Análisis de expresión génica diferencial usando datos de RNAseq

Biología celular y Molecular 2022

Introducción

Instalación y carga de los paquetes

En R, los *paquetes* son conjuntos de herramientas que desarrollaron otras personas y que están a disposición de la comunidad. En este trabajo vamos a usar diferentes paquetes para el *manejo*, *análisis* y *visualización* de datos. En particular, vamos a usar muchos paquetes pertenecientes a la familia de *Bioconductor* que, dentro del universo de R, incluye herramientas especializadas para el trabajo con datos de biología molecular (principalmente datos de secuencias). Para usar un paquete en R hay que realizar dos pasos: *instalarlo* (lo cual hacemos una única vez) e *importarlo* al espacio de trabajo de la sesión de R (lo cual repetimos cada vez que abramos el proyecto/script).

Instalación

Paquetes de Bioconductor

En función de la versión de R que tengamos instalada, el acceso a las bibliotecas de Bioconductor se hace de una u otra manera. Para chequear nuestra versión de R tipeamos `version` en la consola:

```
version
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          0.4
## year           2021
## month          02
## day            15
## svn rev        80002
## language       R
```

```
## version.string R version 4.0.4 (2021-02-15)
## nickname       Lost Library Book
```

En mi caso, la versión es 4.0.4.

Para instalar todos los paquetes de bioconductor que vamos a usar, primero guardamos sus nombres en un vector (`bioc.p`) usando la función para concatenar elementos (nombres en este caso) `c()`.

```
bioc.p=c("DESeq2", "vsn", "apeglm", "genefilter", "IHW", "edgeR")
```

Si la versión de R es < 3.5

```
if (as.numeric(version$minor)+as.numeric(version$major)>3.5) {
  source("https://bioconductor.org/biocManager.R")
  BiocInstaller::biocManager(bioc.p)
}
```

Si la versión de R es > 3.5

```
install.packages("BiocManager")

BiocManager::install(version = "3.12", force = T)

BiocManager::install(bioc.p, force = T)
```

Otros paquetes

Además de las herramientas específicas para trabajar con datos de secuencias, vamos a usar otras que sirven para, por ejemplo, manejar más fácilmente las tablas de datos o generar gráficos elaborados.

```
cran.p=c("dplyr", # manipulación de datos
         "tidyr", # manipulación de datos
         "ggplot2", # generación de gráficos
         "pheatmap", # gráficos de mapas de calor
         "RColorBrewer", # paletas de colores para los gráficos
         "PoiClaClu", # cálculo de distancias de Poisson
         #"glmpca",
         "ggbeeswarm")

new.packages=cran.p[!(cran.p %in% installed.packages()[,"Package"])]

if (length(new.packages)>0) {
  install.packages(new.packages)
}
```

Importación de *paquetes*

Si la instalación de todo funcionó bien (no saltaron Errores), entonces podemos cargar los paquetes en la sesión de trabajo de R actual. A diferencia de la instalación, esta parte hay que correrla cada vez que abramos una nueva sesión en R.

```
# Definimos la lista de nombres de los paquetes que R va a llamar

pckg.ls=c("DESeq2", "vsn", "apeglm", "genefilter", "IHW", "edgeR", # análiis de
datos moleculares
```

```
          "dplyr", "tidyr", # manipulación de datos
          "ggplot2", # generación de gráficos
          "pheatmap", # gráficos de mapas de calor
          "RColorBrewer", # paletas de colores para los gráficos
          "PoiClaClu", # cálculo de distancias de Poisson
          #"glmpca",
          "ggbeeswarm")

# E importamos los paquetes de esa lista

lapply(pckg.ls, # para cada nombre de la lista de paquetes instalados...
       require, # aplicar la función "require" para importarlo
       character.only = TRUE) # (ignorar este argumento, es para las mañas de
R)
```

Importación y preparación de los datos

Tabla de conteos

La tabla de conteos tiene, en cada **celda**, el número de fragmentos de RNA secuenciados de cada *muestra* que fueron identificados como provenientes de cada *gen*. Por lo tanto, tiene tantas columnas como muestras hayamos utilizado como fuente de RNA, y tantas filas como genes para los cuales se registró el numero de fragmentos de RNA secuenciados.

```
# importación
cts <- read.delim("./Conteos.tsv", row.names = "gene")

# dimensiones de la tabla
dim(cts) %>% `names<-`(c('genes', 'muestras'))
##    genes muestras
##    17483       96
```

Podemos explorar las primeras filas y columnas para tener una idea de cómo se organiza esta tabla.

```
# filas de 1 a 10, columnas de 1 a 3
cts[1:10, 1:3]
##            ctrlRNAi_M.BRN_S19 ctrlRNAi_M.BRN_S20 ctrlRNAi_M.BRN_S21
## OTAU000001                166                155                195
## OTAU000002                955                975                976
## OTAU000003               2318               2228               2466
## OTAU000004                  2                  1                  0
## OTAU000005                 29                 26                 38
## OTAU000006                 97                 78                 73
## OTAU000007                  0                  0                  0
## OTAU000008               1841               1902               1820
## OTAU000009                730                549                617
## OTAU000010                490                535                568
```

Vemos que cada **fila** tiene el nombre asociado al gen correspondiente. `OTAU000001` no es el *nombre* de ese gen propiamente dicho, sino que es el identificador que se le asignó a esa región codificante en el genoma que estamos utilizando. Si queremos saber de qué gen se trata, podemos utilizar el explorador de genomas **(ver link a i5k)** para buscar esa región codificante, copiar la secuencia de la misma y buscar secuencias parecidas en una base de datos de genes (por ej. en NCBI). Esta

búsqueda se realiza normalmente mediante el uso del algoritmo BLAST (*Basic Local Alignment Search Tool*). Por otra parte, en cada **columna** vemos un nombre asociado a la muestra de donde se sacó el RNA para secuenciar. Estos nombres tienen la información sobre el *diseño experimental*, es decir, el tratamiento con RNA de interferencia sobre dsx (ctrl/trat), el sexo (M/F), y el tejido (BRN/CHE/GEN/THE). `ctrlRNAi_M.BRN_S19` es una muestra del cerebro (BRN = brain) de un macho (M = male) al que no se le aplicó interferencia de RNA dirigida a dsx (ctrl) **¿Se aplicó RNAi ctrl? ¿Con qué secuencia?**.

Tabla de diseño experimental

Si bien los nombres de las columnas de la tabla de conteos tienen la información sobre el diseño experimental (tratamiento, sexo y tejido), esto es una fuente un poco "incómoda" para comparar los patrones de expresión entre grupos de forma programática. Para realizar análisis de comparación entre grupos, utilizamos una segunda tabla que tiene toda la información sobre las muestras bien ordenada.

```
samples0 <- read.delim("./Muestras.tsv", sep="\t", row.names=1)
samples <- samples0[,c(5, 6, 4, 2, 13)]
samples=rename(samples, lib.size=genome_unique)
```

Al igual que antes, podemos explorar la tabla visualmente filtrando solo algunas filas

```
# filas de 1 a 8
samples[1:8,]
##                     Treatment Sex Tissue sample lib.size
## ctrlRNAi_M.BRN_S19       ctr   M    BRN    S19 10748437
## ctrlRNAi_M.BRN_S20       ctr   M    BRN    S20 10129825
## ctrlRNAi_M.BRN_S21       ctr   M    BRN    S21 10571624
## ctrlRNAi_M.BRN_S22       ctr   M    BRN    S22 10899420
## ctrlRNAi_M.BRN_S23       ctr   M    BRN    S23 10275664
## ctrlRNAi_M.BRN_S24       ctr   M    BRN    S24 10281103
## ctrlRNAi_M.THE_S43       ctr   M    THE    S43 10175292
## ctrlRNAi_M.THE_S44       ctr   M    THE    S44  4640668
# (por ahora ignoremos la última columna)
```

Combinación en un único dataset

Para usar la mayoría de herramientas de análisis de este TP es necesario combinar estas dos tablas en un único *objeto*. Para esto, tenemos que chequear que cada columna de la tabla de conteos sea asociable inequívocamente a una fila de la tabla de muestras. Podemos corroborar que sean el mísmo número

```
# Número de columnas de la tabla de conteos
ncol(cts)
## [1] 96
# Número de filas de la tabla de muestras
nrow(samples)
## [1] 96
```

y que tengan los mismos nombres (para que no haya ambigüedades).

```
# el operador == devuelve TRUE si ambos elementos son iguales.
all(rownames(samples) == colnames(cts))
## [1] TRUE
```

Una vez que chequeamos esto podemos unir ambas tablas en un tipo de objeto utilizado por las herramientas de análisis de expresión diferencial (DE) incluidas en el paquete DESeq:

```
#objeto DESeqDataSet
data_0 <- DESeqDataSetFromMatrix(countData = cts,
                                 colData = samples[,c("Treatment", "Sex",
"Tissue")],
                                 design = ~ Tissue + Sex + Treatment)
```

Filtrado del dataset

Podemos descartar genes que tengan menos fragmentos secuenciados (reads) que un determinado umbral. Probar con diferentes valores.

```
reads.minimos=100

# bajo el umbral (muy pocos reads)
c(rowSums(counts(data_0)) < reads.minimos ) %>% sum()
## [1] 5011
# sobre el umbral (suficientes reads)
c(rowSums(counts(data_0)) >= reads.minimos ) %>% sum()
## [1] 12472
```

Para el resto del trabajo vamos a utilizar un umbral de 10 lecturas para considerar genes con suficiente información, pero sin descartar demasiados.

```
reads.minimos=10

data_filtered <- data_0[rowSums(counts(data_0)) >= reads.minimos,]
```

Exploración y análisis

Comparación entre conjuntos de muestras

```
poisd <- PoissonDistance(t(counts(data_filtered)))

#Trazamos el mapa de calor en una figura a continuaci?n
samplePoisDistMatrix <- as.matrix( poisd$dd )

# rownames(samplePoisDistMatrix) <- paste(data_filtered$Tissue,
#                                         data_filtered$Sex,
#                                         data_filtered$Treatment,
#                                         sep = " - " )



rownames(samplePoisDistMatrix) <- rownames(samples)
colnames(samplePoisDistMatrix) <- rownames(samples)

#colnames(samplePoisDistMatrix) <- NULL

colors <- colorRampPalette( brewer.pal(9, "YlOrRd")[6:1] )(255)
```

```
pheatmap(samplePoisDistMatrix,
         clustering_distance_rows = poisd$dd,
         clustering_distance_cols = poisd$dd,
         treeheight_row=0, treeheight_col=50,
         col = colors, fontsize_row = 4.5, border_color = NA, fontsize = 10,
         cluster_cols = T, cluster_rows = T,
         annotation_col  = samples[,1:3], annotation_row  = samples[,1:3],
         show_colnames=F, show_rownames=T,
         annotation_names_row=F, legend = F
         )
```

Análisis de expresión diferencial

Estabilización de la varianza

El tipo de herramientas estadísticas que vamos a utilizar para buscar patrones de expresión génica diferencial requiere que los datos cumplan ciertas propiedades. En particular,