PEC2

Javier Amado Bouza

09 de junio, 2020

Table of Contents

El link al repositorio de GitHub es: https://github.com/amadobouza/ADO_PEC2.git

1. Abstract

A partir de secuencias obtenidas mediante técnicas de RNA-Seq se ha realizado un procesamiento de las mismas, para obtener un análisis de expresión diferencial. El origen de las secuencias son análisis de tiroides en 3 diferentes grados de infiltración: Not Infiltrated Tissues (NIT), Small Focal Infiltrates (SFI), y Extensive Lymphoid

Infiltrates (ELI). Se detectó expresión diferencial en todas las comparaciones realizadas, y los datos obtenidos se representaron gráficamente para ver cuáles son las rutas bioquímicas más enriquecidas.

2. Objetivos

Estudiar las diferencias en la expresión génica entre 3 grados de infiltración en tejido de tiroides.

3. Materiales y métodos

3.1 Naturaleza de los datos

Los datos de RNA-Seq de este estudio se encuentran alojados en el repositorio GTEx1. Se hallan en forma de dos archivos en formato CSV. El primero de ellos llamado counts.csv, y el segundo targets.csv.

Counts.csv contiene una tabla con el código de Gene Ontology de cada secuencia leída en el estudio, y en las columnas cada una de las muestras medidas siendo las celdas el número de counts de cada secuencia en esa muestra. Estas medidas son crudas, ya que todavía no se han procesado.

Targets.csv contiene una tabla que asocia cada muestra del archivo counts una serie de variables que la definen.

Este estudio presenta un solo factor, que es el grado de infiltración en el tejido tiroideo. Aquí aparecen 3 niveles: Not Infiltrated Tissues (NIT), Small Focal Infiltrates (SFI), y Extensive Lymphoid Infiltrates (ELI). El número de muestras para cada nivel es de 236 para el grupo NIT, 42 para el grupo SFI, y 14 para el grupo ELI. Esto hace un total de 292 muestras.

3.2 Métodos utilizados durante el procedimiento de análisis.

Para este estudio hemos utilizado el software estadístico R, y nos hemos basado en el pipeline “Analyzing RNA-seq data with DESeq2” que es el oficial de este paquete de R.

3.2.1 Carga de los archivos counts y targets en RStudio

Para este procedimiento utilizamos la función read.csv. Llamando a cada nuevo dataset con el nombre de su archivo original. Además, para que el informe pudiese ejecutarse desde cualquier computador(siempre que en la misma carpeta se encuentren los respectivos archivos csv) eliminamos las rutas absolutas en la carga de los archivos.

3.2.2 Preparación de los datos para comenzar el análisis

Tal y como se indica en el PDF de la PEC2 realizamos un muestreo aleatorio de 10 muestras de cada uno de los grupos. Para ello fijamos previamente una semilla. Tras unir los datos extraídos del dataset counts con los factores extraídos del dataset targets obtenemos un DESeqDataSet con 56202 filas (una por cada secuencia), y 30 columnas (una por cada muestra medida).

Como indicamos en puntos anteriores, los valores en este objeto son valores crudos de counts, y no están normalizados. Pero es importante que para utilizar la librería DESeq2 la alimentemos con datos de counts. El modelo de DESeq corrige internamente para el tamaño de las librerías, así que los datos transformados o normalizados no sirven para el flujo de trabajo realizado en este estudio.

Aunque per se no es necesario un filtrado los genes con menor número de counts antes de utilizar las funciones del paquete DESeq2. Existen dos razones que lo hacen útil: la primera es que con ello reducimos el tamaño de memoria del objeto dds, y la segunda es que con el filtrado incrementamos la velocidad de las funciones de transformación y testeo de nuestro pipeline.

En este caso hemos realizado un filtrado suave, eliminando sólamente las filas que tienen menos de 10 reads en total. Así, tras este proceso tenemos 36338 genes mientras que anteriormente teníamos 56202.

3.2.3 Reducción de la heterocedasticidad

Para testear la expresión diferencial operaremos con datos en crudo. Pero como ejercicio para observar la distribución de los datos vamos a realizar una transformación que reduce la heterocedasticidad de los mismos.

Normalmente la varianza de los datos de NGS aumenta según se incrementa el valor absoluto de los counts. Para corregir ese efecto se utilizan transformaciones que la homogeneizan. En este estudio hemos utilizado la función *vst* del paquete *DESeq2*, ya que es la indicada para casos con 30 muestras o más. El nombre es acrónimo de variance stabilizing transformation. Para ello utilizaremos el dataset con los datos filtrados.

El efecto de vst además de homogeneizar la dispersión,también incluye corrección para factores de tamaño de las secuencias, y convierte los datos a escala logarítmica en base 2. El el resultado obtenido está almacenado en un objeto tipo DESeqTransform.

3.2.4 Identificación de los genes diferencialmente expresados

El fin último de este estudio es el de identificar genes diferencialmente expresados.

Para identificar genes diferencialmente expresados entre los distintos grupos del estudio utilizamos la función DESeq perteneciente al paquete DESeq2. Para ello partiremos de los datos almacenados en el dataset con los datos filtrados.

Esta función utiliza un modelo lineal generalizado, y utiliza varias funciones de manera interna para obtener los resultados finales:

- Estima los factores de tamaño mediante *estimateSizeFactors*.
- Estima la dispersión mediante *estimateDispersions*.
- Ajuste de un modelo lineal generalizado binomial negativo, y estadístico de Wald mediante *nbinomWaldTest*.

Los pasos dados por la función DESeq son básicamente los siguientes:

- Filtrado independiente automático de la media de los datos una vez normalizados.
- Detección y manejo automático de los outliers.
- Como método de inferencia, por defecto se utiliza el test de Wald para la significación de los coeficientes de GLM,

Obtenemos un DESeqDataSet con entre otros valores, los resultados de pvalor y log2foldchange para las comparaciones entre todos los grupos.

Posteriormente almacenamos en otros datasets el resultado concreto de cada uno de los 3 contrastes individuales realizados por la función DESeq.

Cada uno de los 3 DESeqResults contiene el resultado concreto de su contraste de expresión diferencial, lo que luego puede ser utilizado para todo tipo de comparaciones.

El shrinkage de los estimados del LFC es útil para su visualización y establecer ránkings de los genes, sin necesidad de filtros arbitrarios para los genes con bajo número de counts.

Para constreñir el LFC pasamos el DESeqDataSet a la función *lfcShrink*. Como estimados de la constricción utilizamos *ashr*, que es el estimador adaptativo de shrinkage del paquete ashr.

Los log fold changes obtenidos utilizan una distribución normal centrada en cero y con una escala que se ajusta a los datos.

Obtenemos 3 objetos de tipo DESeqResults, uno por cada comparación.

3.2.5 Comparación entre distintas comparaciones

Con los objetos DESeqResults del paso anterior y utilizando las funciones *vennCounts* y *vennDiagram* del paquete *limma* contamos los genes que se han seleccionado como diferencialmente expresados en cada una de las tres comparaciones. Así podemos ver de un vistazo cuántos genes están diferencialmente expresados en cada una de ellas, y cuántos son comunes.

Previamente a la utilización de las funciones antes indicadas realizamos un trabajo de selección y preparación de los resultados para que puedan ser utilizados por las mismas.

Obtenemos un diagrama de Venn a tres colores, uno por cada comparación. Conteniendo en cada uno el número de genes diferencialmente expresados, e indicando cuáles son comunes a las comparaciones.

3.2.6 Anotación de los resultados y análisis de significación biológica

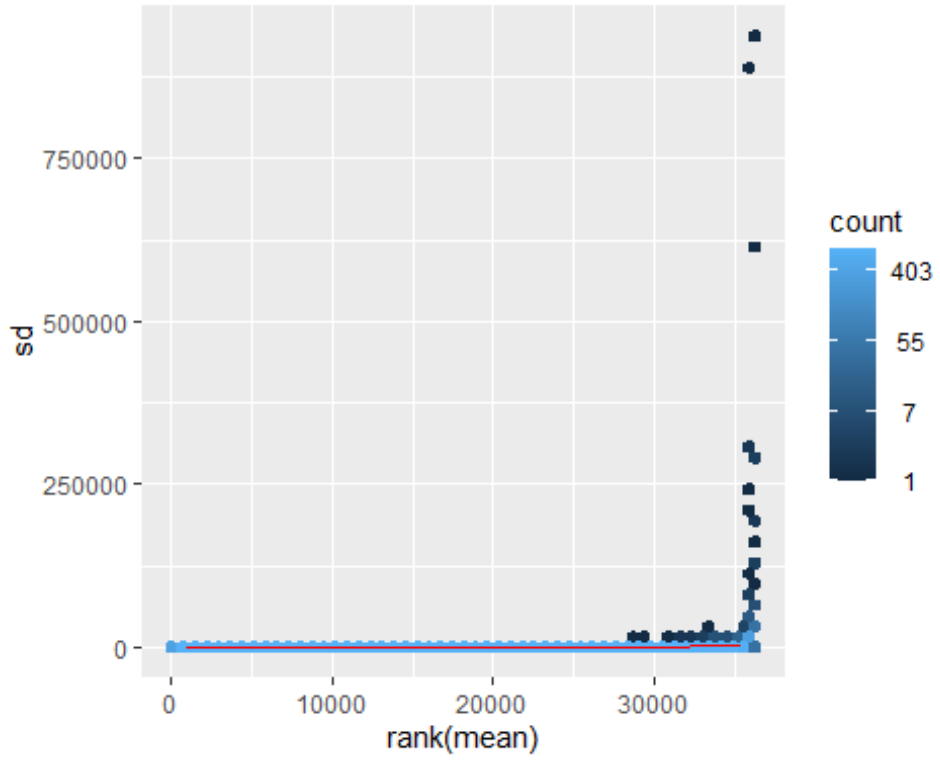
Para este paso utilizaremos la función *enrichGO* del paquete *clusterProfiler*.

Desde el principio del estudio, el nombre de cada una de las filas corresponde al código ensembl del transcrito en cuestión. Así que partiendo del código en cuestión, y utilizando la función antes indicada realizamos un análisis de las funciones de los genes diferencialmente expresados según la base de datos GO.

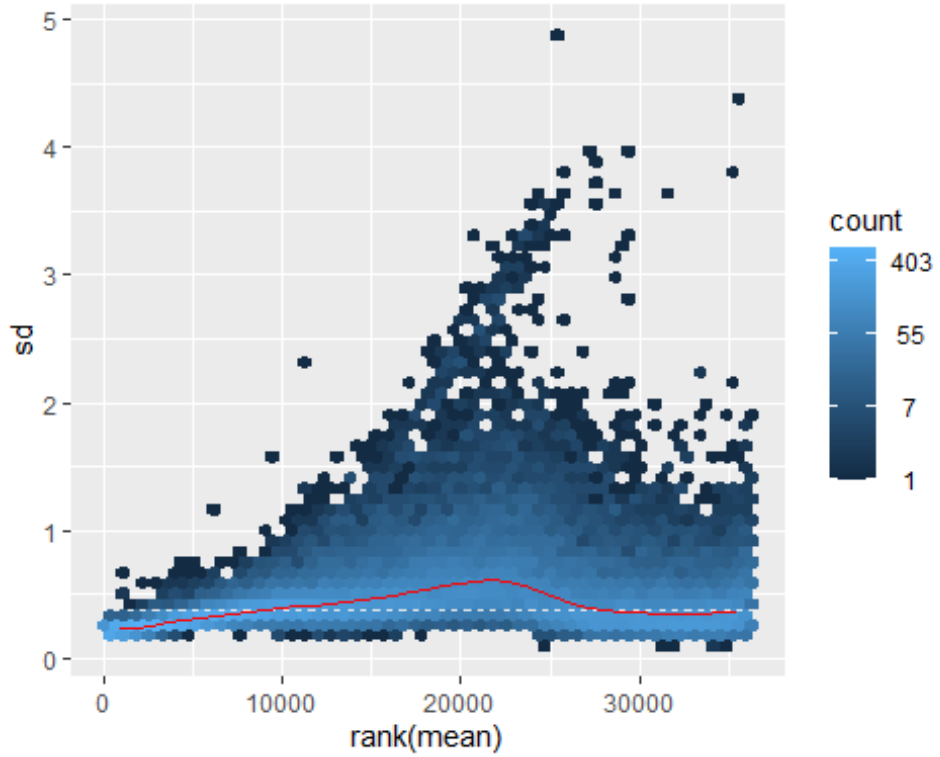
Obtenemos un enrichResult, que utilizaremos para hacer representaciones gráficas de las funciones enriquecidas en cada una de las comparaciones.

4. Resultados

4.1 Reducción de la heterocedasticidad



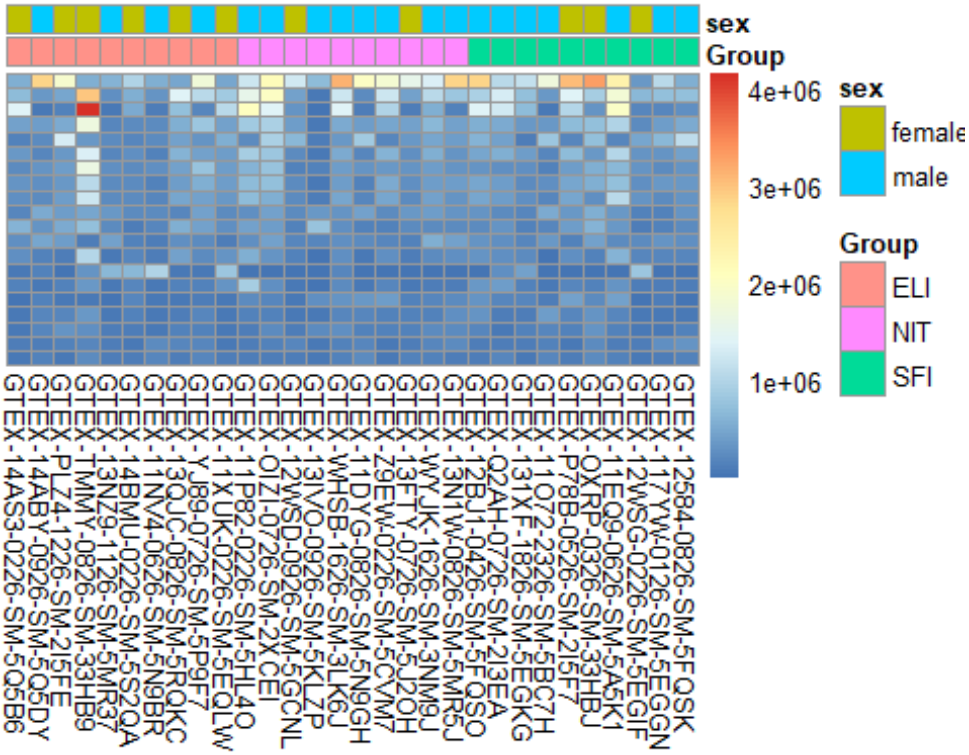
meanSdPlot de los datos sin transformar



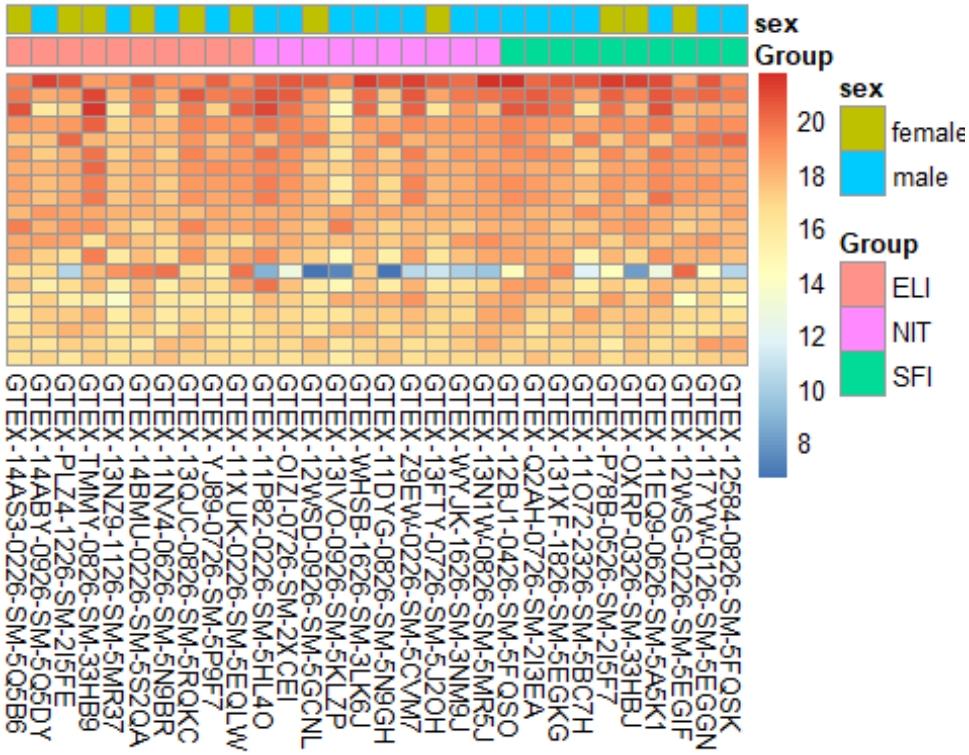
meanSdPlot de los datos transformados con vst

Vemos que en la gráfica de los datos sin transformar la desviación en los valores más altos es mucho más elevada que en cualquier otro rango de valores.

Sin embargo cuando analizamos la representación de los datos transformados con vst la distribución es mucho más homogénea, con lo que vemos el efecto positivo de la transformación.



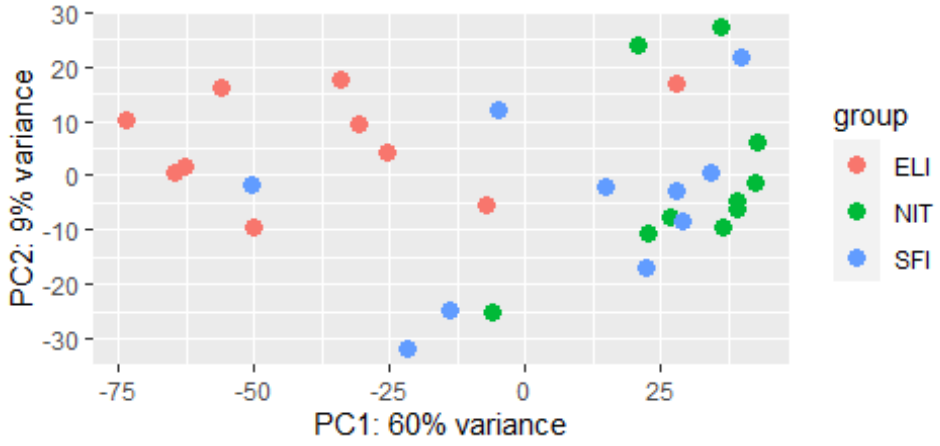
heatmap datos crudos



heatmap datos vst

Podemos observar que tras la transformación, los valores del número de counts y su distribución son más homogéneos.

A continuación realizamos un análisis de componentes principales



PCA datos transformados con vst

Podemos ver que los componentes principales 1 y 2 explican buena parte de la variabilidad, y también que las muestras tienden a agruparse con las pertenecientes a su grupo.

4.2 Genes diferencialmente expresados

Vamos a ver ahora las carácterísticas resumidas de los DESeqResults obtenidos tras la utilización de las funciones *DESeq* y *results*.

```
##
## out of 36332 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 3179, 8.7%
## LFC < 0 (down)     : 1079, 3%
## outliers [1]       : 0, 0%
## low counts [2]     : 5641, 16%
## (mean count < 1)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

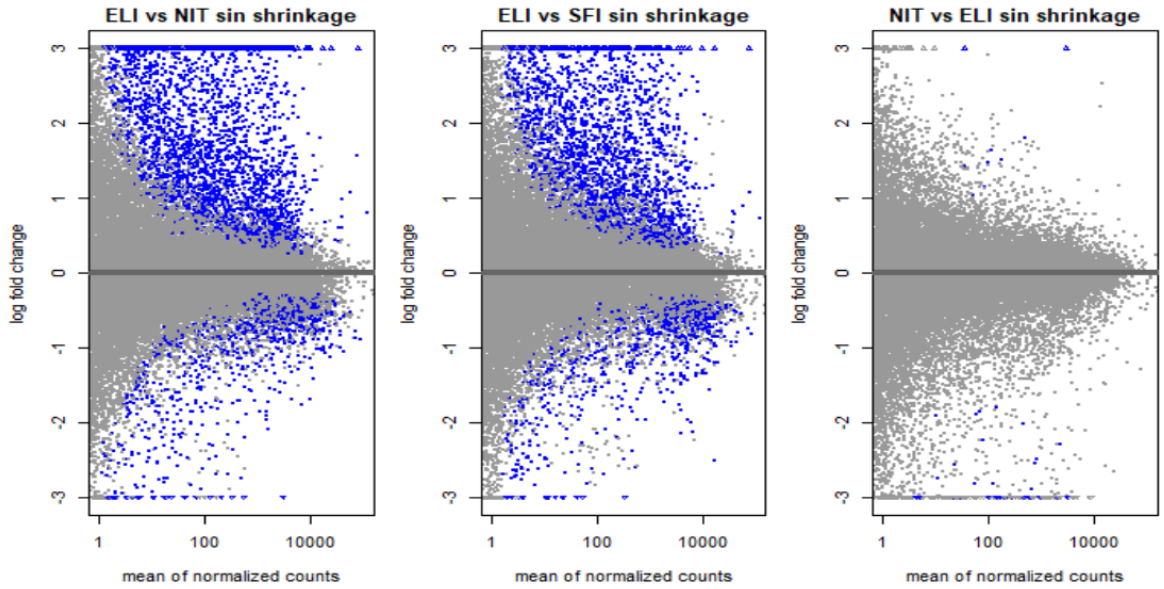
```
##
## out of 36332 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 2369, 6.5%
## LFC < 0 (down)     : 921, 2.5%
## outliers [1]       : 0, 0%
## low counts [2]     : 7050, 19%
## (mean count < 2)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results

##
## out of 36332 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 10, 0.028%
## LFC < 0 (down)     : 51, 0.14%
## outliers [1]       : 0, 0%
## low counts [2]     : 10572, 29%
## (mean count < 4)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Observamos los datos de genes que aumentan y disminuyen su expresión para cada una de las tres comparaciones.

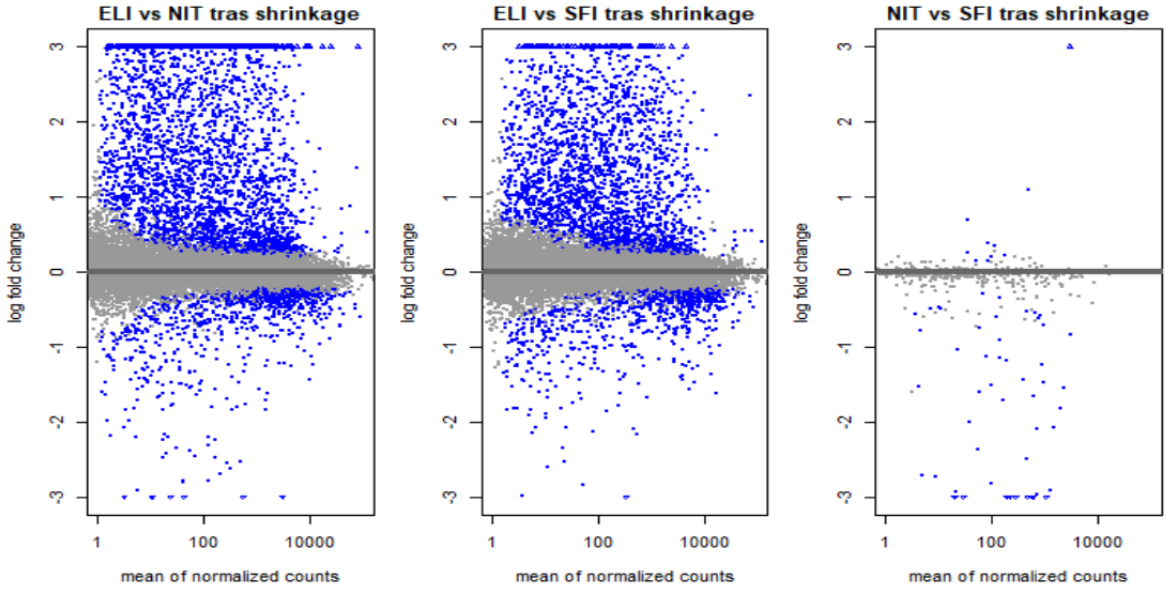
4.3 Efectos del shrinkage

Los MA plots son una manera muy efectiva de ver el efecto que el shrinkage ha tenido sobre los datos.

Para comparar entre los datos con y sin shrinkage y ver los efectos de este procedimiento. Vamos a graficar tres MA plots (uno por cada comparación) de datos sin shrinkage, y posteriormente el mismo número de gráficos tras el shrinkage.



MA plots con los datos sin shrinkage



MA plots con los datos con shrinkage

Comparando los MA plots de los datos sin constreñir frente a los de los datos una vez constreñidos podemos ver que el ruido asociado a los genes con bajo número de counts se ha reducido considerablemente en los segundos.

4.4 Comparación entre distintas comparaciones

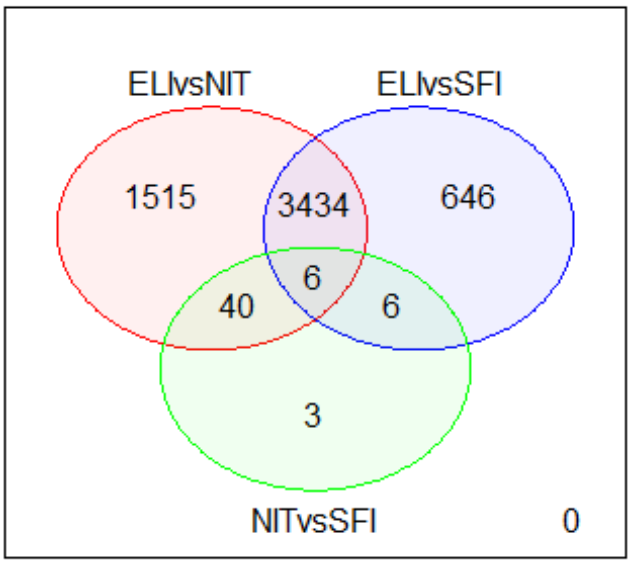
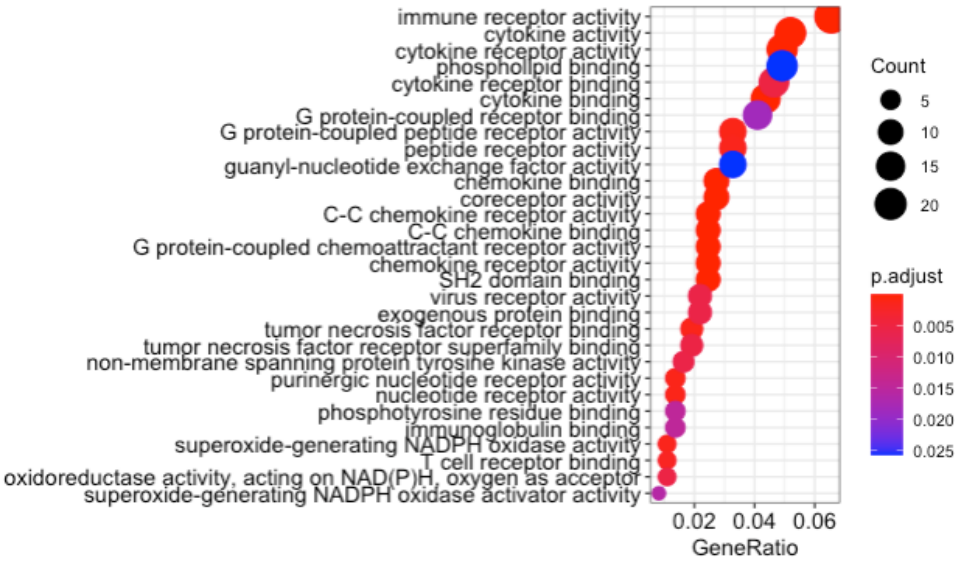


Diagrama de Venn del número de genes diferencialmente expresados entre comparaciones

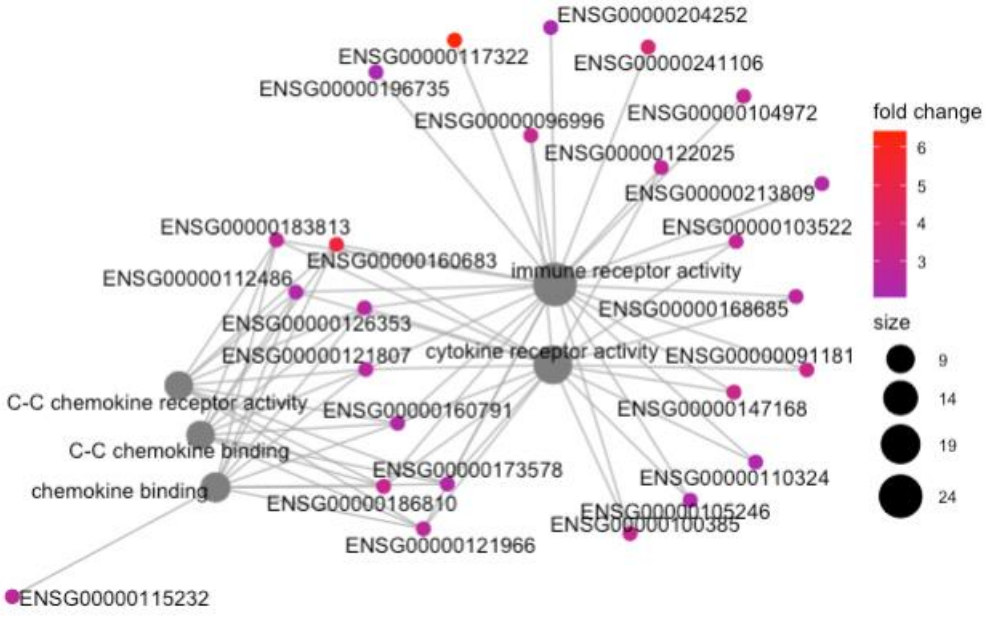
La comparación con mayor cantidad de genes diferencialmente expresados es la de ELIvsNIT, le sigue la de ELIvsSFI, y para finalizar la que menos genes diferencialmente expresados tiene es la comparación NITvsSFI.

4.5 Anotación de los resultados y significación biológica

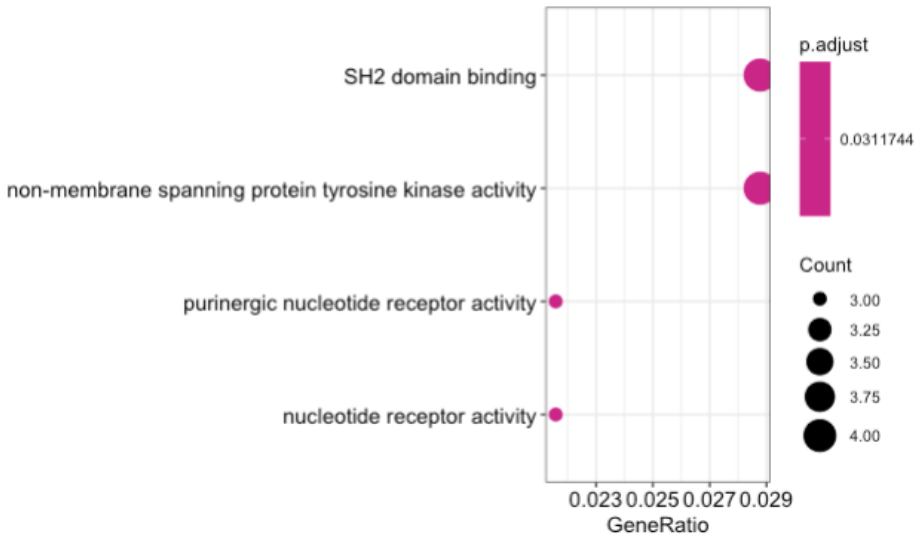
Veamos ahora las rutas bioquímicas en las que están implicadas los genes diferencialmente expresados en cada una de nuestras comparaciones.



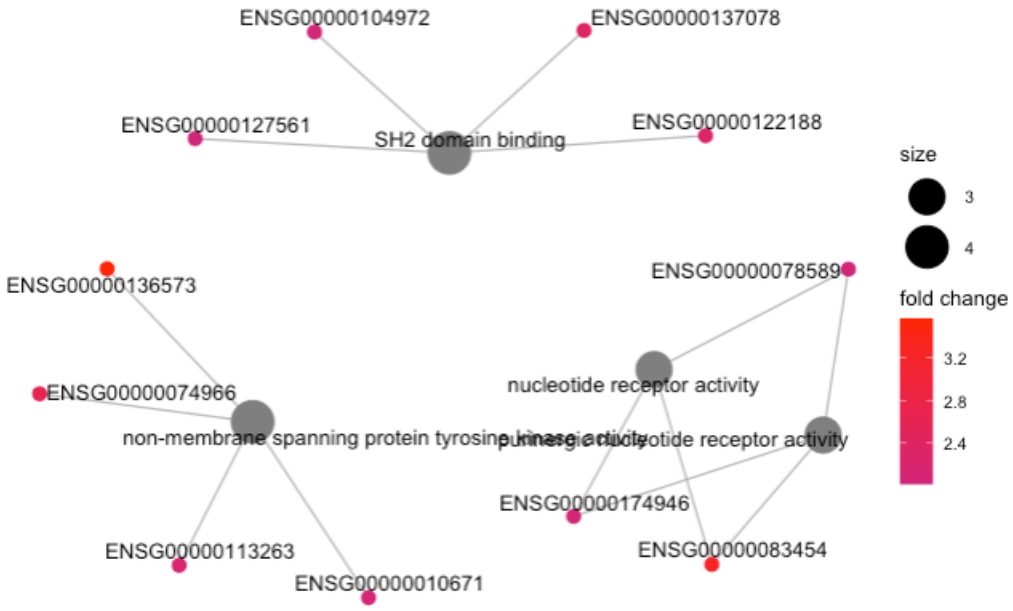
dotplot funciones biológicas genes comparación ELIvsNIT



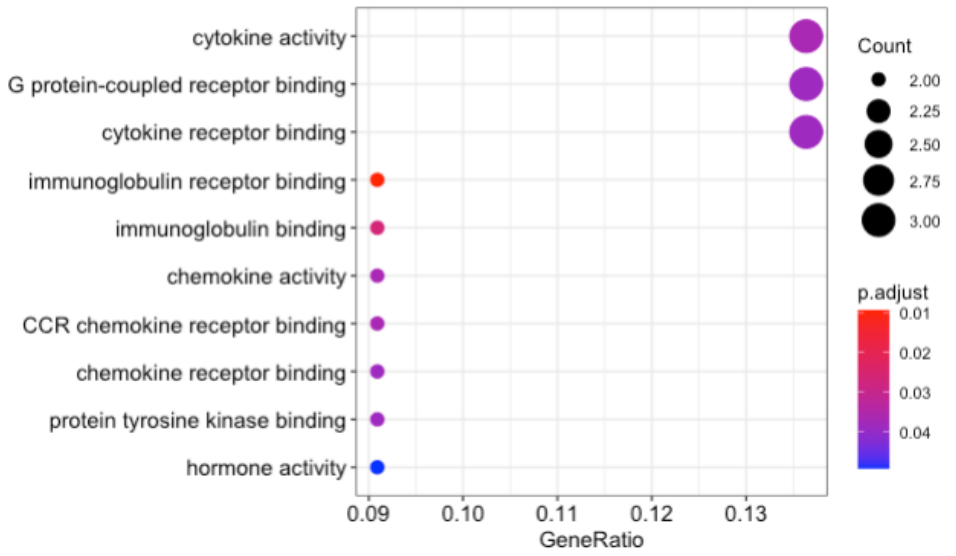
cnetplot redes bioquímicas genes comparación ELIvsNIT



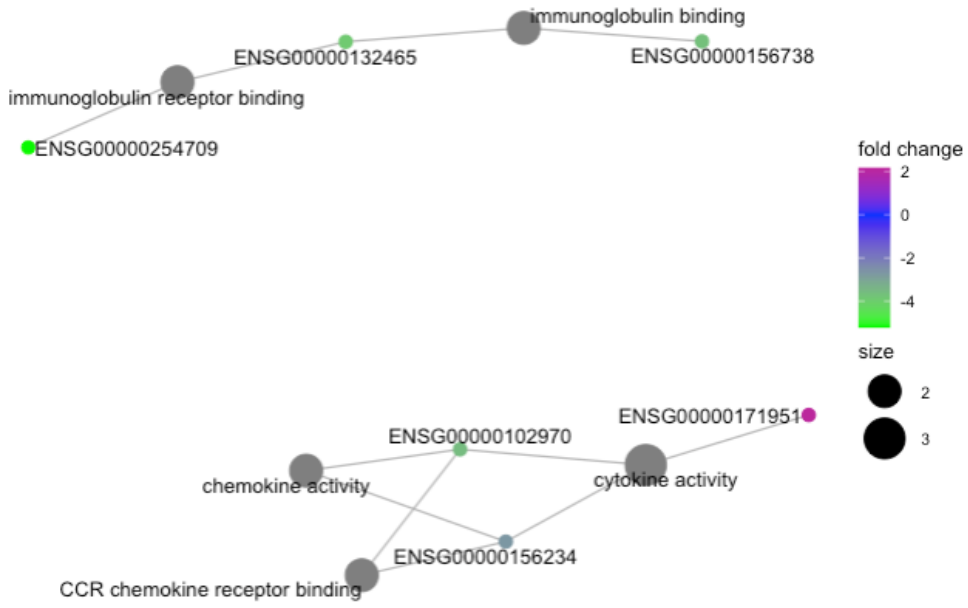
dotplot funciones biológicas genes comparación ELIvsSFI



cnetplot redes bioquímicas genes comparación ELIvsSFI



dotplot funciones biológicas genes comparación NITvsSFI



cnetplot redes bioquímicas genes comparación NITvsSFI

Podemos observar que los genes enriquecidos en las comparaciones están incolucrados en la actividad de las citoquinas y en la respuesta inmune.

5.Discusión

Sería interesante utilizar todos los datos disponibles a fin de obtener una mayor potencia en la búsqueda de genes diferencialmente expresados.

Por otra parte, tras realizar este protocolo tenemos una visión del estado actual del análisis de los datos de RNA-Seq. Se trata de un campo que está en plena ebullición, con mejoras constantes que lo hacen cada vez más potente.

6. Apéndice

```
library(dplyr)
library(stringr)
library(BiocManager)
library(DESeq2)
library(vsn)
library(pheatmap)
library(ashr)
library(limma)
library(clusterProfiler)
library(org.Hs.eg.db)
targets <- read.csv(file = "targets.csv", header = T, sep = ",")
counts <- read.csv(file = "counts.csv", header = T, sep = ";")
set.seed(15614)
datos<- targets %>% group_by(Group) %>% sample_n(size = 10, replace = F)
colnames(counts) <- str_replace_all(colnames(counts), "[.]", "-")
datos_sn <- c(datos$Sample_Name)
counts_2 <- counts[2:293][datos_sn]
tmp=gsub("\\..*","",counts[,1])
rownames(counts_2) <- tmp
factores <- as.data.frame(datos[,c("Group", "sex")])
rownames(factores) <- datos$Sample_Name
factores$Group<- factor(factores$Group)
factores$sex <- factor(factores$sex)
all(rownames(factores) == colnames(counts_2))
counts_3 <- DESeqDataSetFromMatrix(countData = counts_2,
                                   colData = factores,
                                   design = ~ Group)
#Filtramos aquellas filas que no tienen más de 10 reads en total
keep <- rowSums(counts(counts_3)) >= 10
counts_fil <- counts_3[keep,]
# Aplicación del algoritmo vst
counts_vst <- vst(counts_fil)
#Aplicamos la función DESeq sobre el dataset con los datos que han
#superado el filtro de número de counts.
counts_DESeq <- DESeq(counts_fil)
#Extraemos los resultados individuales almacenados en
```

```r
#el counts_DESeq indicando cuál es el contraste cuyos
#resultados deseamos obtener.
res_1 <- results(counts_DESeq,contrast = c("Group", "ELI", "NIT"))
res_2 <- results(counts_DESeq,contrast = c("Group", "ELI", "SFI"))
res_3 <- results(counts_DESeq,contrast = c("Group", "NIT", "SFI"))
#Pasamos a la función lfcShrink el dataset
#counts_DESeq indicando del resultado de cuál
#contraste queremos obtener sus resultado
#constreñidos
res_1_LFC <- lfcShrink(counts_DESeq, contrast = c("Group", "ELI", "NIT"),
type="ashr")
res_2_LFC <- lfcShrink(counts_DESeq, contrast = c("Group", "ELI", "SFI"),
type="ashr")
res_3_LFC <- lfcShrink(counts_DESeq, contrast = c("Group", "NIT", "SFI"),
type="ashr")
foldchange <- res_1_LFC@listData$log2FoldChange
names(foldchange) <- as.character(res_1_LFC@rownames)
foldchange_na <- na.omit(foldchange)
foldnaordenado <- sort(foldchange_na, decreasing = TRUE)
genes_1_LFC <- names(foldnaordenado)[abs(foldnaordenado) > 2]
gene_go <- enrichGO(genes_1_LFC,OrgDb = "org.Hs.eg.db",keyType =
"ENSEMBL",
                    ont = "MF",pvalueCutoff = 0.05,pAdjustMethod =
"BH",qvalueCutoff = 0.2,
                    minGSSize = 10,maxGSSize = 500,readable = FALSE, pool
= F)
#Dibujamos las desviaciones estándar de las filas frente a las medias
#de las filas. En este caso de los datos sin transformar
meanSdPlot(assay(counts_fil))
#Dibujamos las desviaciones estándar de las filas frente a las medias
#de las filas. En este caso de los datos transformados con vst
meanSdPlot(assay(counts_vst))
#Heatmap de los datos crudos
df <- as.data.frame(colData(counts_fil)[,c("Group", "sex")])
select <- order(rowMeans(counts(counts_fil,normalized=F)),
                decreasing=T)[1:20]
pheatmap(assay(counts_fil)[select,], cluster_rows=F, show_rownames=F,
         cluster_cols=F, annotation_col=df)
#Heatmap de los datos transformados
df_2 <- as.data.frame(colData(counts_vst)[,c("Group", "sex")])
pheatmap(assay(counts_vst)[select,], cluster_rows=F, show_rownames=F,
         cluster_cols=F, annotation_col=df_2)
plotPCA(counts_vst, intgroup=c("Group"))
summary(res_1_LFC)
summary(res_2_LFC)
summary(res_3_LFC)
#Creamos una cuadrículo para nuestros tres gráficos,
#y posteriormente graficamos cada uno de ellos.
par(mfrow=c(1,3), mar=c(4,4,2,1))
xlim <- c(1,1e5); ylim <- c(-3,3)
```

```r
plotMA(res_1, xlim = xlim, ylim= ylim, main = "ELI vs NIT sin shrinkage")
plotMA(res_2, xlim = xlim, ylim= ylim, main = "ELI vs SFI sin shrinkage")
plotMA(res_3, xlim = xlim, ylim= ylim, main = "NIT vs SFI sin shrinkage")
#Creamos una cuadrículo para nuestros tres gráficos,
#y posteriormente graficamos cada uno de ellos.
par(mfrow=c(1,3), mar=c(4,4,2,1))
xlim <- c(1,1e5); ylim <- c(-3,3)
plotMA(res_1_LFC, xlim = xlim, ylim= ylim, main = "ELI vs NIT tras
shrinkage")
plotMA(res_2_LFC, xlim = xlim, ylim= ylim, main = "ELI vs SFI tras
shrinkage")
plotMA(res_3_LFC, xlim = xlim, ylim= ylim, main = "NIT vs SFI tras
shrinkage")
#Hacemos el subset de los genes con un p_valor por debajo del umbral
subs_1 <- subset(res_1_LFC, res_1_LFC$padj <= 0.05)
subs_2 <- subset(res_2_LFC, res_2_LFC$padj <= 0.05)
subs_3 <- subset(res_3_LFC, res_3_LFC$padj <= 0.05)
#Cogemos los nombres de las filas que persisten
res_1_genes <- row.names(subs_1)
res_2_genes <- row.names(subs_2)
res_3_genes <- row.names(subs_3)
#Combining the two above..
comb_res <- c(res_1_genes, res_2_genes, res_3_genes)
#Comparing comb with the above three
comp_1_genes <- comb_res %in% res_1_genes
comp_2_genes <- comb_res %in% res_2_genes
comp_3_genes <- comb_res %in% res_3_genes
#Generating venn counts to plot venn diagram
suma_de_res <- cbind(comp_1_genes, comp_2_genes, comp_3_genes)
suma_venn <- vennCounts(suma_de_res)
vennDiagram(suma_venn, cex = 1,names = c("ELIvsNIT","ELIvsSFI",
"NITvsSFI"),
            circle.col = c("red", "blue","green"))
dotplot(gene_go, showCategory=30)
cnetplot(gene_go, foldChange=foldnaordenado)
foldchange_2 <- res_2_LFC@listData$log2FoldChange
names(foldchange_2) <- as.character(res_2_LFC@rownames)
foldchange_2_na <- na.omit(foldchange_2)
foldnaordenado_2 <- sort(foldchange_2_na, decreasing = TRUE)
genes_2_LFC <- names(foldnaordenado_2)[abs(foldnaordenado_2) > 2]
gene_2_go <- enrichGO(genes_2_LFC,OrgDb = "org.Hs.eg.db",keyType =
"ENSEMBL",
                      ont = "MF",pvalueCutoff = 0.05,pAdjustMethod =
"BH",
                      qvalueCutoff = 0.2,minGSSize = 10,maxGSSize = 500,
                      readable = FALSE, pool = F)
dotplot(gene_2_go, showCategory=30)
cnetplot(gene_2_go, foldChange=foldnaordenado_2)
foldchange_3 <- res_3_LFC@listData$log2FoldChange
names(foldchange_3) <- as.character(res_3_LFC@rownames)
```

```
foldchange_3_na <- na.omit(foldchange_3)
foldnaordenado_3 <- sort(foldchange_3_na, decreasing = TRUE)
genes_3_LFC <- names(foldnaordenado_3)[abs(foldnaordenado_3) > 2]
gene_3_go <- enrichGO(genes_3_LFC,OrgDb = "org.Hs.eg.db",keyType =
"ENSEMBL",
                      ont = "MF",pvalueCutoff = 0.05,pAdjustMethod =
"BH",
                      qvalueCutoff = 0.2,minGSSize = 10,maxGSSize = 500,
                      readable = FALSE, pool = F)
dotplot(gene_3_go, showCategory=30)
cnetplot(gene_3_go, foldChange=foldnaordenado_3)
```