

✓ Análisis GSEA

El Análisis de Enriquecimiento de Conjuntos de Genes (GSEA) es una herramienta bioinformática utilizada para interpretar conjuntos de genes y sus funciones en experimentos de expresión génica. Su objetivo principal es **determinar si un conjunto específico de genes muestra una distribución significativamente diferente en una lista ordenada de genes en comparación con lo que se esperaría al azar**.

Primero se realiza la preparación de Datos: Se comienza con un conjunto de datos de expresión génica que compara dos o más condiciones experimentales. Se asocian los genes con información biológica, como vías metabólicas, funciones celulares o procesos biológicos.

A continuación, se ordenan los genes: La lista de genes se ordena según su nivel de expresión diferencial entre las condiciones experimentales. Esto puede basarse en estadísticas de prueba, como el valor p o el cambio en el logaritmo del plegamiento ($\log_{2}\text{FoldChange}$).

Después se definen los Conjuntos de Genes: Se seleccionan conjuntos predefinidos de genes que representan vías biológicas, conjuntos de funciones o términos biológicos. Estos conjuntos a menudo se obtienen de bases de datos como Gene Ontology (GO), KEGG (Kyoto Encyclopedia of Genes and Genomes), Reactome, entre otras.

Luego se calcula el Estadístico de Enriquecimiento: Este estadístico evalúa si los genes del conjunto están distribuidos de manera uniforme a lo largo de la lista ordenada o si están agrupados en una región específica.

Finalmente se evalúa la Significancia: Se evalúa la significancia estadística del enriquecimiento de cada conjunto de genes mediante pruebas de permutación o métodos similares.

Para la visualización de los resultados del análisis GSEA se pueden utilizar distintos gráficos. En nuestro caso será un bubble plot

```
## Based on the scripts by Eva Sacristán and Sandra
#González (GENGS CBMSO)

suppressPackageStartupMessages({
  library(rstudioapi)
  library(DESeq2)
  library(clusterProfiler, quietly = TRUE)
  library(msigdbr, quietly = T)
  library(UpSetR, quietly = TRUE)
  library(enrichplot, quietly = TRUE)
  library(ggplot2)
})
```

Primero se cargan las librerías necesarias para los posteriores análisis.

```
#Parameters
#Which database inside msigdbr?
category <- 'C2'
subcategory <- 'KEGG'
#msigdbr_collections()

#Will you use stat parameter for ordering or the shrinked log2fold change?
#Log2fold is traditional statistic, but lately t statistic has been more recommended
#https://www.biorxiv.org/content/10.1101/060012v3.full.pdf
statP <- T
#plot the x top categories
topCat <- 15
#parameter for group comparison: 'males_', 'females_' or NULL
sexo <- 'females_'
```

Este script establece algunos parámetros para realizar un análisis de enriquecimiento de conjuntos de genes (GSEA) utilizando la base de datos MSigDB

1. Configuración de la Base de Datos MSigDB:

category <- 'C2' establece la categoría de conjuntos de genes en MSigDB. En este caso, se ha seleccionado la categoría 'C2', que contiene conjuntos de genes curados de bases de datos biológicas, como vías metabólicas, señalización celular, etc. Este parámetro lo modificaremos para conseguir toda la información que queramos de distintas bases de datos.

subcategory <- 'KEGG' establece la subcategoría dentro de la categoría 'C2'. En este caso, se ha seleccionado 'KEGG', lo que indica que se utilizarán conjuntos de genes de la base de datos KEGG en MSigDB. Este parámetro también lo modificaremos

2. Parámetros para el Análisis:

statP <- T determina si se utilizará el valor estadístico (como t-statistic) o el logaritmo plegamiento ajustado (shrinked log2fold change) para ordenar los genes en la lista. En este caso, T indica que se utilizará el valor estadístico.

topCat <- 15 especifica cuántas de las categorías principales se visualizarán en el resultado del análisis. En este caso, se mostrarán las 15 categorías principales.

sexo <- 'females_' Introduce un parámetro para la comparación entre grupos. En este caso, se utiliza 'females_' como un parámetro que podría indicar una comparación específica relacionada con género o sexo (por ejemplo, diferencias en conjuntos de genes entre grupos de hembras).

```
#Paths
workingD <- rstudioapi::getActiveDocumentContext()$path
setwd(dirname(workingD))
#Inputte
input <- paste0('DEG_results_', sexo,'sinFamilia/deseq_objects.RData')

#Outputs
resD0 <-paste0('results_GSEA_', sexo,'sinFamilia/')
if (statP){
  resD1 <- paste0(resD0,'stat/')
} else {
  resD1 <- paste0(resD0, 'log2fold/')
}
resD <- gsub('::','_',paste0(resD1,category,'_', subcategory, '/'))
if (!file.exists(resD)){
  dir.create(file.path(resD), recursive = TRUE)
}

resTSV <- paste0(resD,'GSEA_results_', sexo,'sinFamilia.txt')
dotplotF <- paste0(resD, "dotplot_", sexo,"sinFamilia.jpeg")
geneconceptF <- paste0(resD,'gene_concept_net_', sexo,'sinFamilia.jpeg')
ridgeF <- paste0(resD,'GSEA_ridge_', sexo,'sinFamilia.jpeg')
upsetF <- paste0(resD,'upset_plot_', sexo,'sinFamilia.jpeg')
gseaplotsF <- paste0(resD,'all_gseaplots', sexo,'sinFamilia.jpeg')
```

Este script está estableciendo rutas de archivos y directorios para el análisis GSEA, específicamente para el procesamiento y almacenamiento de los resultados.

1. Configuración del Directorio de Trabajo:

workingD <- rstudioapi::getActiveDocumentContext()\$path obtiene la ruta del directorio de trabajo actual del documento activo en RStudio.

setwd(dirname(workingD)) establece el directorio de trabajo actual al directorio padre de la ubicación del documento activo. Esto se hace para asegurarse de que las rutas de archivos y directorios se definen correctamente.

2. Definición de Rutas y Nombres de Archivos:

input <- paste0('DEG_results_', sexo,'sinFamilia/deseq_objects.RData') define la ruta al archivo que contiene los resultados del análisis de expresión génica diferencial.

3. Definición de directorios para resultados:

resD0 <-paste0('results_GSEA_', sexo,'sinFamilia/') directorio principal para los resultados de GSEA.

resD1 <- paste0(resD0,'stat/') o **resD1 <- paste0(resD0, 'log2fold/')** directorio específico dependiendo de si se utiliza el valor estadístico o el logaritmo del plegamiento para ordenar los genes.

resD <- gsub('::','_',paste0(resD1,category, '_', subcategory, '/')) define el directorio de resultados específicos para la categoría y subcategoría seleccionadas, reemplazando los dos puntos (":") en el nombre de la categoría/subcategoría con guiones bajos.

if (!file.exists(resD)){dir.create(file.path(resD), recursive = TRUE)} crea el directorio resD si no existe.

4. Definición de nombres de archivos de salida:

```
resTSV <- paste0(resD,'GSEA_results_', sexo,'sinFamilia.txt')
dotplotF <- paste0(resD, "dotplot_", sexo,"sinFamilia.jpeg")
geneconceptF <- paste0(resD,'gene_concept_net_', sexo,'sinFamilia.jpeg')
ridgeF <- paste0(resD,'GSEA_ridge_', sexo,'sinFamilia.jpeg')
upsetF <- paste0(resD,'upset_plot_', sexo,'sinFamilia.jpeg')
gseaplotsF <- paste0(resD,'all_gseaplots', sexo,'sinFamilia.jpeg')
```

```
#1) Load data
load(input)

if(statP){
  res <- results(dds, contrast = c('Grupo', 'DH', 'C'))
  res <- res[complete.cases(res),]
  dat <- res$stat
  names(dat) <- as.character(rownames(res))
  dat <- sort(dat, decreasing=TRUE)
} else { #when using log2fold change it is necessary to shrink the values
  shrink <- lfcShrink(dds, coef = 12, type="apeglm", quiet =T)
  dat <- shrink$log2FoldChange
  names(dat) <- as.character(rownames(shrink))
  dat <- sort(dat, decreasing=TRUE)
}
```

Este bloque de código carga los resultados del análisis de expresión génica diferencial y prepara un conjunto de datos (dat) para su posterior uso en el análisis de enriquecimiento de conjuntos de genes (GSEA). La elección entre usar estadísticas o el logaritmo del plegamiento ajustado depende del valor de statP.

1. Carga de Datos:

load(input) carga los resultados del análisis de expresión génica diferencial almacenados en el archivo especificado por la variable input. Este archivo generalmente contiene un objeto RData generado previamente que contiene los resultados del análisis de expresión génica diferencial.

2. Selección de Estadística a Utilizar:

if(statP) { ... } else { ... } realiza una bifurcación basada en el valor de statP. Si statP es TRUE, utiliza la estadística (por ejemplo, t-statistic) para ordenar los genes; si es FALSE, utiliza el logaritmo del plegamiento ajustado (shranked log2fold change).

a. Cuando statP es TRUE:

res <- results(dds, contrast = c('Grupo', 'DH', 'C')) extrae los resultados del análisis de expresión génica diferencial para la comparación de grupos especificada ('DH' frente a 'C').

res <- res[complete.cases(res),] elimina filas con valores NA.

dat <- res\$stat selecciona la columna de estadísticas (puede ser t-statistic) de los resultados.

names(dat) <- as.character(rownames(res)) asigna nombres a los datos basados en los nombres de las filas de los resultados.

dat <- sort(dat, decreasing=TRUE) ordena los datos de forma descendente.

b. Cuando statP es FALSE:

shrink <- lfcShrink(dds, coef = 12, type="apeglm", quiet = T) realiza el proceso de "shrinkage" (reducción) del logaritmo del plegamiento ajustado utilizando el método 'apeglm'.

dat <- shrink\$log2FoldChange selecciona el logaritmo del plegamiento ajustado.

names(dat) <- as.character(rownames(shrink)) asigna nombres a los datos basados en los nombres de las filas de los resultados "shranked".

dat <- sort(dat, decreasing=TRUE) ordena los datos de forma descendente.

```
#2) Calculate GSEA and write tables of results
#Get genes and categories
db_sets <- msigdbr(species = 'Rattus norvegicus', category = category,
                     subcategory = subcategory)%>%
  dplyr::select(gs_name, ensembl_gene)
head(db_sets) #each gene associated with each msig group
```

Este código obtiene conjuntos de genes específicos de MSigDB para la especie 'Rattus norvegicus' y la categoría/subcategoría proporcionadas. La salida, almacenada en la variable db_sets, contendrá información sobre los genes asociados a cada conjunto de genes en la base de datos MSigDB.

db_sets <- msigdbr(species = 'Rattus norvegicus', category = category, subcategory = subcategory) utiliza la función msigdbr para obtener conjuntos de genes de MSigDB específicos para la especie 'Rattus norvegicus' y la categoría y subcategoría especificadas. Estos conjuntos de genes pueden representar vías biológicas, funciones celulares u otros conjuntos temáticos.

dplyr::select(gs_name, ensembl_gene) selecciona las columnas 'gs_name' (nombre del conjunto de genes) y 'ensembl_gene' (identificación ENSEMBL del gen) del objeto resultante.

head(db_sets) muestra las primeras filas del conjunto de genes obtenido, proporcionando una vista previa de los datos.

```
#3) Perform GSEA
set.seed(1)
egs <- GSEA(geneList = dat, pvalueCutoff = 0.05, eps = 0, pAdjustMethod = "BH",
             seed = T, TERM2GENE = db_sets) #for more accurate p value set eps to 0
#https://bioconductor.org/packages/release/bioc/vignettes/fgsea/inst/doc/fgsea-tutorial.html
#head(egs@result)
egs_df <- data.frame(egs@result)
egs_df <- egs_df[, -c(1,2)]

write.table(egs_df, file = resTSV, sep= "\t", quote = F, row.names = T)

#Reconsider the top category number if there are less terms than specified
if (dim(egs_df)[1] < topCat){
  topCat <- dim(egs_df)[1]
}
```

Este bloque de código ejecuta el análisis GSEA, procesa los resultados y escribe un archivo de texto con información sobre los conjuntos de genes enriquecidos. También ajusta el número de categorías principales a mostrar (topCat) según el número real de categorías enriquecidas.

1. Ejecución del Análisis GSEA:

set.seed(1) fija la semilla del generador de números aleatorios para asegurar reproducibilidad. **egs <- GSEA(geneList = dat, pvalueCutoff = 0.05, eps = 0, pAdjustMethod = "BH", seed = T, TERM2GENE = db_sets)** realiza el análisis GSEA utilizando la función GSEA del paquete fgsea. **geneList = dat** es la lista ordenada de genes para el análisis GSEA. **pvalueCutoff = 0.05** establece el umbral de p-value para considerar significativamente enriquecido un conjunto de genes. **eps = 0** establece epsilon a cero para obtener valores p más precisos. **pAdjustMethod = "BH"** ajusta el p-values utilizando el método de Benjamini-Hochberg. **seed = T** utiliza la semilla fijada anteriormente. **TERM2GENE = db_sets** son conjuntos de genes de MSigDB utilizados para el análisis.

2. Procesamiento y Escritura de Resultados:

egs_df <- data.frame(egs@result) convierte los resultados del análisis GSEA a un marco de datos para facilitar su manipulación. **egs_df <- egs_df[, -c(1,2)]** elimina las columnas innecesarias del marco de datos resultante. **write.table(egs_df, file = resTSV, sep= "\t", quote = F, row.names = T)** escribe los resultados del análisis GSEA en un archivo de texto separado por pestañas (tsv). Esto incluye información sobre los conjuntos de genes enriquecidos y sus respectivos p-values ajustados.

3. Ajuste del Número de Categorías Principales a Mostrar:

if (dim(egs_df)[1] < topCat){topCat <- dim(egs_df)[1]} verifica si hay menos categorías enriquecidas que el número especificado (topCat). Si es así, ajusta topCat para que sea igual al número real de categorías enriquecidas.

▼ Plots

▼ Dot plot

```
#4) Plot the results
##Dotplot
jpeg(file = dotplotF, units = 'in', width = 15, height = 10,
      res = 300)
par(mar = c(2, 2, 2, 5))
title <- 'Dot plot with GSEA categories'
dotplot(egs, x = "GeneRatio", color = "p.adjust", showCategory = 15,
        font.size = 10, title = title)
invisible(dev.off())
```

Este bloque de código crea un dotplot para visualizar gráficamente las categorías enriquecidas obtenidas del análisis GSEA. Cada punto en el dotplot representa una categoría, y su posición vertical indica la proporción de genes enriquecidos en esa categoría. Los puntos están coloreados según su p-value ajustado.

jpeg(file = dotplotF, units = 'in', width = 15, height = 10, res = 300) inicia la creación de un archivo JPEG para almacenar el gráfico. Establece el tamaño y la resolución del gráfico.

par(mar = c(2, 2, 2, 5)) ajusta los márgenes del gráfico para dejar espacio suficiente para etiquetas y títulos.

title <- 'Dot plot with GSEA categories' establece el título del gráfico.

dotplot(egs, x = "GeneRatio", color = "p.adjust", showCategory = 15, font.size = 10, title = title) crea el dotplot utilizando la función dotplot del paquete fgsea. **egs** objeto resultante del análisis GSEA. **x = "GeneRatio"**, utiliza la proporción de genes enriquecidos para ordenar el dotplot.

color = "p.adjust" colorea los puntos según el p-value ajustado. **showCategory = 15** muestra las 15 categorías principales en el dotplot.

font.size = 10 establece el tamaño de la fuente en el dotplot. **title = title** utiliza el título especificado.

✓ Network plot

```
##Gene-concept network
jpeg(file = geneconceptF, units = 'in', width = 15, height = 10, res = 300)
par(mar = c(2, 2, 2, 5))
cnetplot(egs, categorySize="p.adjust", font.size = 15, colorEdge = T)
invisible(dev.off())
```

Este script crea un gráfico de red que representa visualmente la asociación entre genes y conjuntos de genes basándose en los resultados del análisis GSEA. Los nodos del gráfico pueden representar genes o conjuntos de genes, y el tamaño de los nodos y el color de los bordes pueden indicar la relevancia estadística de la asociación entre genes y conceptos enriquecidos. Este tipo de visualización facilita la identificación de genes clave y sus conexiones con funciones biológicas o categorías específicas.

`jpeg(file = geneconceptF, units = 'in', width = 15, height = 10, res = 300)` inicia la creación de un archivo JPEG para almacenar el gráfico de red. Establece el tamaño y la resolución del gráfico.

`par(mar = c(2, 2, 2, 5))` ajusta los márgenes del gráfico para dejar espacio suficiente para etiquetas y títulos.

`cnetplot(egs, categorySize="p.adjust", font.size = 15, colorEdge = T)` utiliza la función `cnetplot` del paquete fgsea para generar el gráfico de red. `egs` objeto resultante del análisis GSEA. `categorySize="p.adjust"` utiliza el valor ajustado (`p.adjust`) como tamaño de los nodos en la red. Este tamaño puede reflejar la significancia estadística de la asociación de un gen con un conjunto de genes. `font.size = 15` establece el tamaño de la fuente en el gráfico de red. `colorEdge = T` colorea los bordes (edges) de la red, lo que puede resaltar las relaciones entre genes y conjuntos de genes.

`invisible(dev.off())` cierra el dispositivo gráfico, finalizando la creación del archivo JPEG.

✓ Ridge line plot

```
##Ridge line plot
jpeg(file = ridgeF, units = 'in', width = 15, height = 10, res = 300)
par(mar = c(2, 2, 2, 5))
ridgeplot(egs, fill="p.adjust", orderBy= 'NES', core_enrichment = T,
          showCategory = topCat)
invisible(dev.off())
```

Este script crea un gráfico de línea de crestas que representa visualmente el enriquecimiento de conjuntos de genes basándose en los resultados del análisis GSEA. Cada cresta en el gráfico representa un conjunto de genes, y la posición y el color de las crestas reflejan la fuerza y significancia del enriquecimiento del conjunto de genes en las condiciones experimentales. Este tipo de visualización facilita la identificación de conjuntos de genes relevantes y su posición en términos de enriquecimiento.

`jpeg(file = ridgeF, units = 'in', width = 15, height = 10, res = 300)` inicia la creación de un archivo JPEG para almacenar el gráfico de línea de crestas. Establece el tamaño y la resolución del gráfico.

`par(mar = c(2, 2, 2, 5))` ajusta los márgenes del gráfico para dejar espacio suficiente para etiquetas y títulos.

`ridgeplot(egs, fill="p.adjust", orderBy= 'NES', core_enrichment = T, showCategory = topCat)` utiliza la función `ridgeplot` del paquete ggridges para generar el gráfico de línea de crestas. `egs` objeto resultante del análisis GSEA. `fill="p.adjust"` colorea las crestas según el valor ajustado (`p.adjust`), indicando la significancia estadística. `orderBy= 'NES'` ordena las crestas en el gráfico según el Score de Enriquecimiento Normalizado (NES por sus siglas en inglés), que es una medida de la fuerza y dirección del enriquecimiento del conjunto de genes. `core_enrichment = T` muestra solo los conjuntos de genes principales enriquecidos. `showCategory = topCat` muestra el número especificado de categorías principales en el gráfico de línea de crestas.

`invisible(dev.off())` cierra el dispositivo gráfico, finalizando la creación del archivo JPEG.

✓ Upset plot

```
##Upset plot (of the 10 first terms)
#Save the genes in each category in a list
genes_top <- as.data.frame(as.factor(head(egs$result$core_enrichment, topCat)))
list_top <- list()
for (i in 1:topCat) {
  list_top[[i]] <- unlist(strsplit(as.character(genes_top[i,]), split="/"))
}
```

Extrae los genes asociados a las categorías principales del resultado del análisis GSEA y los almacena en una lista.

```
#Store all unique gene IDs
uniq <- as.character(unique(names(dat)))
#Get top functions
func_top <- egs$Description[1:topCat]
```

Obtiene todos los identificadores únicos de genes y las descripciones de las categorías principales.

```
#Make sparse matrix with 1 for every gene in each category
mat <- matrix(0L, nrow = length(uniq), ncol = length(func_top))
for (gene in 1:length(uniq)) {
  for (func in 1:length(func_top)) {
    gen <- uniq[gene]
    if (gen %in% list_top[[func]]) {
      mat[gene,func] = 1
    }
  }
}
```

Crea una matriz dispersa que indica la presencia o ausencia de cada gen en las categorías principales. Un valor de 1 indica la presencia del gen en la categoría, mientras que 0 indica ausencia.

```
#Make a data frame
mat_df <- as.data.frame(mat)
colnames(mat_df) <- func_top
row.names(mat_df) <- uniq
```

Convierte la matriz a un marco de datos (data frame) donde las filas representan genes y las columnas representan categorías. Los nombres de las filas y columnas se establecen usando los genes y las descripciones de las categorías.

```
#Plot
jpeg(file = upsetF, units = 'in', width = 15, height = 10, res = 300)
upset(mat_df, nsets=10, order.by="freq", sets.bar.color="skyblue")
invisible(dev.off())
```

Crea un gráfico Upset que muestra la intersección de los conjuntos de genes en las 10 categorías principales. Cada barra representa una categoría, y las intersecciones muestran qué genes están presentes en más de una categoría.

```
###Plot the first 5 more abundant terms or all if there are less hits

for (j in 1:topCat){
  plot <- gseaplot2(egs, geneSetID=j, title = egs$Description[j], base_size=40, color="red")
  desc <- gsub(" ", "_", egs$Description[j], fixed = TRUE)
  filename <- paste0(resD, desc, ".jpeg")
  ggsave(plot, file=filename, device = "jpeg", units= "in", height = 15, width = 20)
}
```

Itera sobre las primeras 5 categorías (o las categorías superiores definidas por topCat) y crea un gráfico GSEA individual para cada una. El título del gráfico se toma de la descripción de la categoría, y el gráfico se guarda como un archivo JPEG.

```
#For all top categories at once:
gseap <- gseaplot2(egs, geneSetID = 1:topCat, pvalue_table = F)
ggsave(gseap, file=gseaplotsF, device = "jpeg", units= "in",
       height = 15, width = 20)
```

Crea un gráfico GSEA combinado que muestra la información para todas las categorías principales al mismo tiempo. El gráfico se guarda como un archivo JPEG.

▼ Bubble plot

Un bubble plot es un tipo de gráfico de dispersión que agrega una tercera dimensión a la representación de datos mediante el uso de burbujas de diferentes tamaños.

En un bubble plot, cada punto en el gráfico representa una observación y se coloca en coordenadas específicas según dos variables, generalmente en los ejes x e y. La tercera variable se representa mediante el tamaño de la burbuja. Así, cada burbuja tiene una posición en el gráfico determinada por sus valores en las dos variables principales y un tamaño que refleja el valor de la tercera variable.

Si se desea incluir una cuarta dimensión, como la puntuación de significancia estadística, el color de las burbujas podría representar esto. Burbujas más coloridas podrían indicar una mayor significancia estadística en términos de enriquecimiento de genes.

```
workingD <- rstudioapi::getActiveDocumentContext()$path
setwd(dirname(workingD))
```

Primero, el script se asegura de que el directorio de trabajo de R sea el mismo que el directorio donde se encuentra el documento activo en RStudio.

```
library(DESeq2)
library(GOpplot)
```

A continuación se cargan las librerías necesarias para generar el gráfico.

```
input <- 'DEG_results_males_sinFamilia/all_genes.csv'
res <- read.csv(input, header = TRUE, sep = ";")

colores <- c('red', 'green', 'blue', 'orange', 'yellow')

allgenes <- as.data.frame(res$log2FoldChange)
rownames(allgenes) <- res$x
colnames(allgenes) <- 'logFC'
allgenes$gene <- res$x
```

Este script carga datos desde un archivo CSV, crea un marco de datos (allgenes) con ciertas columnas y realiza algunas asignaciones de nombres. La última línea agrega una columna adicional 'gene' a allgenes, que contiene los valores de la columna 'x' de res. En otras palabras, prepara datos para análisis o visualización posteriores.

input <- 'DEG_results_males_sinFamilia/all_genes.csv' define una variable llamada input que almacena la ruta relativa o absoluta del archivo CSV ('DEG_results_males_sinFamilia/all_genes.csv') que se va a leer.

res <- read.csv(input, header = TRUE, sep = ";") lee el archivo CSV especificado en input utilizando la función read.csv(). Se espera que el archivo tenga cabeceras (header) y que el separador de columnas sea punto y coma (;). El resultado se almacena en un objeto llamado res.

colores <- c('red', 'green', 'blue', 'orange', 'yellow') crea un vector llamado colores que contiene cinco colores ('red', 'green', 'blue', 'orange', 'yellow'). Este vector de colores se usará más adelante en visualizaciones o asignaciones de colores.

allgenes <- as.data.frame(res\$log2FoldChange) crea un nuevo objeto llamado allgenes que contiene una columna llamada 'logFC' con los valores de res\$log2FoldChange. Se utiliza la función as.data.frame() para asegurarse de que el resultado sea un marco de datos.

rownames(allgenes) <- res\$x asigna los nombres de fila en allgenes utilizando la columna 'x' de res. Esto asume que la columna 'x' de res contiene nombres únicos para las filas.

colnames(allgenes) <- 'logFC' asigna el nombre de columna 'logFC' al conjunto de datos allgenes.

allgenes\$gene <- res\$x agrega una nueva columna llamada 'gene' a allgenes, que contiene los valores de la columna 'x' de res.

```
#Bubble plot of 68 categories validated from GSEA
circ0 <- read.csv('tabla_males_bubble.txt', header = T, sep="\t")
```

Este script en R está diseñado para leer un archivo de datos en formato de tabla tabulada (tsv) y crear un gráfico de burbujas (bubble plot) utilizando los datos contenidos en el archivo.

circ0 <- read.csv('tabla_males_bubble.txt', header = T, sep="\t") lee un archivo llamado 'tabla_males_bubble.txt' que está en formato de tabla tabulada (tsv) utilizando la función read.csv(). Los argumentos header = T indican que la primera fila del archivo contiene nombres de columna, y sep="\t" especifica que el separador de columnas es un tabulador. El resultado se almacena en un objeto llamado circ0.

```
#To compute the Z-score
cats <- as.data.frame(circ0$term)
cats$gene <- circ0$gene
cats$set_size <- circ0$count
colnames(cats) <- c('term', 'gene', 'set_size')

# Assuming you have loaded your data frames 'cats' and 'allgenes'

# Create an empty vector to store Z-scores
z_scores <- numeric(nrow(cats))
```

Este script en R está configurado para calcular los Z-scores de un conjunto de datos.

1. Creación de un marco de datos 'cats':

circ0term extrae la columna 'term' del objeto circ0 y crea un nuevo marco de datos (cats) a partir de ella.

circ0gene agrega una columna 'gene' a cats con los valores de la columna 'gene' de circ0.

circ0count agrega una columna 'set_size' a cats con los valores de la columna 'count' de circ0.

colnames(cats) <- c('term', 'gene', 'set_size') renombra las columnas del marco de datos a 'term', 'gene' y 'set_size'.

2. Creación de un vector vacío:

numeric(nrow(cats)) crea un vector numérico de longitud igual al número de filas en el marco de datos cats y lo asigna a la variable z_scores. Este vector se inicializa con todos los elementos establecidos en 0.

```
# Iterate through each row of 'cats' and calculate the Z-score
for (i in 1:nrow(cats)) {
  # Extract the list of genes in the current term and split it into a vector
  genes_in_term <- unlist(strsplit(cats$gene[i], "/"))

  # Filter 'allgenes' to get log2 fold changes for genes in the current term
  log2foldchanges_in_term <- allgenes$logFC[allgenes$gene %in% genes_in_term]

  # Calculate the Z-score for the term
  upregulated_count <- sum(log2foldchanges_in_term > 0)
  downregulated_count <- sum(log2foldchanges_in_term < 0)
  set_size <- length(log2foldchanges_in_term)

  z_score <- (upregulated_count - downregulated_count) / sqrt(set_size)

  # Store the calculated Z-score in the vector
  z_scores[i] <- z_score
}
```

Este script calcula el z-score.

El bucle for itera a través de cada fila del marco de datos cats utilizando el índice i que va desde 1 hasta el número de filas en cats (nrow(cats)).

Extracción de Genes: Para cada fila, se extrae la lista de genes en el término actual (cats\$gene[i]) y se divide en un vector utilizando strsplit(). La lista de genes se almacena en genes_in_term.

Filtrado de 'allgenes': Se filtra el marco de datos allgenes para obtener los **log2 fold changes correspondientes a los genes** en el término actual utilizando la condición allgenes\$gene %in% genes_in_term. Los log2 fold changes se **almacenan en log2foldchanges_in_term**.

Cálculo del Z-score: Se cuentan los genes upregulated (aquellos con log2 fold change > 0) y los genes downregulated (aquellos con log2 fold change < 0). Luego, se calcula el Z-score utilizando la fórmula (upregulated_count - downregulated_count) / sqrt(set_size).

Almacenamiento del Z-score: El Z-score calculado se almacena en el vector z_scores en la posición correspondiente (z_scores[i]).

```
# Add the Z-scores to the 'cats' data frame
cats$ZScore <- z_scores

#Relate to the circ data frame
circ0$zscore <- cats$ZScore[cats$term == circ0$term]
circ <- circ0.
```

Este fragmento de código R agrega los Z-scores calculados al marco de datos cats y luego relaciona estos Z-scores al marco de datos circ0.

Añadir Z-scores a 'cats': Se agrega una nueva columna llamada 'ZScore' al marco de datos cats y se le asigna el contenido del vector z_scores. Cada valor en esta columna corresponde al Z-score calculado para el término asociado en cats.

Relacionar a 'circ0': Se agrega una nueva columna llamada 'zsore' al marco de datos circ0. Esta columna se llena tomando los Z-scores de cats\$ZScore donde las columnas 'term' coinciden entre cats y circ0. En otras palabras, se asigna el Z-score calculado en cats al término correspondiente en circ0.

Asignar a 'circ': Se crea un nuevo marco de datos llamado circ y se le asigna el contenido de circ0.

```
#Z-score P value GSEA
trace(GOBubble, edit = T)

jpeg(file = 'prueba2.jpeg', units = 'in', width = 15, height = 20, res = 300)
par(mar = c(2, 2, 2, 5))
GOBubble(circ, labels = 2.5, colour = colores, ID = T, table.col=F, table.legend = F)

invisible(dev.off())
```

Este fragmento de código R está utilizando la función GOBubble() para trazar un gráfico de burbujas relacionado con análisis de enriquecimiento de genes (GSEA).

`trace(GOBubble, edit = T)` invoca la función trace() para realizar seguimiento de la función GOBubble. La opción edit = T indica que se está abriendo un entorno de edición para la función GOBubble.

`jpeg(file = 'prueba2.jpeg', units = 'in', width = 15, height = 20, res = 300)` establece las opciones para guardar el gráfico en un archivo JPEG llamado 'prueba2.jpeg' con un ancho de 15 pulgadas, alto de 20 pulgadas y una resolución de 300 píxeles por pulgada.

`par(mar = c(2, 2, 2, 5))` establece los márgenes de la región gráfica. Los números representan los márgenes izquierdo (2), derecho (2), inferior (2), y superior (5) respectivamente.

`GOBubble(circ, labels = 2.5, colour = colores, ID = T, table.col = F, table.legend = F)` llama a la función GOBubble con ciertos parámetros. circ es el marco de datos utilizado para generar el gráfico. Los otros parámetros controlan aspectos como la colocación de etiquetas (labels), colores (colour), visualización de identificadores (ID), y la visualización de tablas (table.col y table.legend).

`invisible(dev.off())` cierra la grabación del gráfico en el archivo JPEG.

✓ Trace(GOBubble, edit = T)

```
function (data, display, title, colour, labels, ID = T, table.legend = T,
  table.col = T, bg.col = F)
```

1. Parámetros de Entrada:

data: El marco de datos que contiene los resultados del análisis de enriquecimiento de genes (GSEA).

display: Controla la visualización y puede ser "single" o "multiple".

title: El título del gráfico.

colour: Los colores utilizados en el gráfico.

labels: Un umbral para etiquetar términos con un valor de p-valor ajustado superior al umbral.

ID: Un indicador booleano que especifica si se deben mostrar identificadores (ID) en lugar de términos.

table.legend: Un indicador booleano que controla si se muestra una leyenda de la tabla.

table.col: Un indicador booleano que controla si se muestra la leyenda de la tabla con colores.

bg.col: Un indicador booleano que controla el fondo de color.

```
{
  zscore <- adj_pval <- category <- count <- id <- term <- NULL
```

Esta línea de código en R inicializa varias variables (zscore, adj_pval, category, count, id, term) a NULL.

La razón de esta inicialización a NULL es preparar estas variables para su uso posterior dentro de la función GOBubble. Al asignarles NULL, se asegura de que estas variables comiencen sin ningún valor asignado. Luego, a medida que la función avanza, estas variables se llenarán con datos específicos del marco de datos proporcionado a la función GOBubble.

```
if (missing(display))
  display <- "single"
if (missing(title))
  title <- ""
if (missing(colour))
  cols <- c("chartreuse4", "brown2", "cornflowerblue")
else cols <- colour
if (missing(labels))
  labels <- 5
if (bg.col == T & display == "single")
  cat("Parameter bg.col will be ignored. To use the parameter change display to 'multiple'")
```

Este bloque de código R utiliza la función missing() para comprobar si ciertos parámetros de entrada han sido proporcionados cuando se llama a la función GOBubble. Si un parámetro falta, se le asigna un valor predeterminado.

if (missing(display)) display <- "single" verifica si el parámetro display está ausente. Si es así, se le asigna el valor predeterminado "single". Esto significa que si el usuario no proporciona un valor para display al llamar a la función, se utilizará "single" como valor por defecto.

if (missing(title)) title <- "" verifica si el parámetro title está ausente. Si es así, se le asigna una cadena vacía como valor predeterminado.

if (missing(colour)) cols <- c("chartreuse4", "brown2", "cornflowerblue") else cols <- colour verifica si el parámetro colour está ausente. Si es así, se le asigna un vector de colores por defecto ("chartreuse4", "brown2", "cornflowerblue"). Si el usuario proporciona un valor para colour, se

utiliza ese valor.

if (missing(labels)) labels <- 5 verifica si el parámetro labels está ausente. Si es así, se le asigna el valor predeterminado 5. Esto significa que si el usuario no proporciona un valor para labels al llamar a la función, se utilizará 5 como valor por defecto.

if (bg.col == T & display == "single") cat("Parameter bg.col will be ignored. To use the parameter change display to 'multiple'" verifica si bg.col es TRUE y display es "single". Si ambas condiciones son verdaderas, imprime un mensaje indicando que el parámetro bg.col se ignorará y sugiere cambiar display a "multiple". Este mensaje es informativo y podría ayudar al usuario a entender cómo se comporta la función en relación con estos parámetros

```
colnames(data) <- tolower(colnames(data))
if (!"count" %in% colnames(data)) {
  rang <- c(5, 5)
  data$count <- rep(1, dim(data)[1])
}
else {
  rang <- c(1, 30)
}
data$adj_pval <- -log(data$adj_pval, 10)
sub <- data[!duplicated(data$term), ]
```

Este bloque de código R realiza algunas manipulaciones en el marco de datos data, incluyendo cambios en los nombres de las columnas, la adición de una columna "count" en caso de que no exista, transformación de p-values ajustados, y la creación de un subconjunto sub sin filas duplicadas basadas en la columna "term".

colnames(data) <- tolower(colnames(data)) convierte todos los nombres de las columnas de data a minúsculas utilizando la función tolower(). Esto asegura que los nombres de las columnas estén en minúsculas para facilitar la manipulación y la consistencia.

if (!"count" %in% colnames(data)) {...} else {...} verifica si la columna llamada "count" está presente en data. Si no está presente, se ejecuta el bloque dentro del primer conjunto de llaves {...}, y si está presente, se ejecuta el bloque dentro del segundo conjunto de llaves {...}.

1. En el primer bloque (cuando "count" no está presente):

rang <- c(5, 5) define un vector rang con valores 5 y 5.

data\$count <- rep(1, dim(data)[1]) añade una nueva columna llamada "count" a data y la llena con repeticiones del valor 1, con la longitud igual al número de filas en data (obtenido con dim(data)[1]).

2. En el segundo bloque (cuando "count" sí está presente):

rang <- c(1, 30) define un vector rang con valores 1 y 30.

data\$adj_pval <- -log(data\$adj_pval, 10) calcula el logaritmo negativo en base 10 de los valores en la columna "adj_pval" y reemplaza los valores existentes en esa columna con los resultados. Esto es común en análisis de enriquecimiento de genes para transformar p-values ajustados.

sub <- data[!duplicated(data\$term),] crea un nuevo marco de datos llamado sub que contiene solo las filas únicas de data basadas en la columna "term". Elimina las filas duplicadas en función de los valores en la columna "term".

```
g <- ggplot(sub, aes(zscore, adj_pval, fill = category,
size = count)) + labs(title = title, x = "z-score",
y = "-log (adj p-value)") + geom_point(shape = 21, col = "black",
alpha = 1/2) + geom_hline(yintercept = 1.3, col = "orange") +
scale_size(range = rang, guide = "none")
```

Este bloque de código utiliza ggplot2 para construir un gráfico de dispersión (scatter plot) con burbujas, donde las posiciones x e y están determinadas por las variables zscore y adj_pval, respectivamente. La coloración de las burbujas está asociada a la variable category, el tamaño está asociado a la variable count, y se agregan puntos, una línea horizontal y etiquetas al gráfico.

g <- ggplot(sub, aes(zscore, adj_pval, fill = category, size = count)) crea un objeto ggplot llamado g utilizando el marco de datos sub. Especifica las estéticas del gráfico utilizando la función aes(). Las variables zscore y adj_pval se mapean a los ejes x e y, respectivamente. La variable category se mapea al color de relleno, y la variable count se mapea al tamaño de las burbujas.

labs(title = title, x = "z-score", y = "-log (adj p-value)") agrega etiquetas al gráfico. El título es igual al valor de la variable title. Los ejes x e y se etiquetan como "z-score" y "-log (adj p-value)", respectivamente.

geom_point(shape = 21, col = "black", alpha = 1/2) agrega puntos al gráfico (geom_point). Los puntos se representan con la forma 21 (un círculo sólido con borde), tienen un color de borde negro (col = "black"), y tienen una transparencia del 50% (alpha = 1/2).

geom_hline(yintercept = 1.3, col = "orange") agrega una línea horizontal al gráfico (geom_hline) en la posición y = 1.3. La línea es de color naranja.

scale_size(range = rang, guide = "none") ajusta el tamaño de las burbujas. La opción range = rang establece el rango del tamaño de las burbujas según los valores en el vector rang. La opción guide = "none" indica que no se debe mostrar la leyenda del tamaño.

```
if (!is.character(labels))
  sub2 <- subset(sub, subset = sub$adj_pval >= labels)
else sub2 <- subset(sub, sub$id %in% labels | sub$term %in%
  labels)
```

Este script determina el Subconjunto sub2 basado en el Valor de labels.

if (!is.character(labels)) sub2 <- subset(sub, subset = sub\$adj_pval >= labels) indica que si labels no es un carácter (es decir, un número), entonces sub2 se define como un subconjunto de sub donde los valores en la columna "adj_pval" son mayores o iguales a labels.

else sub2 <- subset(sub, sub\$id %in% labels) indica que si labels es un carácter, entonces sub2 se define como un subconjunto de sub donde los valores en la columna "id" o "term" coinciden con los valores en labels.

```
if (display == "single") {
  g <- g + scale_fill_manual("Category", values = cols,
    labels = c("Biological Process", "Cellular Component",
      "Molecular Function")) + theme(legend.position = "bottom") +
  annotate("text", x = min(sub$zscore) + 0.2, y = 1.4,
    label = "Threshold", colour = "orange", size = 4)
  if (ID)
    g <- g + geom_text(data = sub2, aes(x = zscore,
      y = adj_pval, label = id), size = 5)
```

En este bloque, se realizan configuraciones adicionales si display es "single". Esto incluye ajustes en la leyenda, la adición de un umbral ("Threshold"), y la posibilidad de agregar etiquetas al gráfico.

```
else g <- g + geom_text(data = sub2, aes(x = zscore,
y = adj_pval, label = term), size = 4)
if (table.legend) {
  if (table.col)
    table <- draw_table(sub2, col = cols)
  else table <- draw_table(sub2)
g <- g + theme(axis.text = element_text(size = 14),
  axis.line = element_line(colour = "grey80"),
  axis.ticks = element_line(colour = "grey80"),
  axis.title = element_text(size = 14, face = "bold"),
  panel.background = element_blank(), panel.grid.minor = element_blank(),
  panel.grid.major = element_line(colour = "grey80"),
```

En este bloque, se realizan configuraciones adicionales si display no es "single". Esto incluye configuraciones específicas para el modo "multiple", como la posibilidad de agregar colores de fondo (bg.col) y la opción de agregar etiquetas de identificación (ID) o términos al gráfico.

```
else {
  g + theme(axis.text = element_text(size = 14), axis.line = element_line(colour = "grey80"),
  axis.ticks = element_line(colour = "grey80"),
  axis.title = element_text(size = 14, face = "bold"),
  panel.background = element_blank(), panel.grid.minor = element_blank()
```