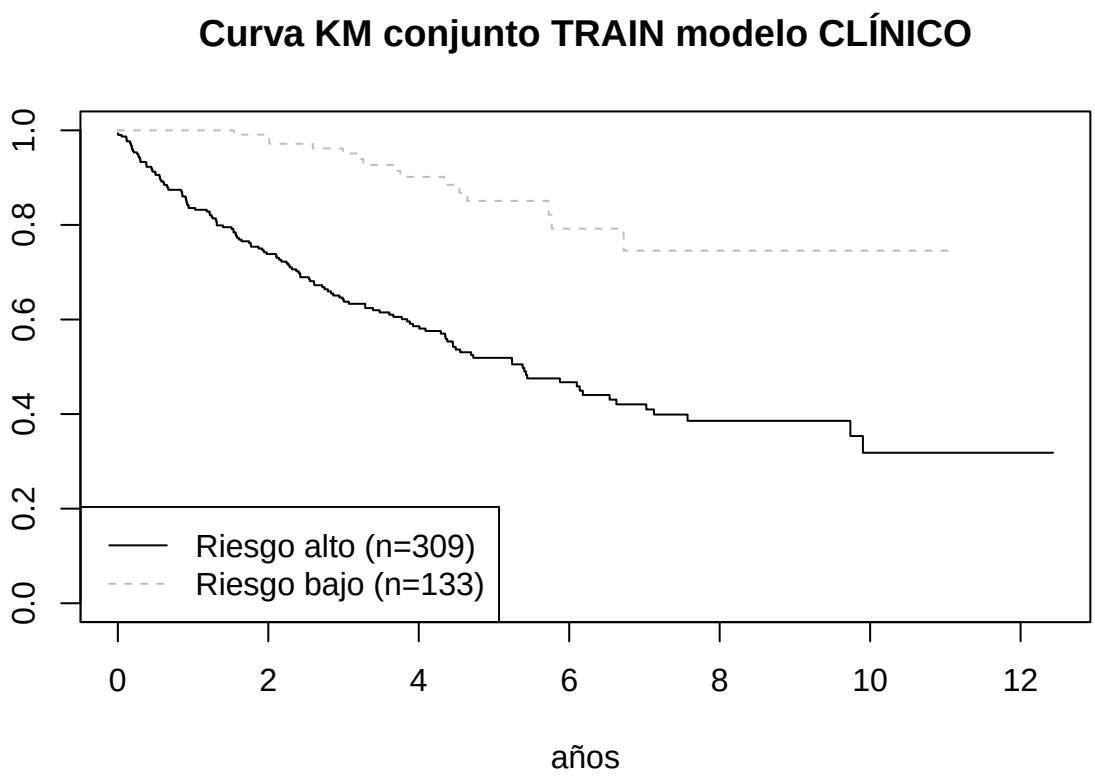
```
##
## alto bajo
##  309  133
```

```
# Unión y curva KM

union_train_clinical <- merge(z_clinico_cox_train,z_predict_clinical_train, by=0)

km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_train_clinical)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjunt
 legend("bottomleft", c("Riesgo alto (n=309)", "Riesgo bajo (n=133)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_train_clinical)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 3.112495e-10
```

13.3 Evaluación en TRAIN modelo COX PH EXPRESIÓN GENÉTICA + CLÍNICO (COMBINADO): res.z_clinico_gen_cox_final

- Pasamos el conjunto TRAIN por el predict de este modelo:

```
z_predict_clinico_gen_cox_train <- data.frame(resultado=predict(res.z_clinico_gen_cox_final,z_clinico_g

z_predict_clinico_gen_cox_train <- z_predict_clinico_gen_cox_train %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))
rownames(z_predict_clinico_gen_cox_train) <- df_cox_model_train$Row.names

table(z_predict_clinico_gen_cox_train$risk)
```

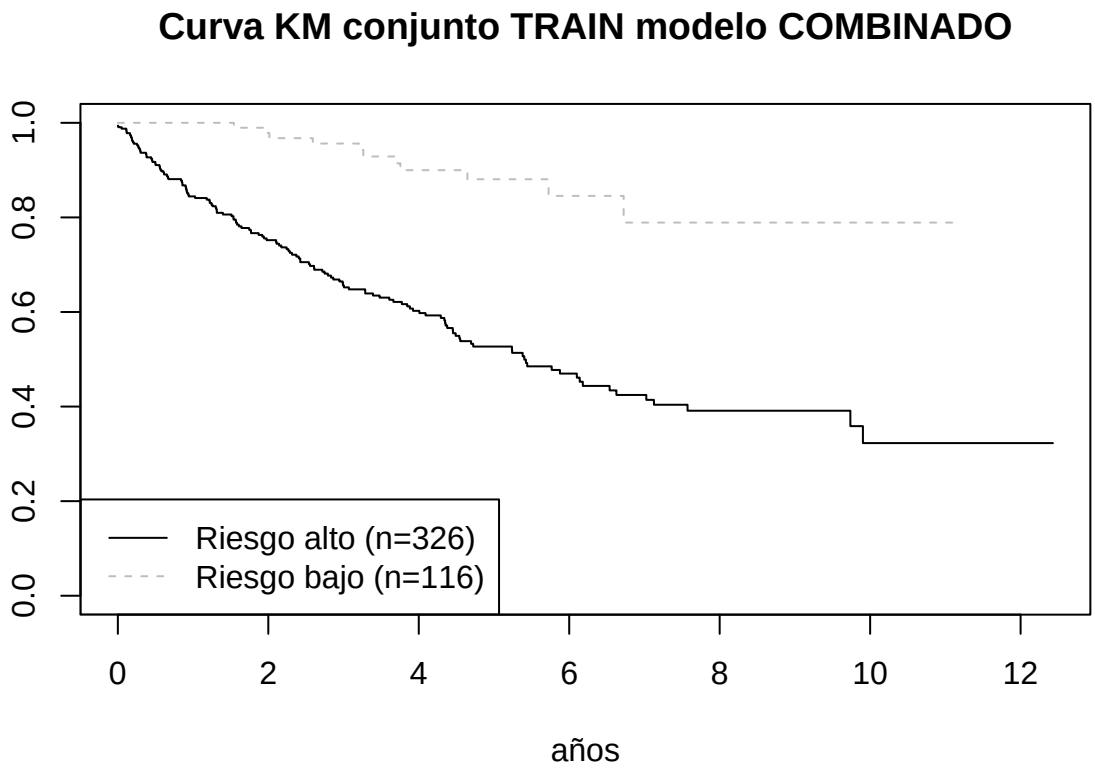
```
##
## alto bajo
##  326  116
```

```
# Creación curva KM:

union_train_clinical_gen <- merge(z_clinico_gen_cox_train2,z_predict_clinico_gen_cox_train, by=0)

km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_train_clinical_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=326)", "Riesgo bajo (n=116)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_train_clinical_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 1.12439e-09
```

14. Evaluación de los modelos COX sobre el conjunto TEST:

En este caso revisamos cuánto de predictivo es cada uno de nuestros modelos sobre un conjunto de datos no visto previamente por el modelo creado (df_cox_model_test). Este es uno de los elementos fundamentales a evaluar, la capacidad predictiva de nuestros modelo a la hora de generar etiquetas de bajo/alto riesgo.

14.1. Evaluación en TEST modelo COX PH EXPRESIÓN GENÉTICA: res.cox_DEGs1_final:

- Pasamos el conjunto TEST por el predict de este modelo:

```
df_predict_test <- data.frame(resultado=predict(res.cox_DEGs1_final,df_cox_model_test, type = "risk"))
df_predict_test <- df_predict_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))
rownames(df_predict_test) <- df_cox_model_test$Row.names

table(df_predict_test$risk)
```

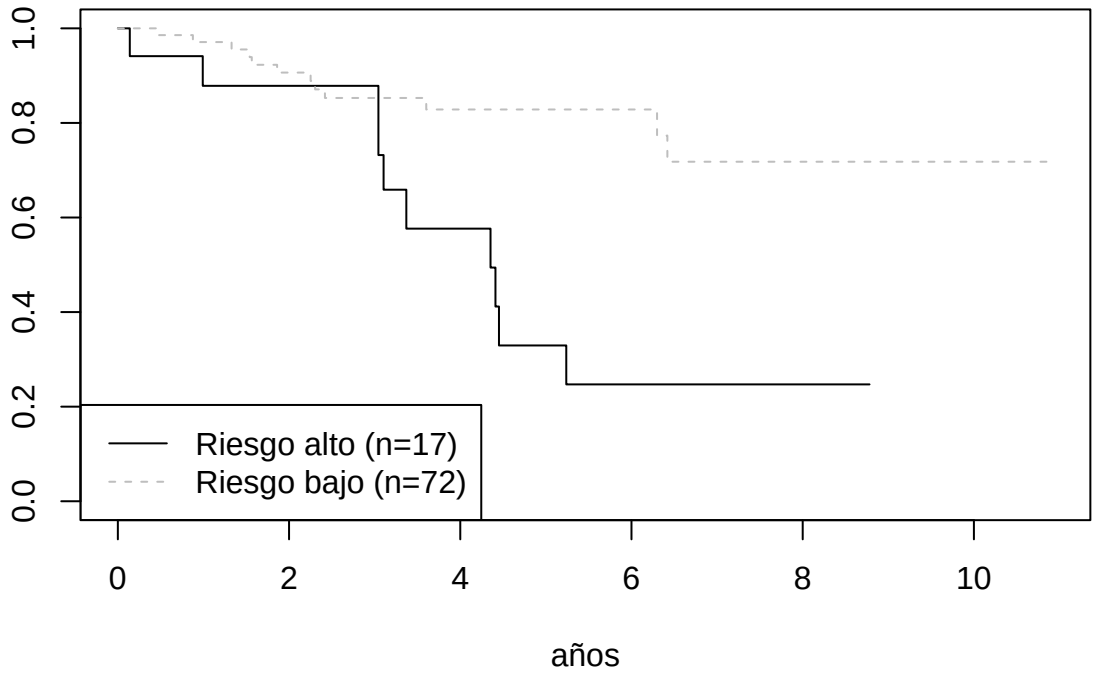
```
##
## alto bajo
##   17   72
```

```
# Curva KM:
union_test_gen <- merge(df_cox_model_test,df_predict_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test, df_predict_test, by = 0): column
## name 'Row.names' is duplicated in the result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=17)", "Riesgo bajo (n=72)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.001067491
```

Como podemos observar, el modelo con información exclusiva de la expresión genética es capaz de diferenciar en las curvas de sobrevida entre pacientes con bajo y alto riesgo

14.2 Evaluación en TEST modelo COX PH CLÍNICO: res.z__clinico__cox__final

- Pasamos el conjunto TEST por el predict de este modelo:

```
z_predict_clinical_test <- data.frame(resultado=predict(res.z_clinico_cox_final,df_cox_model_test, type

z_predict_clinical_test <- z_predict_clinical_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))

rownames(z_predict_clinical_test) <- df_cox_model_test$Row.names
```

```r
table(z_predict_clinical_test$risk)
```

```
##
## alto bajo
##   50   39
```

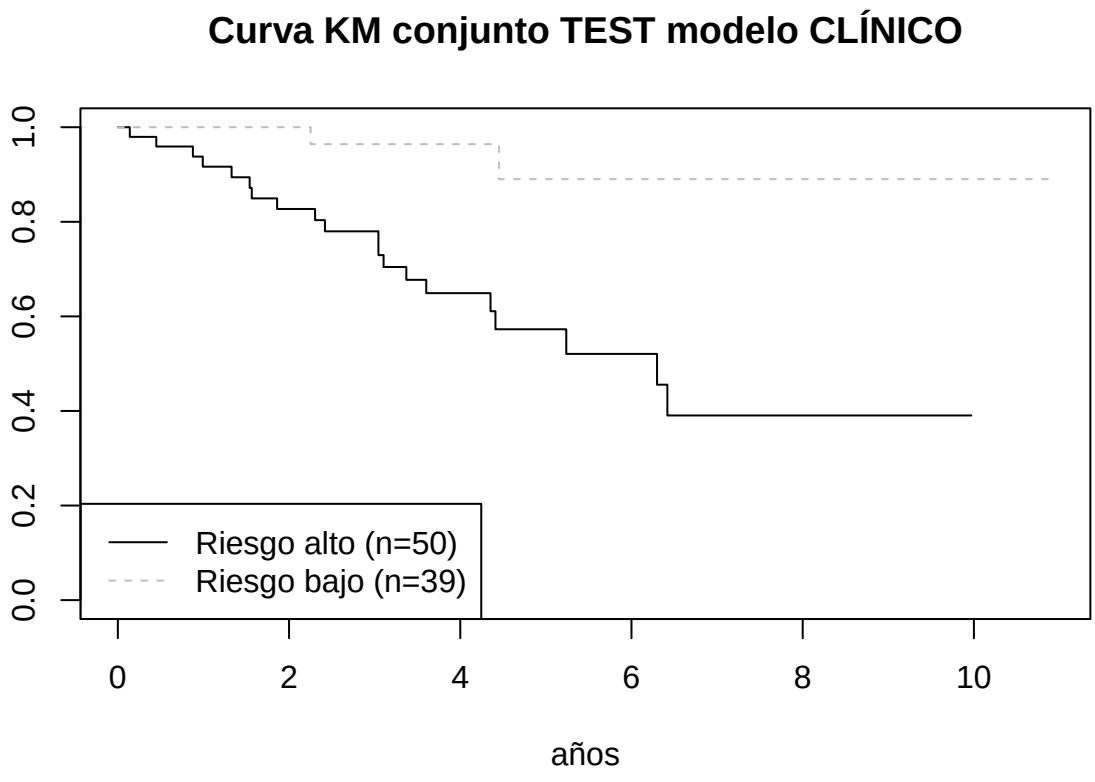
```r
# Graficamos la curva KM:

union_test_clinical <- merge(df_cox_model_test,z_predict_clinical_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test, z_predict_clinical_test, : column
## name 'Row.names' is duplicated in the result
```

```r
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_clinical)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=50)", "Riesgo bajo (n=39)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```r
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_clinical)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.000593552
```

Como vemos, el modelo con información exclusivamente clínica es capaz de diferencia también en un conjunto de datos no vistos previamente

14.3 Evaluación en TEST Modelo COX PH EXPRESIÓN GENÉTICA + CLÍNICO (COMBINADO): res.z_clinico_gen_cox_final

- Pasamos el conjunto TEST por el predict de este modelo:

```
z_predict_clinical_gen_test <- data.frame(resultado=predict(res.z_clinico_gen_cox_final,df_cox_model_tes

z_predict_clinical_gen_test <- z_predict_clinical_gen_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))

rownames(z_predict_clinical_gen_test) <- df_cox_model_test$Row.names

table(z_predict_clinical_gen_test$risk)
```

```
##
## alto bajo
##   54   35
```

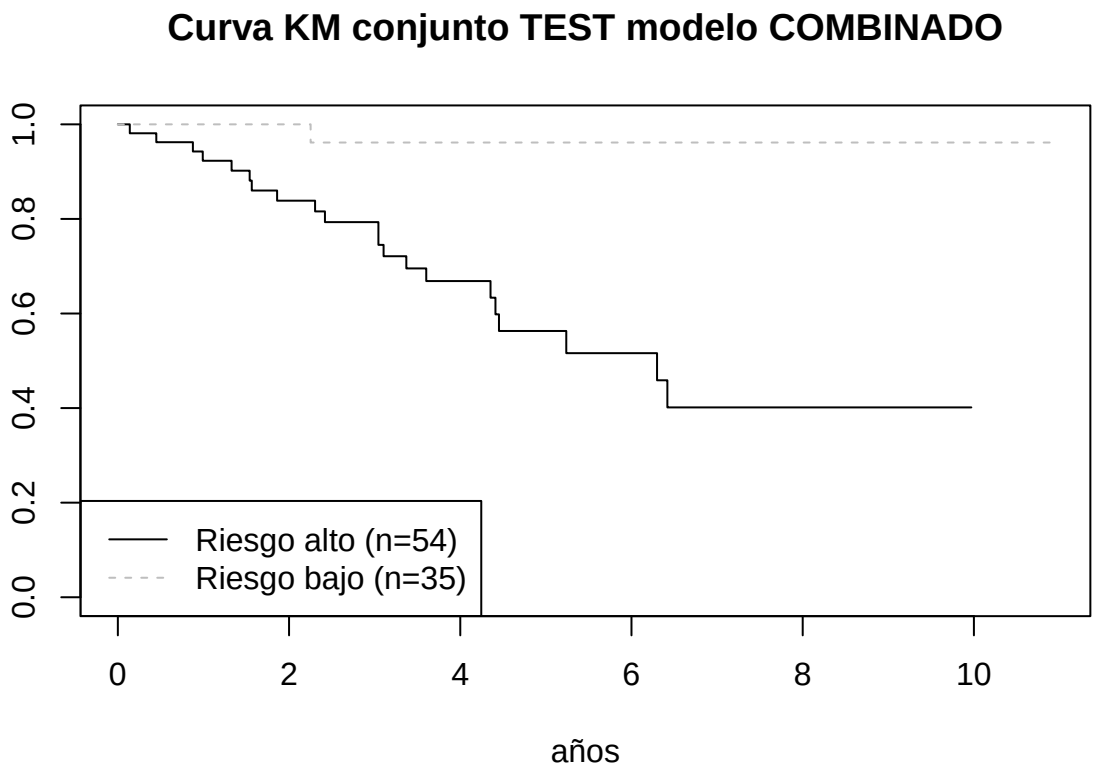
```
# Curva KM:

union_test_clinical_gen <- merge(df_cox_model_test,z_predict_clinical_gen_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test, z_predict_clinical_gen_test, :
## column name 'Row.names' is duplicated in the result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_clinical_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=54)", "Riesgo bajo (n=35)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_clinical_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.0005193294
```

15. Evaluación de los modelos COX sobre el conjunto TEST filtrado (sólo stage I y II):

- Nos interesa por último ver la capacidad predictiva en el grupo test pero sólo de los pacientes con estadío I y II, es decir aquellos que llegan en los estadíos más tempranos, para intentar identificar factores de mayor riesgo
- El dataframe ya estaba creado y era: df_cox_model_test_stage

15.1. Evaluación sobre TEST (I y II) modelo COX PH EXPRESIÓN GENÉTICA: res.cox_DEGs1_final:

- Pasamos el conjunto conjunto **TEST (sólo stage I y II)** por el predict de este modelo:

```
df_predict_test <- data.frame(resultado=predict(res.cox_DEGs1_final,df_cox_model_test_stage, type = "ris
df_predict_test <- df_predict_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))
rownames(df_predict_test) <- df_cox_model_test_stage$Row.names

table(df_predict_test$risk)
```

```
##
## alto bajo
##   10   55
```

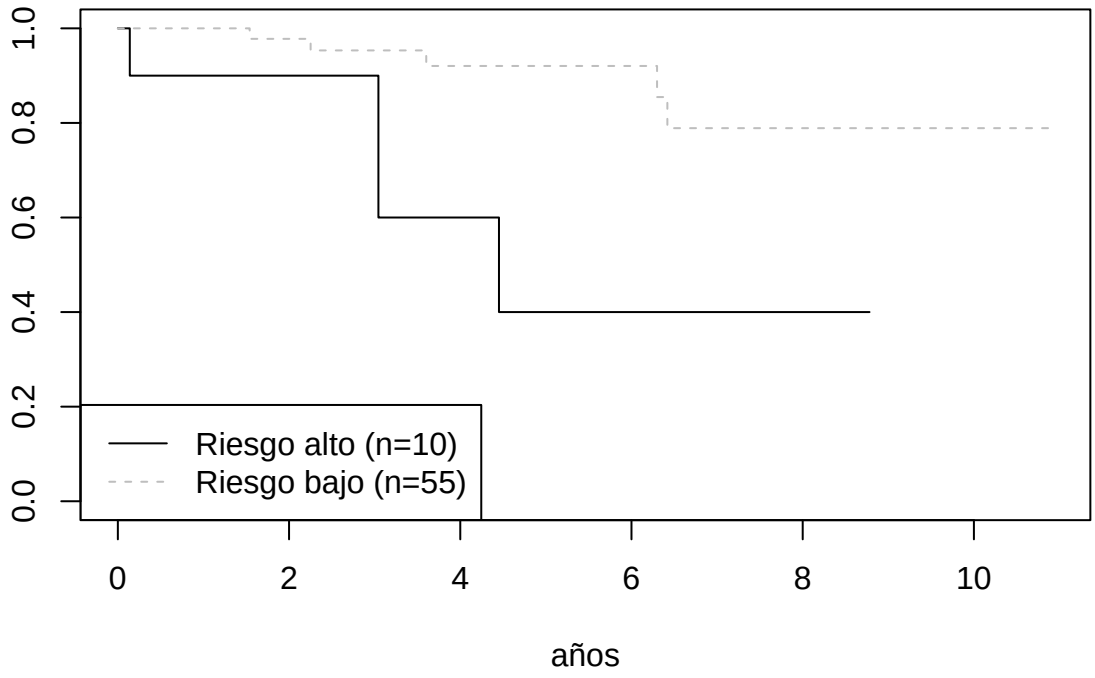
```
# Curva KM:

union_test_gen_stage <- merge(df_cox_model_test_stage,df_predict_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test_stage, df_predict_test, by = 0):
## column name 'Row.names' is duplicated in the result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_gen_stage)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=10)", "Riesgo bajo (n=55)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_gen_stage)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.002777111
```

15.2 Evaluación sobre TEST (I y II) modelo COX PH CLÍNICO: res.z_clinico_cox_final

- Pasamos el conjunto conjunto **TEST (sólo stage I y II)** por el predict de este modelo:

```
z_predict_clinical_test <- data.frame(resultado=predict(res.z_clinico_cox_final,df_cox_model_test_stage

z_predict_clinical_test <- z_predict_clinical_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))

rownames(z_predict_clinical_test) <- df_cox_model_test_stage$Row.names

table(z_predict_clinical_test$risk)
```

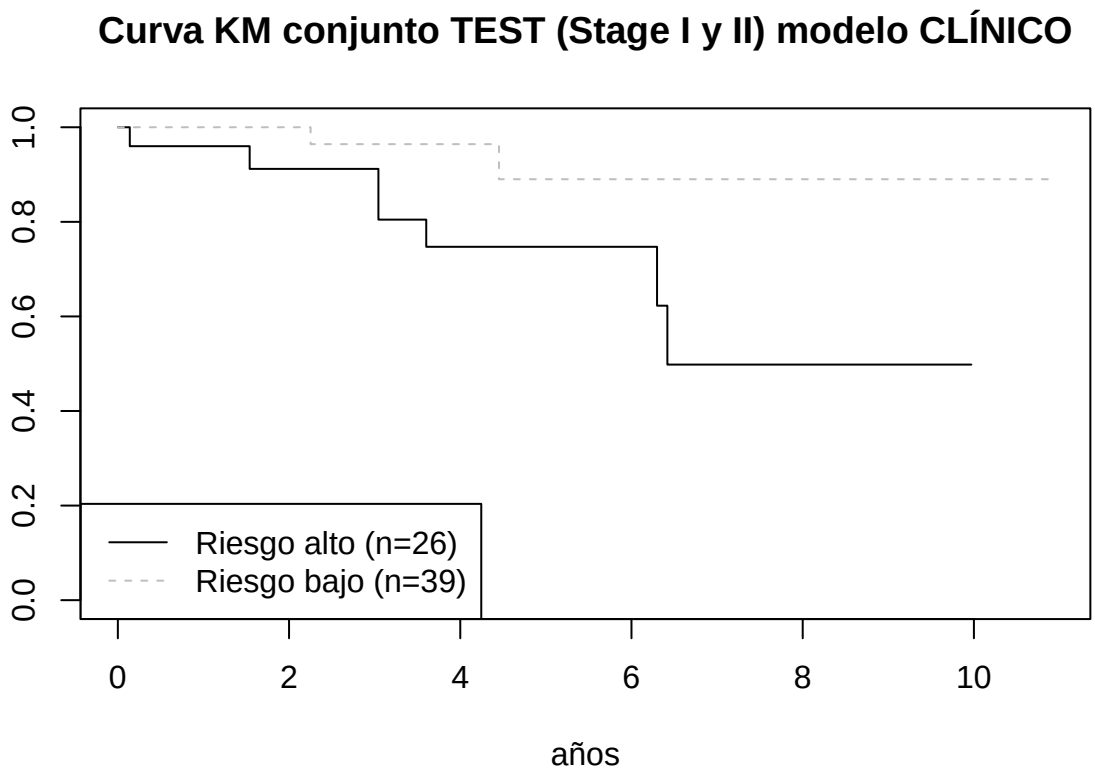
```
##
## alto bajo
##   26   39
```

```
# Graficamos curva KM:
union_test_clinical_stage <- merge(df_cox_model_test_stage,z_predict_clinical_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test_stage, z_predict_clinical_test, :
## column name 'Row.names' is duplicated in the result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_clinical_stage)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=26)", "Riesgo bajo (n=39)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_clinical_stage)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.0234534
```

15.3 Evaluación sobre TEST (I y II) modelo COX PH EXPRESIÓN GENÉTICA + CLÍNICO (COMBINADO): res.z_clinico_gen_cox_final

- Pasamos el conjunto conjunto **TEST (sólo stage I y II)** por el predict de este modelo:

```
z_predict_clinical_gen_test <- data.frame(resultado=predict(res.z_clinico_gen_cox_final,df_cox_model_tes

z_predict_clinical_gen_test <- z_predict_clinical_gen_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))

rownames(z_predict_clinical_gen_test) <- df_cox_model_test_stage$Row.names

table(z_predict_clinical_gen_test$risk)
```

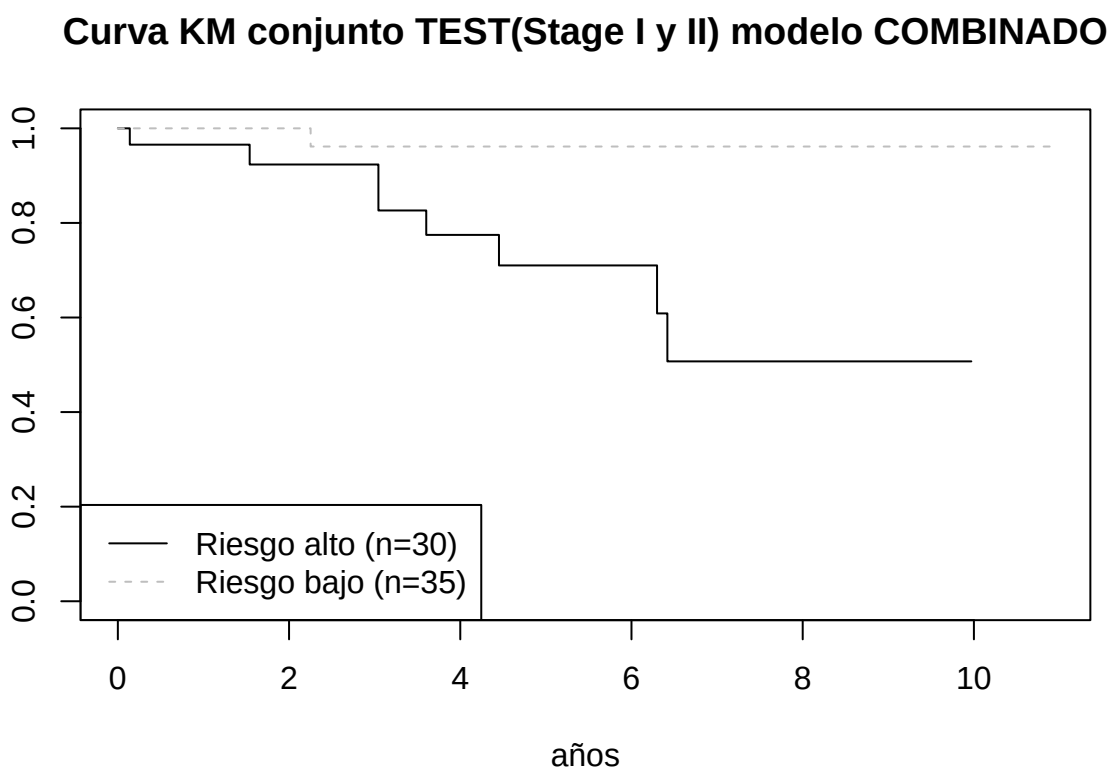
```
##
## alto bajo
##   30   35
```

```
# Curva KM:
union_test_clinical_gen_stage <- merge(df_cox_model_test_stage,z_predict_clinical_gen_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test_stage,
## z_predict_clinical_gen_test, : column name 'Row.names' is duplicated in the
## result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_clinical_gen_stage)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=30)", "Riesgo bajo (n=35)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_clinical_gen_stage)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.01138307
```

16 Evaluación en TEST de la mejora en el modelo SCORE al añadir los resultados del modelo Cox PH genético:

- El modelo score ha sido calculado al crear la variable *risk_clinical_score* en función de las variables clínicas. Vamos a ver si esta etiqueta (que no es creada a partir de un modelo COX) sino a partir de la asignación de un score de riesgo nos sirve también como etiqueta que discrimina en la curva Kaplan-Meire de los pacientes del grupo test:

16.1 Kaplan - Meier del modelo SCORE:

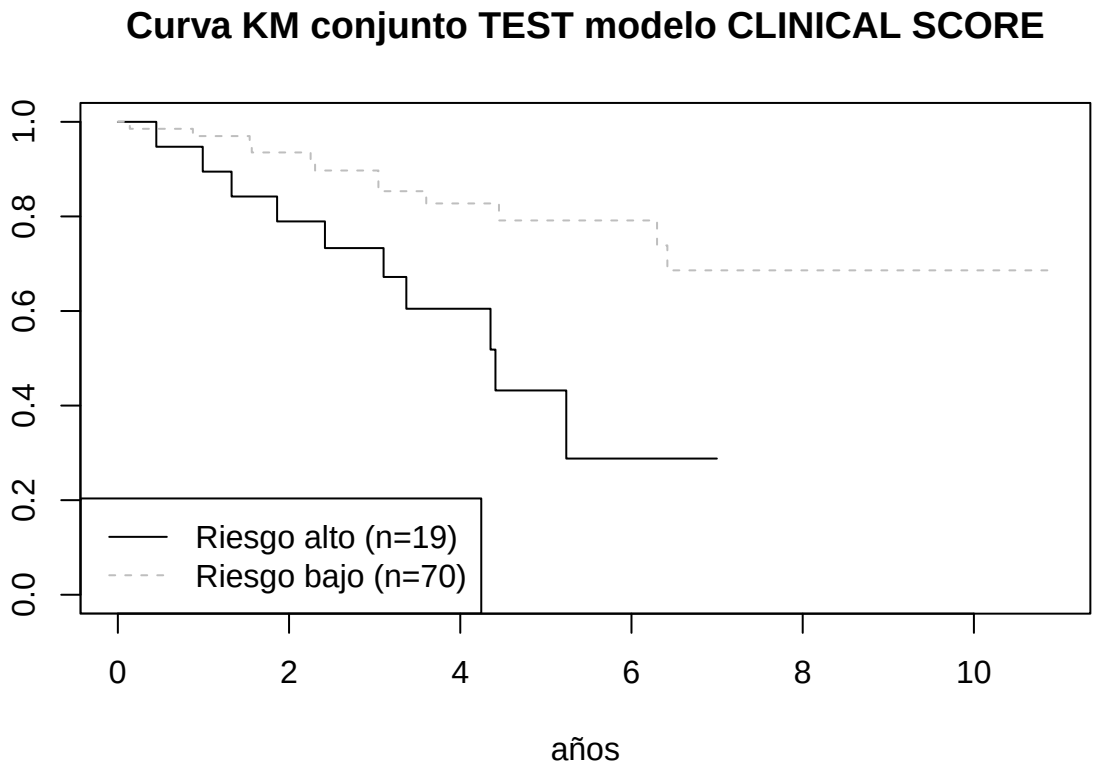
```
union_test_gen_score <- union_test_gen %>%
  select(1,12,13,19,60685)

table(union_test_gen_score$risk_clinical_score)
```

```
##
##  riego clinico alto riesgo clinico bajo
##                  19                   70
```

```r
km_fit <- survfit(Surv(t/365, status) ~ risk_clinical_score, data=union_test_gen_score)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=19)", "Riesgo bajo (n=70)"), lty = c("solid", "dashed"), col =
```




```r
 # método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk_clinical_score, data = union_test_gen_score)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.004207043
```

16.1 Kaplan - Meier del modelo SCORE + etiqueta GENÉTICA:

- Combinamos ambas etiquetas para identificar aquellos pacientes que tengan riesgo alto combinado (es decir, teniendo en cuenta la información clínica como de expresión genética). El algoritmo de cálculo del score combinado es el que comentamos al comienzo de la etapa 4: si alguno de los dos score es alto, el paciente tendrán un riesgo combinado alto:

```
union_test_gen_score <- union_test_gen_score %>%
  mutate(risk_combinado=case_when(
    (union_test_gen_score$risk_clinical_score=="riego clinico alto" | union_test_gen_score$risk=="alto"
    TRUE ~ "riesgo combinado bajo"
  ))
```

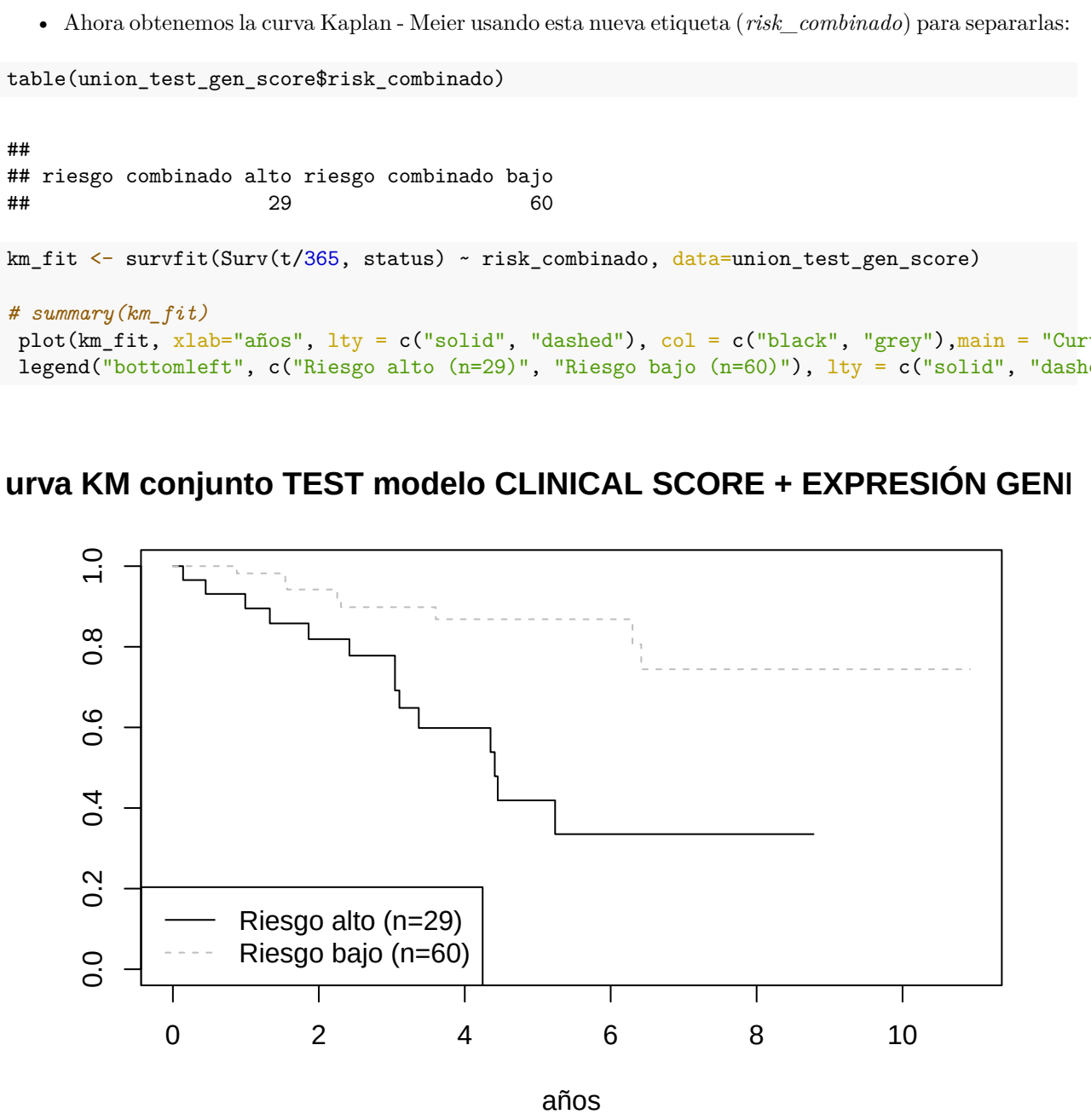
- Ahora obtenemos la curva Kaplan - Meier usando esta nueva etiqueta (*risk_combinado*) para separarlas:

```
table(union_test_gen_score$risk_combinado)
```

```
##
## riesgo combinado alto riesgo combinado bajo
##                    29                    60
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk_combinado, data=union_test_gen_score)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=29)", "Riesgo bajo (n=60)"), lty = c("solid", "dashed"), col =
```




```
 # método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk_combinado, data = union_test_gen_score)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.0005567063
```

Resumen output ETAPA4:

En esta etapa se ha evaluado la bondad de tres modelos Cox PH diferentes en discriminar curvas Kaplan-Meier a partir de la categorización realizada por los modelos (apartados 13 a 15). A su vez, se ha evaluado la bondad del modelo basado en un score clínico y su cambio cuando se combina con un score de expresión genética que procede del modelo Cox PH (apartado 16).

CONCLUSIONES:

Vamos a destacar algunas conclusiones obtenidas:

- A partir de los DEGs determinados entre pacientes con y sin metástasis, creamos un modelo Cox PH sólo con información de expresión genética que, tras etiquetar como alto/bajo riesgo, permite separar las curvas Kaplan-Meier de ambos subgrupos.
- Este modelo funciona tanto para el conjunto test, como para el conjunto test reducido (donde sólo se analizan los pacientes en estadios precoces de la enfermedad)
- Cuando se incorporan los DEGs al modelo clínico Cox PH (que a su vez también tiene la capacidad de discriminar las curvas Kaplan-Meier de los pacientes del grupo test) para crear un nuevo modelo combinado, se produce una mejora en la separación de dichas curvas.
- A su vez, cuando se sigue una metodología de score tanto para la parte clínica como para la parte genética, se observa una mejoría en el modelo que posee sólo información clínica.

ANEXOS

Anexo 1. Análisis influencia partición de datos en cálculo de DEGs

Se muestra en el anexo el análisis que se realizó sobre la influencia de la partición de los datos utilizados en la determinación de los DEGs. Es importante analizar de alguna manera este elemento.

A1.1 Creación de los 5 grupos test/train:

- Se utiliza función para la separación del dataframe original en 5 subgrupos distintos.

La función procede de https://stackoverflow.com/questions/20041239/how-to-randomly-split-a-data-frame-into-three-smaller-ones-with-given-numbers-of

```
#' Splits data.frame into arbitrary number of groups
#'
#' @param dat The data.frame to split into groups
#' @param props Numeric vector. What proportion of the data should
#'              go in each group?
#' @param which.adjust Numeric. Which group size should we 'fudge' to
#'              make sure that we sample enough (or not too much)
split_data <- function(dat, props = c(.2,.2,.2,.2, .2), which.adjust = 1){

    # Make sure proportions are positive
    # and the adjustment group isn't larger than the number
    # of groups specified
    stopifnot(all(props >= 0), which.adjust <= length(props))

    # could check to see if the sum is 1
    # but this is easier
    props <- props/sum(props)
    n <- nrow(dat)
    # How large should each group be?
    ns <- round(n * props)
    # The previous step might give something that
    # gives sum(ns) > n so let's force the group
    # specified in which.adjust to be a value that
    # makes it so that sum(ns) = n
    ns[which.adjust] <- n - sum(ns[-which.adjust])

    ids <- rep(1:length(props), ns)
    # Shuffle ids so that the groups are randomized
    which.group <- sample(ids)
    split(dat, which.group)
}
```

- Aplicamos la función para obtener los 5 subgrupos de datos

```
set.seed(42)
data_split <- split_data(df_clinical_no_metas)
```

- Agrupamos en 5 conjuntos de train/test combinando los 5 subgrupos anteriores:

```
train_1 <- rbind(data_split$`1`,data_split$`2`,data_split$`3`,data_split$`4`)
test_1 <- data_split$`5`

train_2 <- rbind(data_split$`2`,data_split$`3`,data_split$`4`,data_split$`5`)
test_2 <- data_split$`1`

train_3 <- rbind(data_split$`3`,data_split$`4`,data_split$`5`,data_split$`1`)
test_3 <- data_split$`2`

train_4 <- rbind(data_split$`4`,data_split$`5`,data_split$`1`,data_split$`2`)
test_4 <- data_split$`3`

train_5 <- rbind(data_split$`5`,data_split$`1`,data_split$`2`,data_split$`3`)
test_5 <- data_split$`4`
```

Este conjunto de datos train es lo que usaremos para identificar qué DEGs se repiten en el cálculo.

A1.2 Creación función para proceso de determinación de DEGs:

- La función devuelve el listado de DEGs para un determinado grupo train. Los parámetros de entrada son el los valores de corte para que queden identificado o no como DEGs:

```
calculo_genes_cox_ok <- function(train_f,v_log,v_padj){
clinical_train_FUNCION <- train_f %>%
  rbind(df_clinical_metas)

clinical_train_FUNCION<- clinical_train_FUNCION[ order(row.names(clinical_train_FUNCION)), ]

#450
# Obtenemos el df de raw_train:
id_comunes_train <- intersect(colnames(df_raw_final),rownames(clinical_train_FUNCION))
names.use_train <- names(df_raw_final)[(names(df_raw_final) %in% id_comunes_train)]
raw_train_FUNCION <- df_raw_final[, names.use_train]

# 516. Calculo de DEGs sólo con train:
df_clinical_new_F <- clinical_train_FUNCION
df_matriz_gen_new_F <- raw_train_FUNCION

# 523. Para que funcione:
# en el raw data:
df_matriz_gen_new_F <- mutate_all(df_matriz_gen_new_F, function(x) as.integer(x))

# en el clinical data:
df_clinical_new_F$metastasis_a_fecha_diagnostico <- as.factor(df_clinical_new_F$metastasis_a_fecha_diagn

#dds del train:
dds_F <- DESeqDataSetFromMatrix(countData = df_matriz_gen_new_F,
                                colData = df_clinical_new_F,
                                design = ~ metastasis_a_fecha_diagnostico)
# DESeq para train:
dds2_F <- DESeq(dds_F)
res_F <- results(dds2_F)
```

```
# 552. Valores normalizados para train:
norm_train_F <- data.frame(counts(dds2_F, normalized=TRUE))
colnames(norm_train_F) <- colnames(raw_train_FUNCION)

# 563. Vemos cuántos DEGs elegimos:
analisis_m_F <- data.frame(df_raw$gene_id,df_raw$gene_name,df_raw$gene_type, res_F$baseMean,res_F$log2F

analisis_m_F$abs_log2FoldChange <- abs(analisis_m_F$res_F.log2FoldChange)

DEGs_metas_F <- analisis_m_F %>%
  filter(abs_log2FoldChange > v_log & res_F.padj < v_padj)

DEGs_metas_F <- DEGs_metas_F %>%
  arrange(df_raw.gene_name)

# 593. Algunas modficiaciones:
norm_train_cox_T_F <-as.data.frame(t(norm_train_F))

df_cox_model_train_F <- merge(clinical_train_FUNCION, norm_train_cox_T_F, by=0, all=TRUE)
rownames(df_cox_model_train_F) <- df_cox_model_train_F$Row.names

norm_test_cox_T_F <-as.data.frame(t(norm_test_FUNCION))

df_cox_model_test_F <- merge(clinical_test_FUNCION, norm_test_cox_T_F, by=0, all=TRUE)
rownames(df_cox_model_test_F) <- df_cox_model_test_F$Row.names

# ETAPA 3. CUÁNTOS DEGS ELEGIMOS:
DEGs_metas_cox_F <- DEGs_metas_F %>%
  arrange(desc(abs_log2FoldChange))

return(DEGs_metas_cox_F)}
```

A1.3 Apliación la función a los 5 subgrupos train:

- Calculamos la matriz de resultados de DEGs de cada una de las subdivisiones y guardamos en .csv:

```
grupo_1 <- calculo_genes_cox_ok(train_1,1,0.001)
write.csv2(grupo_1,"OUTPUT_DEGS_COX/degs_g1.csv")

grupo_2 <- calculo_genes_cox_ok(train_2,1,0.001)
write.csv2(grupo_2,"OUTPUT_DEGS_COX/degs_g2.csv")

grupo_3 <- calculo_genes_cox_ok(train_3,1,0.001)
write.csv2(grupo_3,"OUTPUT_DEGS_COX/degs_g3.csv")

grupo_4 <- calculo_genes_cox_ok(train_4,1,0.001)
write.csv2(grupo_4,"OUTPUT_DEGS_COX/degs_g4.csv")

grupo_5 <- calculo_genes_cox_ok(train_5,1,0.001)
write.csv2(grupo_5,"OUTPUT_DEGS_COX/degs_g5.csv")
```

- Cargamos los .csv con la información de los DEGs de cada grupo:

```
degs_g1 <- read.csv2("OUTPUT_DEGS_COX/degs_g1.csv")
degs_g2 <- read.csv2("OUTPUT_DEGS_COX/degs_g2.csv")
degs_g3 <- read.csv2("OUTPUT_DEGS_COX/degs_g3.csv")
degs_g4 <- read.csv2("OUTPUT_DEGS_COX/degs_g4.csv")
degs_g5 <- read.csv2("OUTPUT_DEGS_COX/degs_g5.csv")
```

- Obtenemos los valores de DEGs comunes en los 5 subgrupos

```
degs_comunes <- Reduce(intersect, list(degs_g1$df_raw.gene_id,degs_g2$df_raw.gene_id,degs_g3$df_raw.gene
                                       degs_g4$df_raw.gene_id,degs_g5$df_raw.gene_id))
```

En este caso tenemos un total de 189 DEGs que han aparecido en los 5 cálculos.

- Filtramos cada uno de ellos por ese conjunto de DEGs que son comunes:

```
degs_g1_comun <- degs_g1 %>%
  filter(degs_g1$df_raw.gene_id %in% degs_comunes) %>%
  arrange(df_raw.gene_id) %>%
  select(2,11)

degs_g2_comun <- degs_g2 %>%
  filter(degs_g2$df_raw.gene_id %in% degs_comunes) %>%
  arrange(df_raw.gene_id)%>%
  select(2,11)

degs_g3_comun <- degs_g3 %>%
  filter(degs_g3$df_raw.gene_id %in% degs_comunes) %>%
  arrange(df_raw.gene_id)%>%
  select(2,11)

degs_g4_comun <- degs_g4 %>%
  filter(degs_g4$df_raw.gene_id %in% degs_comunes) %>%
  arrange(df_raw.gene_id)%>%
  select(2,11)

degs_g5_comun <- degs_g5 %>%
  filter(degs_g5$df_raw.gene_id %in% degs_comunes) %>%
  arrange(df_raw.gene_id)%>%
  select(2,11)
```

- Agrupamos en un conjunto, los resultados del abs_log2FoldChange para los 5 grupos:

```
emer <- merge(degs_g1_comun,degs_g2_comun,by="df_raw.gene_id")
emer <- merge(emer,degs_g3_comun,by="df_raw.gene_id")
emer <- merge(emer,degs_g4_comun,by="df_raw.gene_id")
degs_comunes_5grupos <- merge(emer,degs_g5_comun,by="df_raw.gene_id")
colnames(degs_comunes_5grupos)[2:6] <- c("g1", "g2","g3", "g4","g5")
```

- Revisamos cómo son los valores logs, y calculamos el promedio:

```
degs_comunes_5grupos2 <- degs_comunes_5grupos %>% select(-1)
degs_comunes_5grupos2$promedio <- rowMeans(degs_comunes_5grupos2)
rownames(degs_comunes_5grupos2) <- degs_comunes_5grupos$df_raw.gene_id
degs_comunes_5grupos2 <- degs_comunes_5grupos2 %>% arrange(desc(promedio))
nrow(degs_comunes_5grupos2)
```

```
## [1] 189
```

Como podemos observar, hay 189 DEGs que han aparecido en los 5 análisis.

- Evaluamos los DEGs que estamos usando en el modelo de EXPRESIÓN GENÉTICA COX PH:

```
DEGs_cox_comunes <- Reduce(intersect, list(result_final_degs$gen_name,rownames(degs_comunes_5grupos2)))

degs_comunes_5grupos2_cox <- degs_comunes_5grupos2 %>%
  filter(rownames(degs_comunes_5grupos2) %in% DEGs_cox_comunes)

degs_cox_comunes_5grupos2 <- result_final_degs %>%
  filter(gen_name %in% DEGs_cox_comunes)
nrow(degs_cox_comunes_5grupos2)
```

```
## [1] 4
```

Como vemos, 4 de los 7 DEGs que forman parte del modelo han aparecido en los 5 análisis. Si miramos grupo a grupo, observamos que en general 6 de los 7 DEGs que están en el modelo han ido apareciendo en cada uno de los análisis.

```
cox_degs_g1 <- result_final_degs %>%
  filter(gen_name %in% degs_g1$df_raw.gene_id)
nrow(cox_degs_g1)
```

```
## [1] 6
```

```
cox_degs_g2 <- result_final_degs  %>%
  filter(gen_name %in% degs_g2$df_raw.gene_id)
nrow(cox_degs_g2)
```

```
## [1] 6
```

```
cox_degs_g3 <- result_final_degs  %>%
  filter(gen_name %in% degs_g3$df_raw.gene_id)
nrow(cox_degs_g3)
```

```
## [1] 6
```

```
cox_degs_g4 <- result_final_degs  %>%
  filter(gen_name %in% degs_g4$df_raw.gene_id)
nrow(cox_degs_g4)
```

```
## [1] 5
```

```
cox_degs_g5 <- result_final_degs  %>%
  filter(gen_name %in% degs_g5$df_raw.gene_id)
nrow(cox_degs_g5)
```

```
## [1] 6
```

El ideal sería que todos los DEGs que forman parte del modelo hubiesen formado parte de este grupo estable de DEGs que es independiente de la partición realizada.

Conclusiones anexo 1

- En este anexo hemos estudiado la variabilidad en el cómputo de los DEGs en función de la partición de datos que se realice (usando k=5).
- El número de DEGs comunes obtenidos para los 5 procesos ha sido de 189, frente a los más de 300 que obtenemos cuando lo realizamos por separado.
- En el cálculo hemos verificado que 4 de los DEGs usados en nuestro modelo estaban dentro de los 189, y que en general 6 DEGs estaban presentes en cada uno de los cálculos por separado.
- Por la alta dependencia que existen entre las lecturas de los genes, es normal que no siempre se obtengan exactamente los mismos DEGs en función de la partición que se emplee. Lo que sí observamos es que los DEGs que están en nuestro modelo sí forman parte de los DEGs en la mayoría de las ocasiones.

Anexo 2. Otras metodologías para selección de los DEGs en modelo EXPRESIÓN GENÉTICA:

Dejamos los diferentes links usados para las dos metodologías de selección analizadas en este apartado:

- Link general:

http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/154-stepwise-regression-essentials-in-r/

- Links sobre función *selectCox* de librería riskRegression (apartado A2.1):

https://search.r-project.org/CRAN/refmans/riskRegression/html/selectCox.html

https://search.r-project.org/CRAN/refmans/rms/html/fastbw.html

- Link sobre función *stepAIC* de la librería MASS (apartado A2.2):

https://www2.karlin.mff.cuni.cz/~pesta/NMFM404/ph.html#Model_selection

A2.1. Función selectCox de la librería riskRegression:

- Con esta función de la librería *riskRegression* realizamos un **backward elimination** partiendo del conjunto completo de los 50 DEGs con mayores diferencias en la expresión genética.

```
library(riskRegression)
```

```
## riskRegression version 2022.11.28
```

- Lanzamos la función con los 50 genes identificados con más diferencia en la expresión genética:

```
res.selectCox <- selectCox(Surv(t, status) ~ ENSG00000167749.11 + ENSG00000046774.10 +ENSG00000010282.15
                  ENSG00000267774.2+ENSG00000166105.16+ENSG00000215117.6+ENSG00000223350.2+
                  ENSG00000077009.13+ENSG00000105141.6+ENSG00000181903.5+ENSG00000256292.2+
                  ENSG00000145192.13+ENSG00000268621.6+ENSG00000260542.1+ENSG00000234844.1+
                  ENSG00000268879.1+ENSG00000174562.13+ENSG00000133020.4+ENSG00000043355.12+
                  ENSG00000223786.1+ENSG00000157005.4+ENSG00000227555.1+ENSG00000235207.1+
                  ENSG00000118271.12+ENSG00000249306.6+ENSG00000133742.14+ENSG00000136110.13+
                  ENSG00000269729.1+ENSG00000171431.4+ENSG00000214978.7+ENSG00000255951.1+
                  ENSG00000227744.4+ENSG00000236366.3+ENSG00000189001.11+ENSG00000248227.1+
                  ENSG00000287583.1+ENSG00000248719.2+ENSG00000238194.2+ENSG00000139800.8+
                  ENSG00000169876.14+ENSG00000158874.11+ENSG00000176826.15+ENSG00000164363.10+
                  ENSG00000286016.1+ENSG00000250149.2+ENSG00000119919.11+ENSG00000155269.12+
                  ENSG00000249407.1+ENSG00000170044.8+ENSG00000155269.12+ENSG00000249407.1+
                  ENSG00000170044.8+ENSG00000102854.16, data =  matriz_cox, rule = "aic" )
```

- Veamos qué genes ha incluído en el modelo con esta selección backward:

```
res.selectCox$In
```

```
##  [1] "ENSG00000256292.2"  "ENSG00000145192.13" "ENSG00000268621.6"
##  [4] "ENSG00000234844.1"  "ENSG00000268879.1"  "ENSG00000174562.13"
##  [7] "ENSG00000235207.1"  "ENSG00000255951.1"  "ENSG00000227744.4"
## [10] "ENSG00000236366.3"  "ENSG00000189001.11" "ENSG00000248227.1"
## [13] "ENSG00000158874.11" "ENSG00000176826.15" "ENSG00000286016.1"
## [16] "ENSG00000155269.12" "ENSG00000249407.1"  "ENSG00000170044.8"
```

- Veamos el modelo completo:

```
res.selectCox$fit
```

```
## Cox Proportional Hazards Model
##
##  rms::cph(formula = newform, data = data, x = TRUE, y = TRUE,
##      surv = TRUE)
##
##                         Model Tests     Discrimination
##                                               Indexes
##  Obs       442    LR chi2     89.60     R2       0.188
##  Events    153    d.f.           18    R2(18,442)0.150
##  Center 0.3149    Pr(> chi2) 0.0000    R2(18,153)0.374
##                   Score chi2 218.92     Dxy      0.327
##                   Pr(> chi2) 0.0000
##
##                     Coef    S.E.   Wald Z Pr(>|Z|)
##  ENSG00000256292.2   0.0033 0.0018  1.78  0.0750
##  ENSG00000145192.13 -0.0007 0.0010 -0.69  0.4871
##  ENSG00000268621.6  -0.0028 0.0013 -2.07  0.0386
##  ENSG00000234844.1  -0.0277 0.0095 -2.91  0.0036
##  ENSG00000268879.1   0.0223 0.0099  2.26  0.0241
##  ENSG00000174562.13  0.0001 0.0002  0.65  0.5172
##  ENSG00000235207.1   0.0174 0.0054  3.19  0.0014
##  ENSG00000255951.1  -0.0145 0.0105 -1.38  0.1685
##  ENSG00000227744.4   0.0172 0.0135  1.28  0.2000
##  ENSG00000236366.3   0.0157 0.0054  2.88  0.0039
##  ENSG00000189001.11  0.0009 0.0002  4.23  <0.0001
##  ENSG00000248227.1   0.0383 0.0077  4.95  <0.0001
##  ENSG00000158874.11  0.0005 0.0004  1.20  0.2317
##  ENSG00000176826.15  0.0000 0.0001 -0.48  0.6281
##  ENSG00000286016.1   0.0123 0.0028  4.34  <0.0001
##  ENSG00000155269.12  0.0318 0.0072  4.42  <0.0001
##  ENSG00000249407.1  -0.0075 0.0066 -1.13  0.2567
##  ENSG00000170044.8  -0.0007 0.0001 -4.47  <0.0001
##
```

Como podemos ver, esta selección de variables incluyen algunos genes que no son estadísticamente significativos en el modelo final.

- Verifiquemos si nuetros 7 DEGs han sido seleccionados o no por este modelo:

```
result_temporal2$gen_name %in% res.selectCox$In
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Vemos que todos los genes seleccionados por nuestro modelo están en el listado de selección a partir de esta metodología.

- Veamos por último si este modelo con los 18 genes seleccionados mejora o no en el conjunto TEST la separación entre las curvas KM y cual es el valor del log-rank obtenido:

```
# Elegimos los DEGs para crear el nuevo modelo. Son en este caso 18 DEGs:
names.use_xxx <- c(res.selectCox$In,"t","status")
matriz_cox_xxx <- matriz_cox[, names.use_xxx]

# creación del modelo:
res.cox_DEGs1_xxx <- coxph(Surv(t, status) ~ ., data =  matriz_cox_xxx,iter.max=500 )

df_predict_testx <- data.frame(resultado=predict(res.cox_DEGs1_xxx,df_cox_model_test, type = "risk"))
df_predict_testx <- df_predict_testx %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))
rownames(df_predict_testx) <- df_cox_model_test$Row.names

table(df_predict_testx$risk)
```

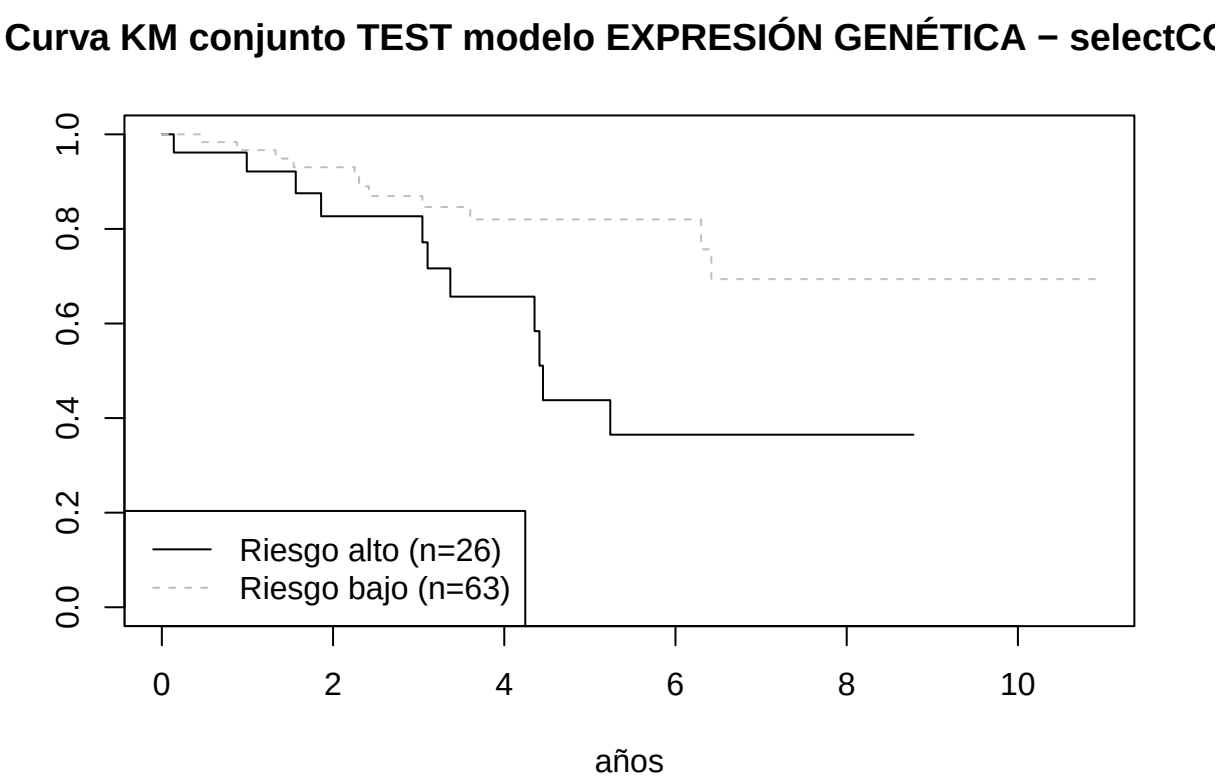
```
##
## alto bajo
##   26   63
```

```
# Curva KM:
union_test_gen <- merge(df_cox_model_test,df_predict_testx, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test, df_predict_testx, by = 0): column
## name 'Row.names' is duplicated in the result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=26)", "Riesgo bajo (n=63)"), lty = c("solid", "dashed"), col =
```




```
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.01682847
```

Como podemos observar, ni las curvas KM ni el log-rank obtenidos son mejores que el modelo seleccionado sólo con 7 DEGs, por lo que seguimos manteniendo este como nuestro modelo preferido. Lo que sí hemos podido verificar es que esta forma de seleccionar las variables del modelo incluyen a los 7 DEGs del nuestro.

A2.2. Función stepAIC de la librería MASS:

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

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

- Creamos el modelo y aplicamos la función stepAIC con el parámetro direction=both (es una metodología combinada entre backward y fordward):

```
fit_cox <- coxph(Surv(t, status) ~ ., data =  matriz_cox)

fitb = stepAIC(fit_cox, direction="both", k=2, trace = FALSE)
```

- Veamos el modelo que finalmente ha sido seleccionado:

```
fitb$formula
```

```
## Surv(t, status) ~ ENSG00000105141.6 + ENSG00000256292.2 + ENSG00000145192.13 +
##     ENSG00000268621.6 + ENSG00000234844.1 + ENSG00000268879.1 +
##     ENSG00000235207.1 + ENSG00000236366.3 + ENSG00000189001.11 +
##     ENSG00000248227.1 + ENSG00000139800.8 + ENSG00000158874.11 +
##     ENSG00000164363.10 + ENSG00000286016.1 + ENSG00000250149.2 +
##     ENSG00000155269.12 + ENSG00000249407.1 + ENSG00000170044.8
```

Traspasamos los DEGs seleccionados en el modelo final a un vector:

```
fitb_vector <- c("ENSG00000105141.6","ENSG00000256292.2","ENSG00000145192.13",
    "ENSG00000268621.6","ENSG00000234844.1","ENSG00000268879.1",
    "ENSG00000235207.1","ENSG00000236366.3","ENSG00000189001.11",
    "ENSG00000248227.1","ENSG00000139800.8","ENSG00000158874.11",
    "ENSG00000164363.10","ENSG00000286016.1","ENSG00000250149.2",
    "ENSG00000155269.12","ENSG00000249407.1","ENSG00000170044.8")
```

Como podemos ver, tenemos 18 DEGs seleccionados en este modelo, cuya selección es ligeramente diferente a los 18 del método previo.

- Como vemos hay 4 DEGs que están seleccionados por este modelo pero que no estaban en el del modelo del apartado A2

```
res.selectCox$In %in% fitb_vector
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE
## [13]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

- Verifiquemos si nuetros 7 DEGs han sido seleccionados o no por este modelo:

```
result_temporal2$gen_name %in% fitb_vector
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Lo que sí podemos comprobar es que los 7 DEGs seleccionados por nuestro modelo se encuentran también seleccionados a partir de la metodología empleada con esta función.

- Veamos por último si este modelo con los 18 genes seleccionados mejora o no en el conjunto TEST la separación entre las curvas KM y cual es el valor del log-rank obtenido:

```r
# Elegimos los DEGs para crear el nuevo modelo. Son en este caso 18 DEGs:
names.use_xxx <- c(fitb_vector,"t","status")
matriz_cox_xxx <- matriz_cox[, names.use_xxx]

# creación del modelo:
res.cox_DEGs1_xxx <- coxph(Surv(t, status) ~ ., data =  matriz_cox_xxx,iter.max=500 )

df_predict_testx <- data.frame(resultado=predict(res.cox_DEGs1_xxx,df_cox_model_test, type = "risk"))
df_predict_testx <- df_predict_testx %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))
rownames(df_predict_testx) <- df_cox_model_test$Row.names

table(df_predict_testx$risk)
```

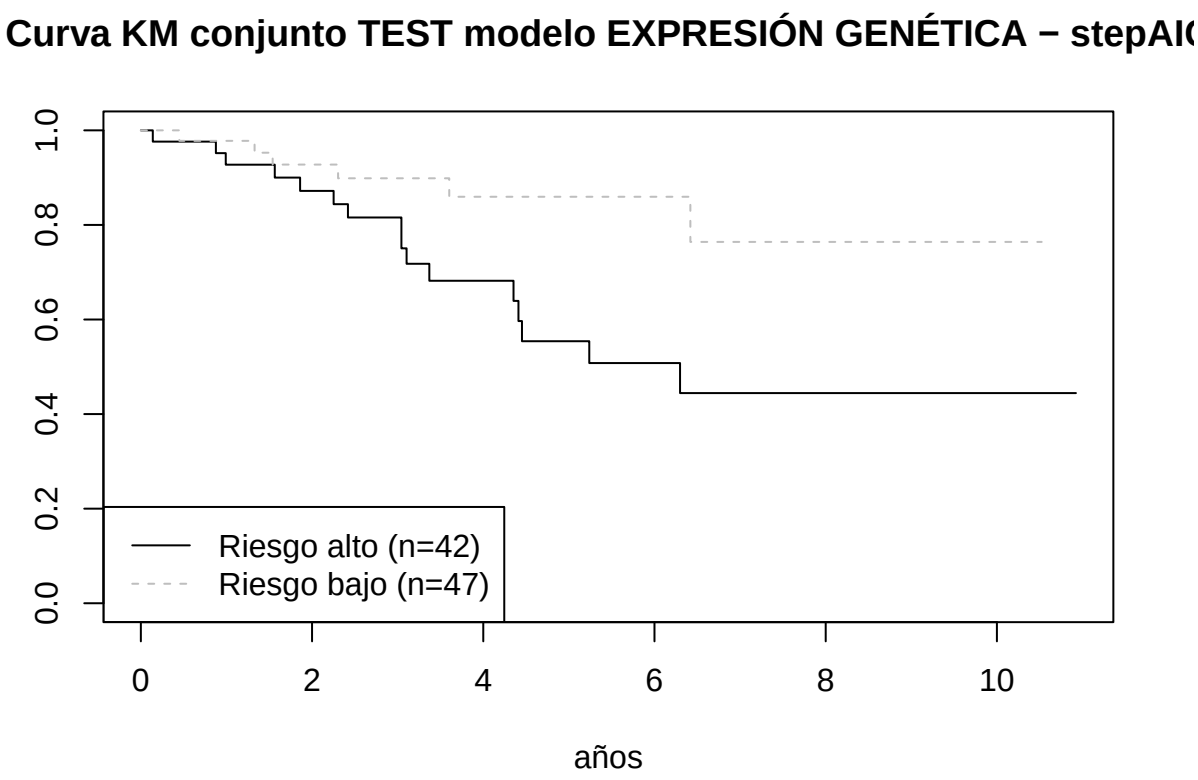
```
##
## alto bajo
##   42   47
```

```r
# Curva KM:
union_test_gen <- merge(df_cox_model_test,df_predict_testx, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test, df_predict_testx, by = 0): column
## name 'Row.names' is duplicated in the result
```

```r
km_fit <- survfit(Surv(t/365, status) ~ risk, data=union_test_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=42)", "Riesgo bajo (n=47)"), lty = c("solid", "dashed"), col =
```




```
logRank <- survdiff(Surv(t/365, status) ~ risk, data = union_test_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.0197829
```

Observamos, tal como ocurría con el modelo seleccionado en el apartado A2.1 que ni las curvas KM y el log-rank obtenidos son mejores que el modelo generado con los 7 DEGs identificados, por lo que tanto consideramos que dicho modelo mejora en predicción a este.

Un elemento importante es que los 7 DEGs seleccionados son identificados por las dos metodologías nuevas empleadas para seleccionar variables, lo que permite asegurar robustez al proceso de selección.

Anexo 3. Otras metodologías para selección de las variables en modelo COMBINADO (DEGs + clinical):

- Generamos el conjunto de entrenamiento del modelo COX con los 50 DEGs y las 4 variables clínicas seleccionadas:

```
anexo3_df_clinical_train <- clinical_train %>%
  dplyr::select(12:15)

a3_train <- merge(anexo3_df_clinical_train, matriz_cox, by=0, all=TRUE)
rownames(a3_train) <- a3_train$Row.names
a3_train <- a3_train %>%
  dplyr::select(-1)
```

- Aplicamos la función stepAIC para identificar qué variables elige:

```
a3_fit_cox <- coxph(Surv(t, status) ~ ., data =  a3_train)

a3_fitb = stepAIC(a3_fit_cox, direction="both", k=2, trace = FALSE)
```

- Veamos el resultado de las variables seleccionadas por la función stepAIC:

```
a3_fitb$formula
```

```
## Surv(t, status) ~ ajcc_pathologic_n_cat + age_at_index_cat +
##     ajcc_pathologic_stage_cat + ENSG00000010282.15 + ENSG00000145192.13 +
##     ENSG00000268621.6 + ENSG00000234844.1 + ENSG00000268879.1 +
##     ENSG00000235207.1 + ENSG00000255951.1 + ENSG00000227744.4 +
##     ENSG00000236366.3 + ENSG00000189001.11 + ENSG00000248227.1 +
##     ENSG00000158874.11 + ENSG00000164363.10 + ENSG00000286016.1 +
##     ENSG00000155269.12 + ENSG00000249407.1 + ENSG00000170044.8
```

- Como podemos observar se están seleccionando 3 variables clínicas y 17 DEGs: Las tres variables clínicas son el ajcc_ patológico_n relacionado con nódulos afectos, la edad y el estadio de la enfermedad. La variable clínica no seleccionada es el tamaño del tumor, algo idéntico a lo obtenido sin usar la función stepAIC.

Vamos primero a verificar si los 7 DEGs de nuestro modelo se encuentran entre los seleccionados por esta metodología:

- Traspasamos el resultado a un vector y verificamos si nuestros 7 DEGs están entre ellos:

```
a3_fitb_vector <- c("ENSG00000010282.15","ENSG00000145192.13","ENSG00000268621.6",
    "ENSG00000234844.1","ENSG00000268879.1","ENSG00000235207.1",
    "ENSG00000255951.1","ENSG00000227744.4","ENSG00000236366.3",
    "ENSG00000189001.11","ENSG00000248227.1", "ENSG00000158874.11",
    "ENSG00000164363.10","ENSG00000286016.1", "ENSG00000155269.12",
    "ENSG00000249407.1","ENSG00000170044.8")

result_temporal2$gen_name %in% a3_fitb_vector
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Como podemos ver los 7 DEGs seleccionados en nuestro modelo final se encuentran también en este caso.

- Veamos por último si sobre el conjunto TEST, tenemos una mejor separación de las curvas KM y valor log-rank o no:

```
a3_z_predict_clinical_gen_test <- data.frame(resultado=predict(a3_fit_cox,df_cox_model_test, type = "ris

a3_z_predict_clinical_gen_test <- a3_z_predict_clinical_gen_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))

rownames(a3_z_predict_clinical_gen_test) <- df_cox_model_test$Row.names

table(a3_z_predict_clinical_gen_test$risk)
```

```
##
## alto bajo
##   64   25
```

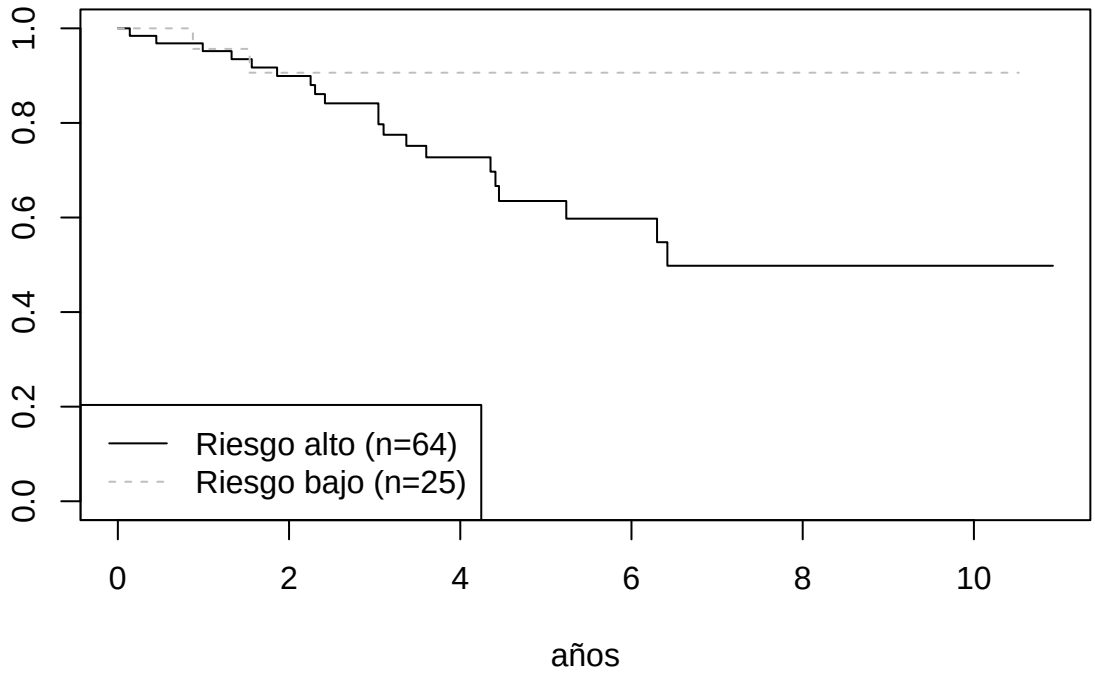
```
# Curva KM:

a3_union_test_clinical_gen <- merge(df_cox_model_test,a3_z_predict_clinical_gen_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test, a3_z_predict_clinical_gen_test, :
## column name 'Row.names' is duplicated in the result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=a3_union_test_clinical_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=64)", "Riesgo bajo (n=25)"), lty = c("solid", "dashed"), col =
```




```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = a3_union_test_clinical_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.06191993
```

Anexo 4. Validación de si existe mejora en el modelo COMBINADO con stepAIC:

- Aplicamos la función stepAIC para identificar qué variables elige sobre el modelo COMBINADO

```
a4_fit_cox <- coxph(Surv(t, status) ~ ., data =  z_clinico_gen_cox_train2)

a4_fitb = stepAIC(a4_fit_cox, direction="both", k=2, trace = FALSE)
```

- Veamos el resultado:

```
a4_fitb$formula
```

```
## Surv(t, status) ~ ajcc_pathologic_n_cat + age_at_index_cat +
##     ajcc_pathologic_stage_cat + ENSG00000234844.1 + ENSG00000235207.1 +
##     ENSG00000189001.11 + ENSG00000248227.1 + ENSG00000286016.1 +
##     ENSG00000155269.12 + ENSG00000170044.8
```

El resultado nos muestra que selecciona los 7 DEGs y 3 variables clínicas, eliminando el tamaño del tumor.

- Si mostramos la gráfica sobre este modelo:

```
a4_z_predict_clinical_gen_test <- data.frame(resultado=predict(a4_fit_cox,df_cox_model_test, type = "ris

a4_z_predict_clinical_gen_test <- a4_z_predict_clinical_gen_test %>%
  mutate(risk=case_when(
    resultado <= 1 ~ "bajo",
    resultado > 1 ~ "alto"
  ))

rownames(a4_z_predict_clinical_gen_test) <- df_cox_model_test$Row.names

table(a4_z_predict_clinical_gen_test$risk)
```

```
##
## alto bajo
##   54   35
```

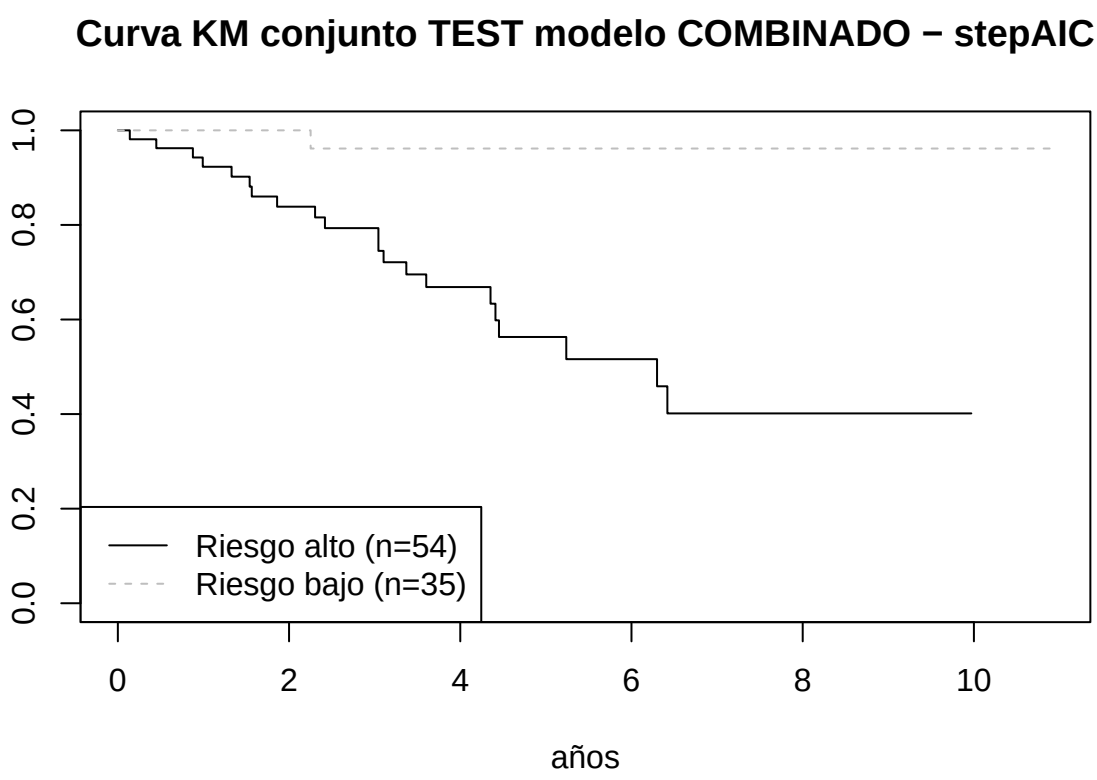
```
# Curva KM:

a4_union_test_clinical_gen <- merge(df_cox_model_test,a4_z_predict_clinical_gen_test, by=0)
```

```
## Warning in merge.data.frame(df_cox_model_test, a4_z_predict_clinical_gen_test, :
## column name 'Row.names' is duplicated in the result
```

```
km_fit <- survfit(Surv(t/365, status) ~ risk, data=a4_union_test_clinical_gen)

# summary(km_fit)
 plot(km_fit, xlab="años", lty = c("solid", "dashed"), col = c("black", "grey"),main = "Curva KM conjun
 legend("bottomleft", c("Riesgo alto (n=54)", "Riesgo bajo (n=35)"), lty = c("solid", "dashed"), col =
```



- Evaluamos si la diferencia entre ambas curvas es estadísticamente significativa:

```
# método log-rank:
logRank <- survdiff(Surv(t/365, status) ~ risk, data = a4_union_test_clinical_gen)
pval <- p.val <- 1 - pchisq(logRank$chisq, length(logRank$n) - 1)
pval
```

```
## [1] 0.0005193294
```

Anexo 5. Ubicación en github

Este documento queda alojado en: https://github.com/alejandrofp-hub/TFM-Material-complementario/tree/main