

PEC 2

Análisis de datos ómicos

Eduardo A. Sánchez Torres

14 de junio de 2020

Índice

1. Resumen	2
2. Datos para el análisis	3
2.1. Estudio	3
2.2. Tipo de datos de partida	3
2.3. Selección aleatoria de muestras	3
3. Objetivos	6
4. Proceso de análisis	7
4.1. Preparación de los datos	7
4.2. Preprocesado de los datos	7
4.2.1. Filtraje	7
4.2.2. Análisis exploratorio y visualización	7
4.3. Identificación de genes diferencialmente expresados	9
4.4. Anotación de resultados	13
4.5. Búsqueda de patrones de expresión y agrupación de muestras	13
4.6. Análisis de significación biológica	13
5. Resultados y discusión	15
5.1. Contraste SFI-NIT	15
5.2. Contraste ELI-NIT	17
5.3. Contraste ELI-SFI	17
5.4. Comparativa entre contrastes	17
5.5. Análisis de significación biológica	18
6. Repositorio GitHub y código	19
Referencias	25

1. Resumen

2. Datos para el análisis

2.1. Estudio

El conjunto de datos utilizado en este trabajo pertenece a un estudio de expresión RNA-seq, obtenido del repositorio GTEx, en el cual se realiza un análisis de tiroides en el que se compara tres tipos de infiltración medidos en un total de 292 muestras pertenecientes a tres grupos:

- *Not infiltrated tissues* (NIT): 236 muestras.
- *Small focal infiltrates* (SFI): 42 muestras.
- *Extensive lymphoid infiltrates* (ELI): 14 muestras.

Para el análisis realizado aquí, se han seleccionado aleatoriamente 10 muestras de cada grupo. De esta manera, el diseño experimental presenta un único factor *Grupo* con tres niveles, NIT, SFI y ELI, disponiendo de 10 réplicas en cada nivel, constituyendo así un total de 30 muestras.

2.2. Tipo de datos de partida

Los datos crudos de partida se basan en una matriz de conteo (*count matrix*) y una tabla asociada de información sobre las muestras (*targets table*). En la matriz de conteo, cada fila representa un gen Ensembl, cada columna una biblioteca de ARN secuenciada (library) correspondiente a cada muestra, y los valores de cada celda son los números brutos de fragmentos que se asignaron de forma única al gen respectivo en cada biblioteca. Por otro lado, la tabla de información de las muestras, presenta tantas filas como muestras, y tantas columnas como items se quieran asociar con cada muestra [1].

Así se ha partido de una matriz de conteo con 293 columnas (292 muestras + IDs Ensembl) y 56202 filas (genes Ensembl). Y una tabla de información de muestras con 292 filas (muestras) y 9 columnas (items), siendo la columna “Group” el factor de este diseño experimental.

2.3. Selección aleatoria de muestras

Para la selección aleatoria de las 10 muestras de cada grupo se ha diseñado el siguiente algoritmo.

En primer lugar se ha importado la matriz de conteo counts:

```
library(readr)
counts = read_delim("datos/counts.csv",
                     ";", escape_double = FALSE, trim_ws = TRUE)
```

Con la finalidad de evitar problemas en el proceso de anotación, a cada ID de Ensembl, se le debe retirar el punto y el número que le sigue, que se corresponden con la versión de Ensembl. De este modo se ha utilizado la función gsub():

```
counts$X1 = gsub("\\..*", "", counts$X1)
```

Posteriormente, se ha importado la tabla de información de las muestras targets:

```
targets = read_csv("datos/targets.csv")
```

Importados los data frames counts y targets se ha aplicado el siguiente script para la selección aleatoria de 10 muestras de cada grupo (30 en total):

```
# Establece una semilla para asegurar la repetitividad aleatoria.
set.seed(73026030)
# Crea un data frame vacío que contendrá los nombres de las 30 muestras.
df = data.frame()
# Almacena en groups los diferentes grupos de la columna Group del
# data frame targets.
groups = unique(targets[, "Group"])
# Bucle for que se ejecuta tantas veces como grupos haya en groups.
for (i in 1:nrow(groups)) {
  # Crea un data frame con únicamente las filas correspondientes al
  # grupo i.
  targets_group = subset(targets, Group == as.character(groups[i,1]))
  # Cuenta las filas del data frame targets_group, es decir, el número
  # de muestras del grupo i.
  rows = nrow(targets_group)
  # Selecciona aleatoriamente 10 índices de fila sin repetición entre
  # todas las filas de targets_group del grupo i.
  indexes = sample(1:rows, 10, replace = F)
  # Crea el data frame targets_samples que contiene los nombres de las
  # muestras correspondientes a los 10 índices aleatorios seleccionados.
  targets_samples = targets_group[indexes, "Sample_Name"]
  # En cada iteración del bucle for, almacena en df los nombres de las 10
  # muestras de cada grupo que se han seleccionado aleatoriamente.
  df = rbind(df, targets_samples)
}
# Crea el vector de caracteres df a partir del data frame df con los
# nombres de las 30 muestras seleccionadas.
df = df$Sample_Name
# Crea la matriz de conteo de las 30 muestras, indexando las columnas de
# counts con el vector df.
counts_set = counts[,df]
# Ordena alfanuméricamente las columnas de la matriz de conteo counts_set.
counts_set = counts_set[,order(colnames(counts_set))]
# Establece que los nombres de fila de counts_set sean los ID de Ensembl
# de counts.
row.names(counts_set) = counts$X1
# Crea la tabla de información de las muestras targets_set con únicamente
```

```
# las filas correspondientes a las muestras seleccionadas en counts_set.
targets_set = subset(targets, (targets$Sample_Name %in% df))
# Ordena alfanuméricamente las filas de targets_set en función de los
# nombres de la columna Sample_Name para que éstas se correspondan con
# el orden de las columnas de counts_set.
targets_set = targets_set[order(targets_set$Sample_Name),]
```

De esta manera se generó la matriz de conteo `counts_set` (56202 x 30) y la tabla de información de las muestras `targets_set` (30 x 9) utilizadas para desarrollar los objetivos de este trabajo.

Nótese, que es crítico y fácil de pasar por alto, que el orden de las columnas de la matriz de conteo tiene que ser el mismo que el orden de las filas de la tabla de información, para que la asociación de `counts_set` y `targets_set` sea correcta [2].

3. Objetivos

El objetivo general de este trabajo es identificar qué genes están expresados diferencialmente así como su significación biológica, a partir de datos de un estudio de tiroides RNA-seq que comparaba tres tipos de infiltración: *Not infiltrated tissues* (NIT), *Small focal infiltrates* (SFI) y *Extensive lymphoid infiltrates* (ELI).

Para la consecución del objetivo general se han planteado los siguientes objetivos específicos:

- Determinar qué genes están diferencialmente expresados en SFI y ELI frente al grupo control NIT.
- Realizar la comparación SFI-ELI para identificar su expresión diferencial.
- Comparar los tres contrastes (SFI-NIT, ELI-NIT y SFI-ELI) para observar patrones de expresión y determinar si existen genes expresados diferencialmente que sean comunes entre comparaciones.
- Aplicar un análisis de enriquecimiento a los genes expresados diferencialmente en cada contraste.

4. Proceso de análisis

El proceso de análisis de datos RNA-seq se ha llevado a cabo utilizando el lenguaje R versión 4.0.1 a través de los paquetes de Bioconductor versión 3.11. Concretamente se ha utilizado el paquete DESeq2. En este apartado se detallan cada uno de los pasos seguidos en dicho análisis.

4.1. Preparación de los datos

A partir de la matriz de conteo `counts_set` y la tabla de información de las muestras `targets_set`, se ha generado con la función `DESeqDataSetFromMatrix` el correspondiente `DESeqDataSet`, con el que comenzar el análisis.

4.2. Preprocesado de los datos

4.2.1. Filtraje

La matriz de conteo del objeto `DESeqDataSet` puede contener muchas filas con solo ceros o muy pocos fragmentos en total, que poca información aportan en el análisis de expresión diferencial. Para reducir el tamaño del objeto y aumentar la velocidad de las funciones que se apliquen al mismo, se han eliminado las filas que no tienen recuentos, o solo un recuento único en todas las muestras.

De este modo se ha pasado de 56202 a 43429 filas (genes Ensembl).

4.2.2. Análisis exploratorio y visualización

Con la finalidad de evaluar la similitud de las muestras entre grupos se han llevado a cabo los siguientes análisis:

- Distancias euclídeas.
- Componentes principales (PCA).
- Escalado multidimensional (MDS).

Previamente a realizar dichos análisis, se han transformado los conteos brutos con la función `vst()` (*variance stabilizing transformation*). Estabilizar la varianza para que los datos cumplan con el principio de homocedasticidad es recomendable antes de aplicar éstos análisis multidimensionales.

En la Figura 1 podemos ver un heatmap en el cual se han representado las distancias euclidianas entre muestras. Se observa la distinción entre dos grupos, uno constituido por 6 muestras ELI y otro que engloba al resto.

En la Figura 2 observamos la representación de las dos primeras componentes principales, que explican el 75 % de la varianza. Se puede ver la distinción en dos grupos como en el caso anterior, diferenciándose el grupo ELI del resto.

Por último, en la Figura 3 podemos observar los resultados del análisis MDS. En este caso, la distinción entre grupos no es tan marcada.

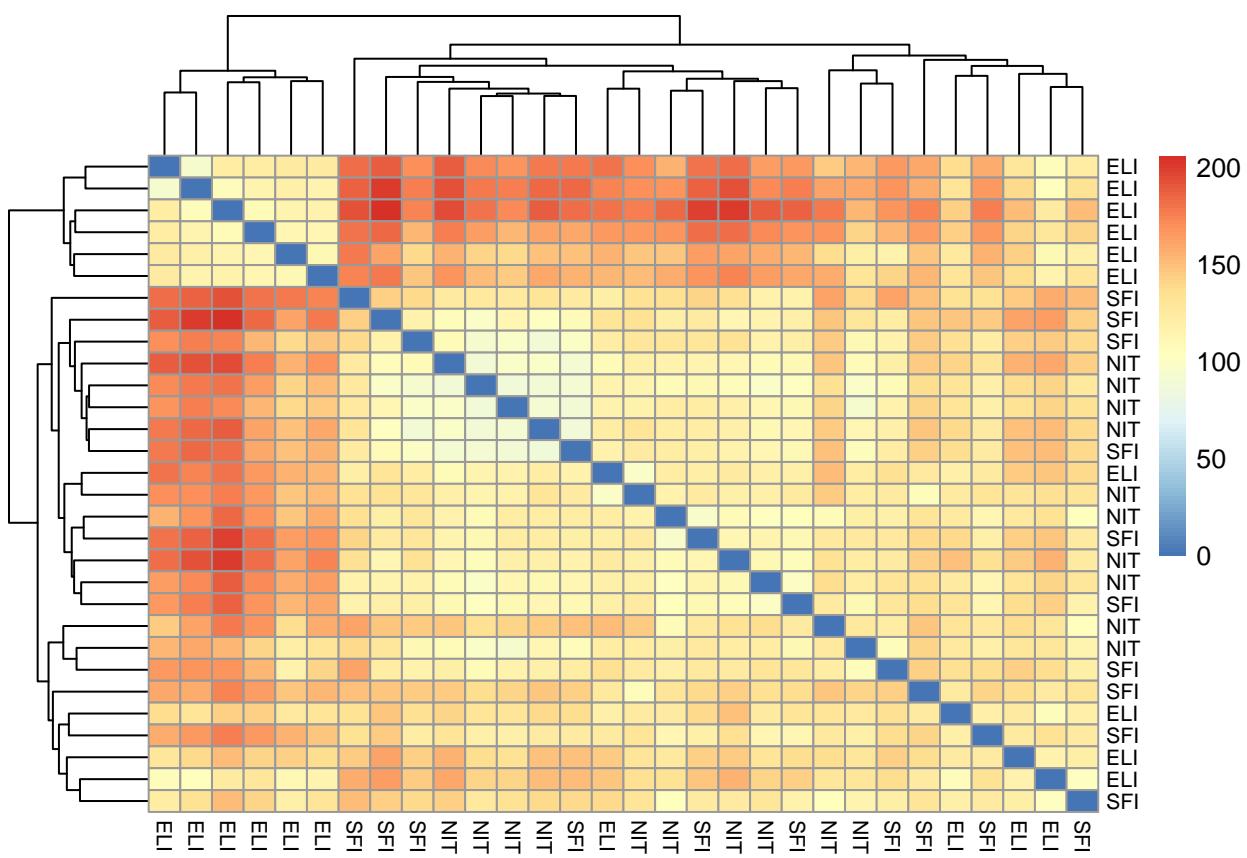


Figura 1: Diagrama heatmap de representación de las distancias euclidianas entre muestras.

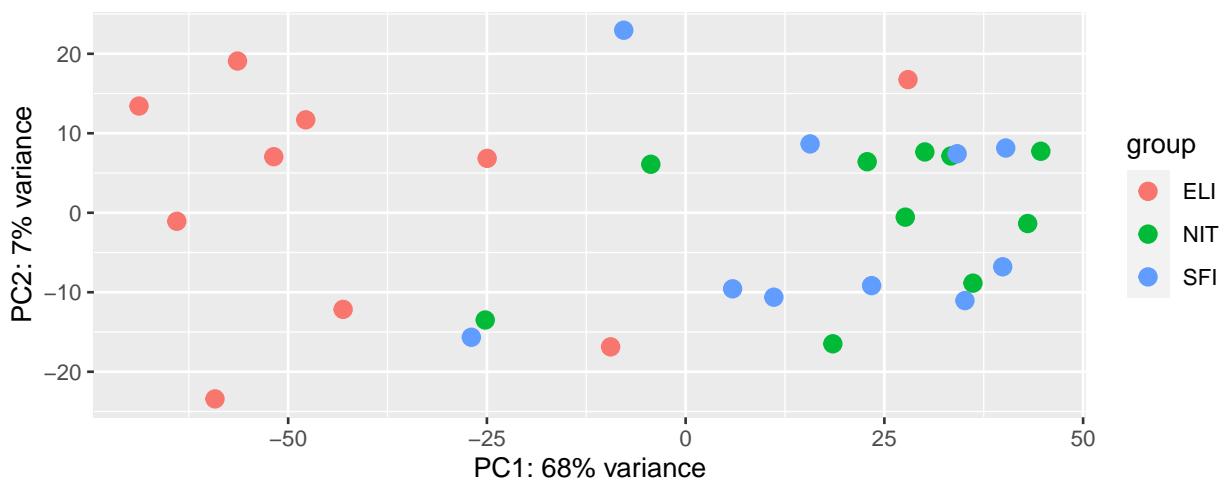


Figura 2: Representación de las dos primeras componentes principales.

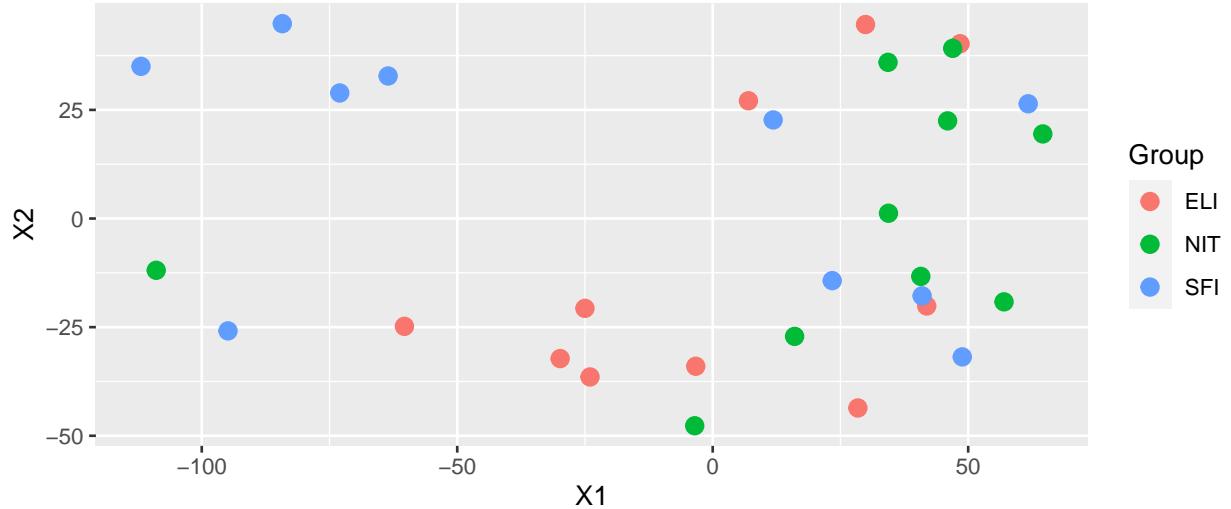


Figura 3: Gráfico resultante del escalamiento multidimensional (MDS).

4.3. Identificación de genes diferencialmente expresados

A partir del DESeqDataSet creado con los conteos brutos se ha ejecutado directamente la función `DESeq()` que ejecuta en una sola llamada el pipeline de análisis de expresión diferencial.

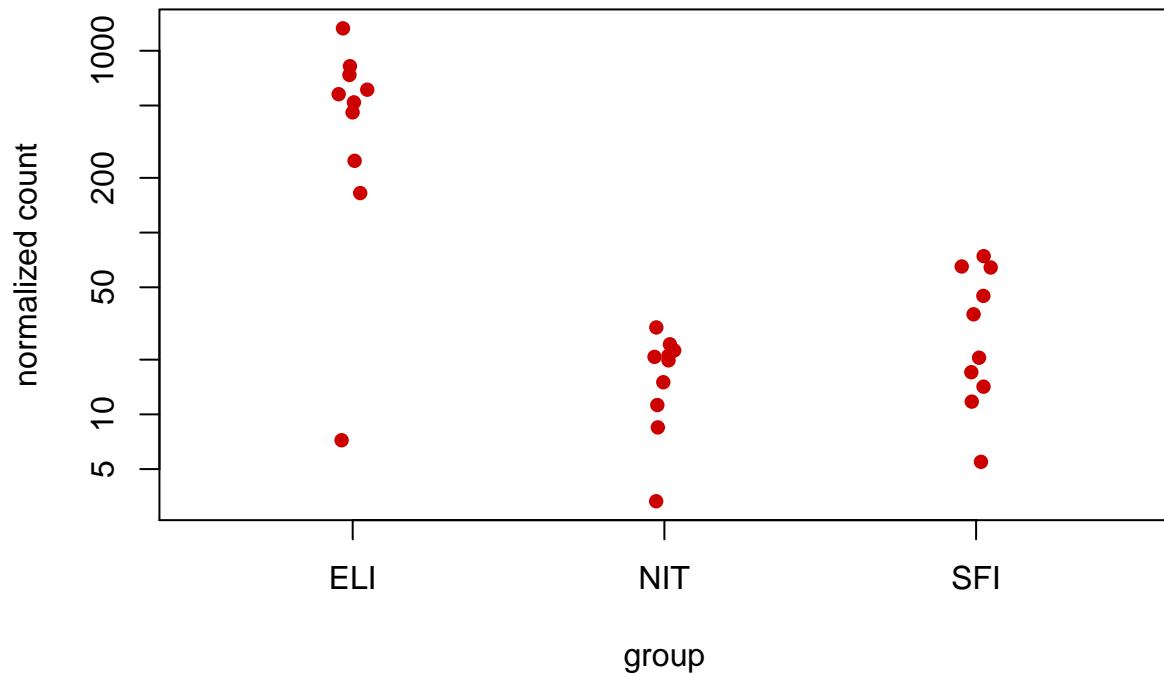
Para extraer los resultados de cada contraste planteado (SFI-NIT, ELI-NIT y SFI-ELI) se ha utilizado la función `results()`. Para controlar el porcentaje de falsos positivos que pueden resultar del alto número de contrastes realizados simultáneamente entre genes, los p-valores se han ajustado por el método de Benjamini y Hochberg (BH) [3].

La distinción entre genes *up* o *down* regulados se ha hecho con la función `summary()` aplicada al `DESeqResults` de cada contraste.

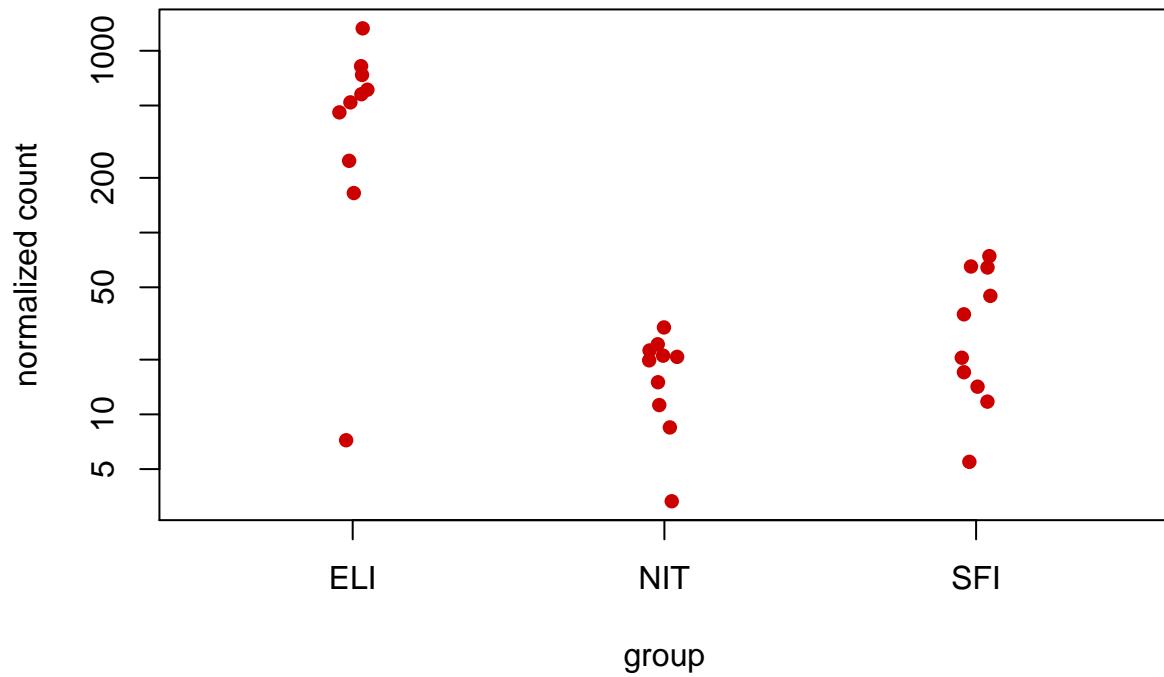
Por último, la visualización global de la expresión diferencial en cada contraste se ha llevado a cabo mediante gráficos tipo `MAnnot`. Para la visualización particular de los conteos para gen más significativo en cada contraste se ha recurrido a gráficos tipo `plotCounts`.

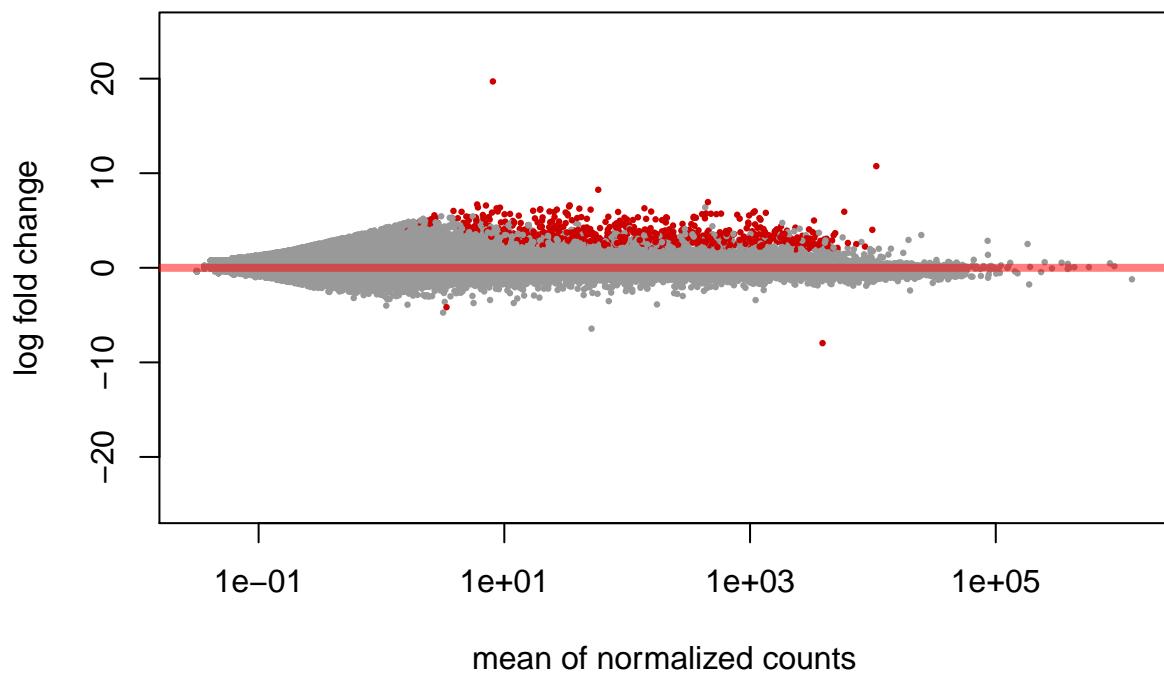
Los genes que se han considerado significativamente diferencialmente expresados en cada contraste son aquellos que presentan un fold change (FC) ≥ 2 (`1fcThreshold = 1`) y un p-valor < 0.05 . Para transformar el ratio `log2FoldChange` a valores de fold change, se ha empleado la función `logratio2foldchange` del paquete `gtools`.

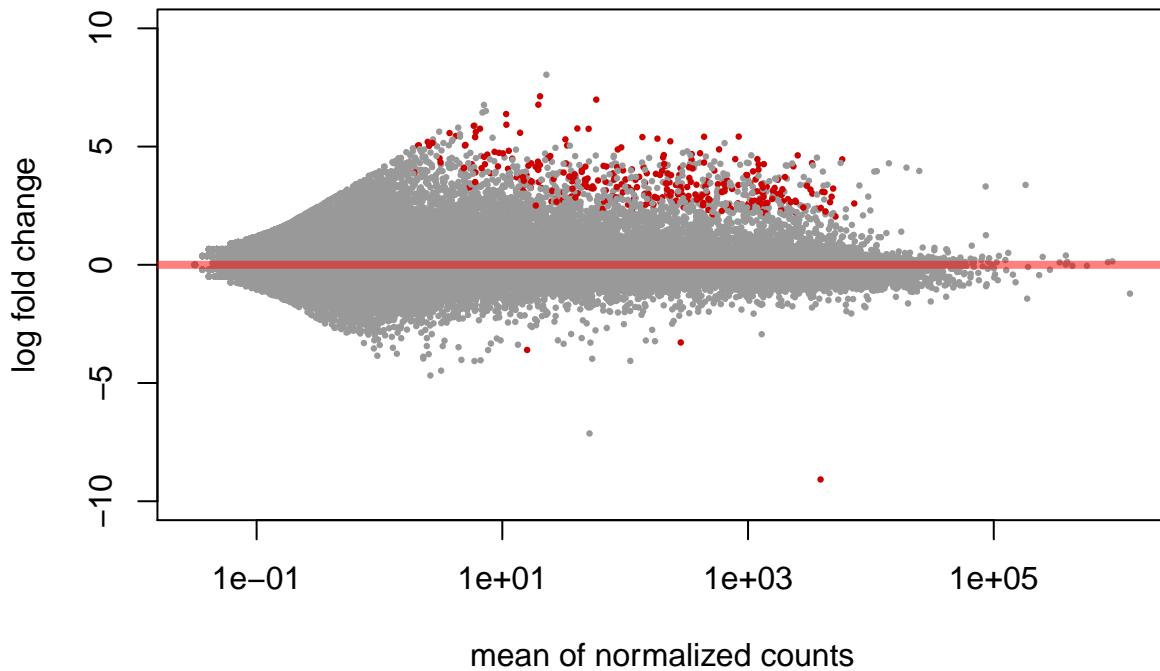
ENSG00000265206



ENSG00000265206







4.4. Anotación de resultados

Para establecer la correspondencia entre los identificadores de Ensembl y otro tipo de identificadores del gen al que pertenecen, se ha utilizado el paquete de anotaciones `EnsDb.Hsapiens.v86`, con la función `mapIds()`. Así se han asociado con los siguientes identificadores:

- *Gene Symbol*: Símbolo del gen.
- *Entrez gene Id*: Identificador del gen en la base de datos Entrez.

4.5. Búsqueda de patrones de expresión y agrupación de muestras

Con el fin de conocer qué genes cambian simultáneamente entre comparaciones se han utilizado diagramas de Venn (`vennDiagram`), sin diferenciar entre genes *up* o *down* regulados.

Para la búsqueda de patrones de expresión y agrupación de muestras en los 20 genes que más varían entre muestras se ha construido un heatmap con la función `pheatmap()`.

4.6. Análisis de significación biológica

Por último, para el análisis de significación biológica se ha llevado a cabo un análisis de enriquecimiento bajo las categorías de Gene Ontology (funciones moleculares, procesos

biológicos o componentes celulares) con el paquete `clusterProfiler` utilizando la función `enrichGO`. Como objeto `OrgDb` se ha utilizado `org.Mm.eg.db`. De esta manera, se ha podido determinar si una determinada categoría GO aparece significativamente más a menudo entre los genes diferencialmente expresados en cada contraste, con un p-valor < 0.15 y sin fold change (FC) mínimo.

5. Resultados y discusión

5.1. Contraste SFI-NIT

La comparación del grupo SFI con el grupo control NIT muestra que únicamente un gen *up* regulado está diferencialmente expresado (Tabla). El perfil de expresión diferencial se puede observar de manera global en la Figura 4. De manera particular observamos los conteos de dicho gen en la Figura 5.

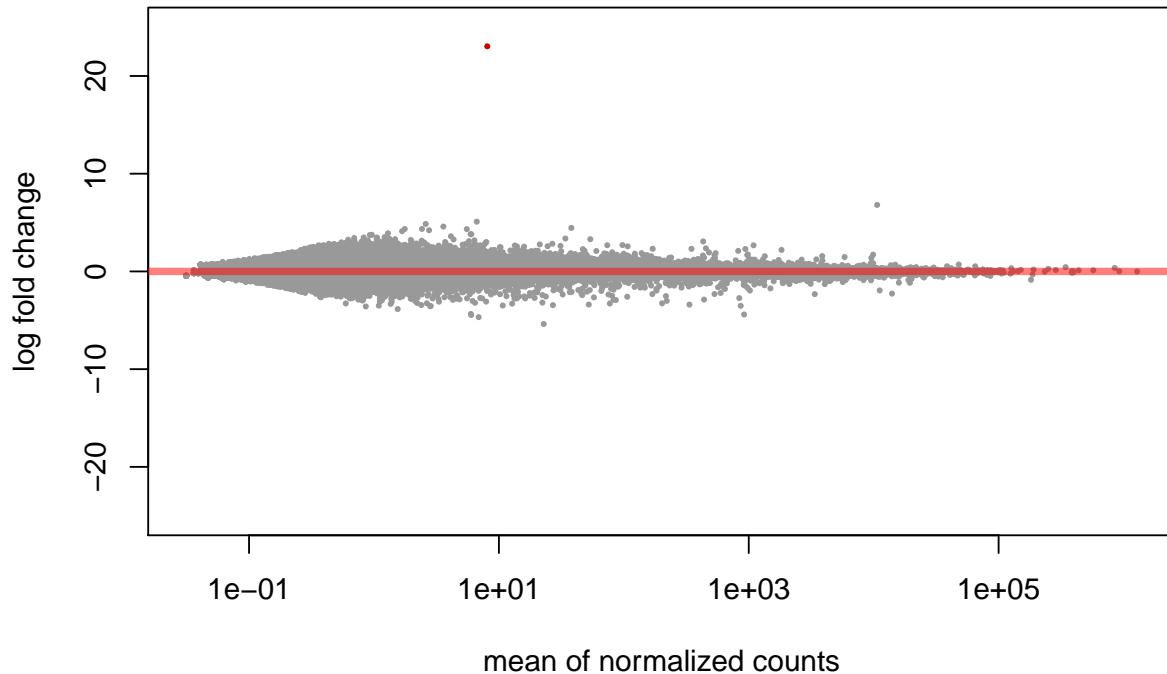


Figura 4: MA plot con los genes diferencialmente expresados en el contraste SFI-NIT.

Tabla 1: Genes diferencialmente expresados en el contraste SFI-NIT.

Ensembl Id.	Gene Symbol	Entrez Gene Id.	log2(FC)
ENSG00000152266	PTH	5741	23.03

ENSG00000152266

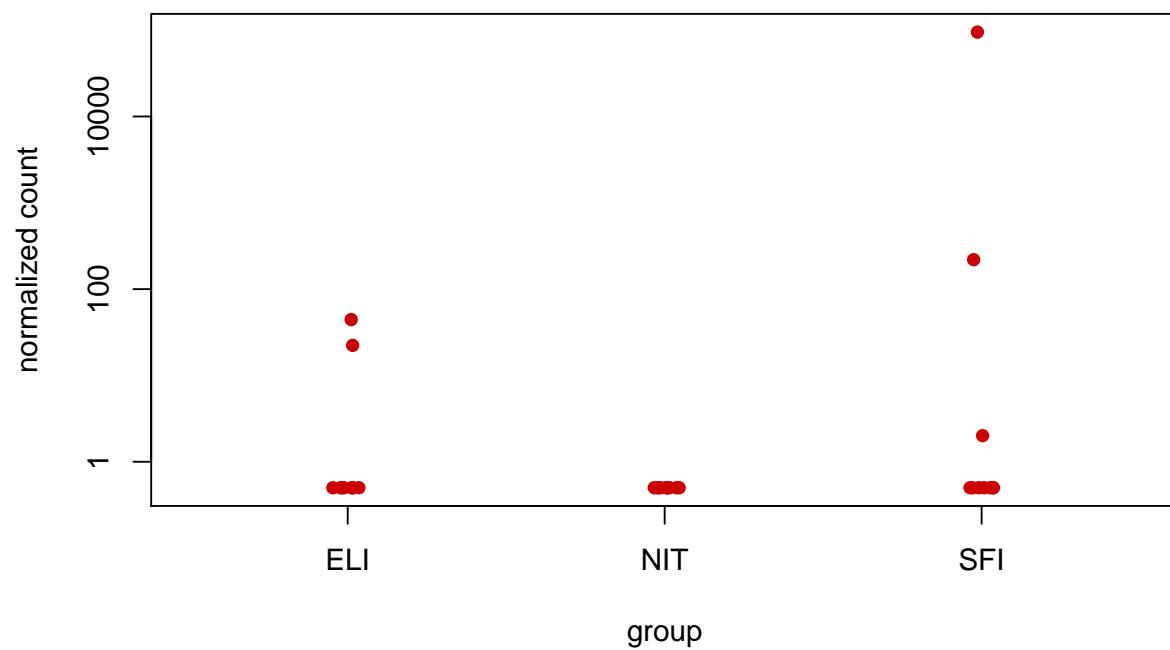
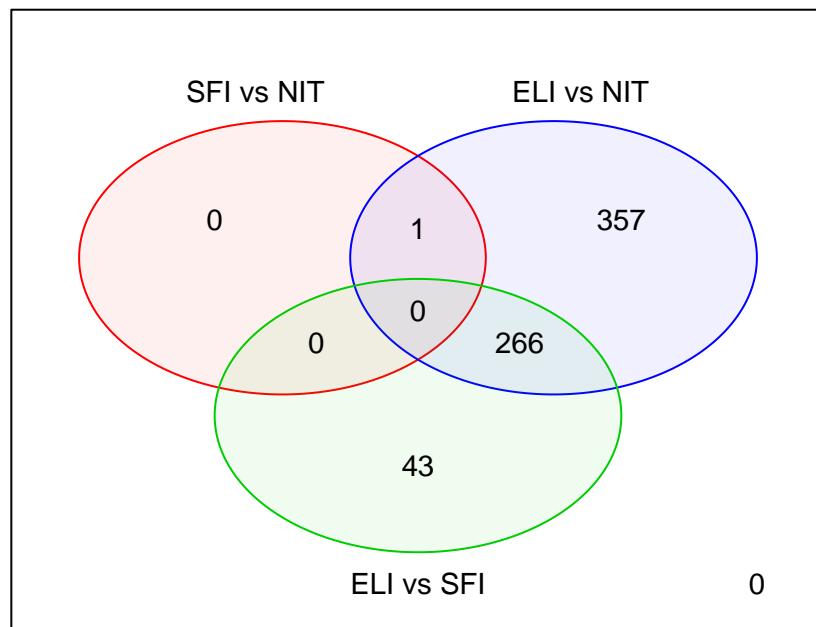


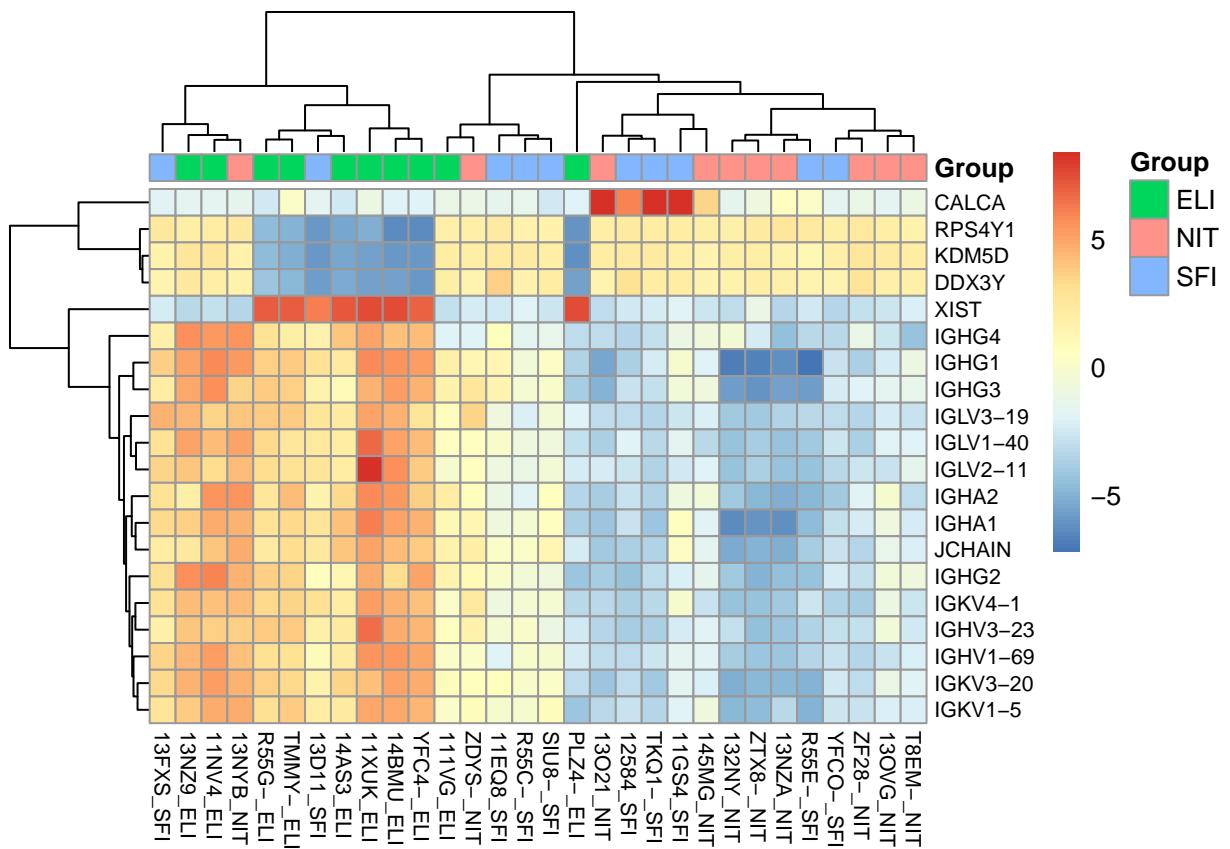
Figura 5: Count plot con el gen más significativo diferencialmente expresado en el contraste SFI-NIT.

5.2. Contraste ELI-NIT

5.3. Contraste ELI-SFI

5.4. Comparativa entre contrastes





5.5. Análisis de significación biológica

6. Repositorio GitHub y código

En el siguiente enlace del repositorio del proyecto en GitHub, se encuentran disponibles los datos, los resultados y todos los archivos derivados del proyecto en R:

https://github.com/edsantor/PEC_2

Por otro lado, a continuación se muestra el código utilizado en el proceso de análisis:

```
# Preparación de los datos

library(DESeq2)
dds = DESeqDataSetFromMatrix(countData = counts_set, colData = targets_set,
                             design = ~ Group)

# Filtado

nrow(dds)
dds = dds[rowSums(counts(dds)) > 1,]
nrow(dds)

# Análisis exploratorio y visualización

vsd = vst(dds, blind = FALSE)

assay_vsd = t(assay(vsd))
rownames(assay_vsd) = vsd$Group
colnames(assay_vsd) = NULL
assay_vsd = assay_vsd[order(row.names(assay_vsd)),]
sampleDists = dist(assay_vsd)
library(pheatmap)
sampleDistMatrix = as.matrix(sampleDists)
pheatmap(sampleDistMatrix,
         clustering_distance_rows = sampleDists,
         clustering_distance_cols = sampleDists,
         fontsize_row = 8, fontsize_col = 8,
         cluster_rows = T, cluster_cols = T)

plotPCA(vsd, intgroup = "Group")

library(ggplot2)
mdsData = data.frame(cmdscale(sampleDistMatrix))
mds = cbind(mdsData, as.data.frame(colData(vsd)))
ggplot(mds, aes(X1,X2,color=Group)) + geom_point(size=3)
```

```

# Identificación de genes expresados diferencialmente

dds = DESeq(dds, parallel =TRUE)

res_SFI_NIT = results(dds, contrast = c("Group", "SFI", "NIT"),
                      alpha = 0.05, lfcThreshold = log2(2))
res_ELI_NIT = results(dds, contrast = c("Group", "ELI", "NIT"),
                      alpha = 0.05, lfcThreshold = log2(2))
res_ELI_SFI = results(dds, contrast = c("Group", "ELI", "SFI"),
                      alpha = 0.05, lfcThreshold = log2(2))

summary(res_SFI_NIT)

summary(res_ELI_NIT)

summary(res_ELI_SFI)

topGene_SFI_NIT = rownames(res_SFI_NIT)[which.min(res_SFI_NIT$padj)]
plotCounts(dds, topGene_SFI_NIT, "Group", col = "red3", pch = 16)

topGene_ELI_NIT = rownames(res_ELI_NIT)[which.min(res_ELI_NIT$padj)]
plotCounts(dds, topGene_ELI_NIT, "Group", col = "red3", pch = 16)

topGene_ELI_SFI = rownames(res_ELI_SFI)[which.min(res_ELI_SFI$padj)]
plotCounts(dds, topGene_ELI_SFI, "Group", col = "red3", pch = 16)

plotMA(res_SFI_NIT, ylim=c(25,25), colSig = "red3",
       colLine = rgb(1,0,0,.5))

plotMA(res_ELI_NIT, ylim=c(25,25), colSig = "red3",
       colLine = rgb(1,0,0,.5))

plotMA(res_ELI_SFI, ylim=c(10,10), colSig = "red3",
       colLine = rgb(1,0,0,.5))

# Anotación de resultados

library("AnnotationDbi")
library(EnsDb.Hsapiens.v86)

res_SFI_NIT$symbol = mapIds(EnsDb.Hsapiens.v86,
                           keys=row.names(res_SFI_NIT),
                           column="SYMBOL",
                           keytype="GENEID",

```

```

            multiVals="first")
res_SFI_NIT$entrezid = mapIds(EnsDb.Hsapiens.v86,
                               keys=row.names(res_SFI_NIT),
                               column="ENTREZID",
                               keytype="GENEID",
                               multiVals="first")

res_ELI_NIT$symbol = mapIds(EnsDb.Hsapiens.v86,
                           keys=row.names(res_ELI_NIT),
                           column="SYMBOL",
                           keytype="GENEID",
                           multiVals="first")
res_ELI_NIT$entrezid = mapIds(EnsDb.Hsapiens.v86,
                               keys=row.names(res_ELI_NIT),
                               column="ENTREZID",
                               keytype="GENEID",
                               multiVals="first")

res_ELI_SFI$symbol = mapIds(EnsDb.Hsapiens.v86,
                           keys=row.names(res_ELI_SFI),
                           column="SYMBOL",
                           keytype="GENEID",
                           multiVals="first")
res_ELI_SFI$entrezid = mapIds(EnsDb.Hsapiens.v86,
                               keys=row.names(res_ELI_SFI),
                               column="ENTREZID",
                               keytype="GENEID",
                               multiVals="first")

res_SFI_NIT_df = as.data.frame(res_SFI_NIT)
res_SFI_NIT_df = subset(res_SFI_NIT_df, padj < 0.05
                        & abs(log2FoldChange) > 1)
res_SFI_NIT_df = res_SFI_NIT_df[order(res_SFI_NIT_df$padj),]

res_ELI_NIT_df = as.data.frame(res_ELI_NIT)
res_ELI_NIT_df = subset(res_ELI_NIT_df, padj < 0.05
                        & abs(log2FoldChange) > 1)
res_ELI_NIT_df = res_ELI_NIT_df[order(res_ELI_NIT_df$padj),]

res_ELI_SFI_df = as.data.frame(res_ELI_SFI)
res_ELI_SFI_df = subset(res_ELI_SFI_df, padj < 0.05
                        & abs(log2FoldChange) > 1)
res_ELI_SFI_df = res_ELI_SFI_df[order(res_ELI_SFI_df$padj),]

```

```

write.csv(res_SFI_NIT_df, file = "resultados/res_SFI_NIT_df.csv",
          row.names = T)

write.csv(res_ELI_NIT_df, file = "resultados/res_ELI_NIT_df.csv",
          row.names = T)

write.csv(res_ELI_SFI_df, file = "resultados/res_ELI_SFI_df.csv",
          row.names = T)

sum(is.na(res_SFI_NIT_df$symbol))
sum(is.na(res_SFI_NIT_df$entrezid))

sum(is.na(res_ELI_NIT_df$symbol))
sum(is.na(res_ELI_NIT_df$entrezid))

sum(is.na(res_ELI_SFI_df$symbol))
sum(is.na(res_ELI_SFI_df$entrezid))

# Búsqueda de patrones de expresión y agrupación de muestras

res_SFI_NIT_genes = row.names(res_SFI_NIT_df)
res_ELI_NIT_genes = row.names(res_ELI_NIT_df)
res_ELI_SFI_genes = row.names(res_ELI_SFI_df)

comb = c(res_SFI_NIT_genes, res_ELI_NIT_genes, res_ELI_SFI_genes)

res_SFI_NIT_genes2 = comb %in% res_SFI_NIT_genes
res_ELI_NIT_genes2 = comb %in% res_ELI_NIT_genes
res_ELI_SFI_genes2 = comb %in% res_ELI_SFI_genes

counts_vennDiagram = cbind(res_SFI_NIT_genes2, res_ELI_NIT_genes2,
                           res_ELI_SFI_genes2)

library(limma)

results_vennDiagram = vennCounts(counts_vennDiagram)

for (i in 1:nrow(results_vennDiagram)) {
  d = sum(results_vennDiagram[i,ncol(results_vennDiagram)])
  if (d == 0) next
  results_vennDiagram[i,"Counts"] = results_vennDiagram[i,"Counts"] / d
}

vennDiagram(results_vennDiagram, cex = 0.9, main = "",

```

```

names = c("SFI vs NIT", "ELI vs NIT", "ELI vs SFI"),
circle.col = c("red", "blue", "green3"))

topVarGenes = head(order(rowVars(assay(vsd))), decreasing = TRUE), 20)

mat = assay(vsd)[topVarGenes, ]
mat = mat - rowMeans(mat)
colnames(mat) = targets_set$ShortName
row.names(mat) = mapIds(EnsDb.Hsapiens.v86,
                        keys=row.names(mat),
                        column="SYMBOL",
                        keytype="GENEID",
                        multiVals="first")
anno = as.data.frame(colData(vsd)[, "Group"])
row.names(anno) = colnames(mat)
colnames(anno) = "Group"
pheatmap(mat, annotation_col = anno, cluster_rows = T, cluster_cols = T,
          fontsize_row = 8, fontsize_col = 8)

# Análisis de significación biológica

res_SFI_NIT_df = as.data.frame(res_SFI_NIT)
res_SFI_NIT_df = subset(res_SFI_NIT_df, padj < 0.15)

res_ELI_NIT_df = as.data.frame(res_ELI_NIT)
res_ELI_NIT_df = subset(res_ELI_NIT_df, padj < 0.15)

res_ELI_SFI_df = as.data.frame(res_ELI_SFI)
res_ELI_SFI_df = subset(res_ELI_SFI_df, padj < 0.15)

library(clusterProfiler)

ego_ELI_NIT = enrichGO(res_ELI_NIT_df$entrezid,
                       OrgDb = "org.Hs.eg.db",
                       ont = "ALL",
                       pAdjustMethod = "BH",
                       pvalueCutoff = 0.01,
                       qvalueCutoff = 0.05,
                       readable = TRUE)

ego_ELI_SFI = enrichGO(res_ELI_SFI_df$entrezid,
                       OrgDb = "org.Hs.eg.db",
                       ont = "ALL",
                       pAdjustMethod = "BH",

```

```
    pvalueCutoff = 0.01,
    qvalueCutoff = 0.05,
    readable = TRUE)

ego_ELI_NIT = as.data.frame(ego_ELI_NIT)
write.csv(ego_ELI_NIT, file = "./resultados/ego_ELI_NIT.csv",
          row.names = F)

ego_ELI_SFI = as.data.frame(ego_ELI_SFI)
write.csv(ego_ELI_SFI, file = "./resultados/ego_ELI_SFI.csv",
          row.names = F)
```

Referencias

- [1] CSAMA. RNA-seq workflow - gene-level exploratory analysis and differential expression. Bioconductor 2016.
- [2] Love MI, Anders S, Huber W. Analyzing RNA-seq data with DESeq2. Bioconductor 2020.
- [3] Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)* 1995;57:289–300.