ANÁLISIS DE DATOS ÓMICOS

PEC2

Ager Zenzano Sarasola - UOC

2024-12-22

Contents

List of Figures

List of Tables

1 Introducción y Objetivos

1.1 Introducción

En esta PEC procederemos a la realización de un análisis de datos, que nos permitirán mejorar nuestra comprensión de un problema biológico mediante métodos, herramientas estadísticas y bioinformáticas. Haremos uso de métodos y herramientas para la selección de genes y el análisis de la significación biológica. Mostraremos los resultados obtenidos en cada fase del análisis y finalmente presentaremos una breve discusión sobre las posibles limitaciones encontradas.

La PEC se basará en los datos de un estudio que, utilizando un modelo `murino` (de ratón) investigaremos la utilidad de los antibióticos `LINEZOLID` y `VANCOMICINA` para inmunomodulación durante infecciones por `Staphylococcus aureus` resistente a `meticilina` (MRSA).

1.2 Estructura del documento

La construcción del presente informe lo hemos realizado mediante `R`, utilizando Rmarkdown para generar mediante código `R` embebido y `LateX` un documento `PDF`.

En cuanto al contenido, el informe tendrá la siguiente estructura: Tabla de contenidos, Introducción y Objetivos, Métodos, Resumen de resultados, Discusión, Referencias, y Apéndices. Para ello nos basaremos en el propio enunciado de la `PEC` y en los diferentes ejemplos y ejercicios realizados en la asignatura:[1][2][3][4] .

1.3 Objetivos

Nuestro objetivo será intentar caracterizar, a través del cambio en la expresión génica, el efecto de la infección y del tratamiento con antibióticos así como comparar los efectos de éstos.

Por ello, mediante este trabajo realizaremos un estudio de datos de microarrays y haremos las comparaciones siguientes:

- Infectados vs no infectados sin tratamiento.
- Infectados vs no infectados tratados con `LINEZOLID`.
- Infectados vs no infectados tratados con `VANCOMICINA`.

Esto generará tres listas de genes que deberemos, por un lado caracterizar mediante análisis de significación biológica y, por otro lado, comparar entre ellas.

[1] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".
[2] Mireia Ferrer, "Workflow de análisis de expresión diferencial usando RNA-seq".
[3] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".
[4] Sánchez-Pla, "An Introduction to Pathway Analysis with R and Bioconductor".

2 Métodos

2.1 Contexto

Las metodología que utilizaremos en el análisis, así como las técnicas y herramientas seguirán las pautas descritas en los materiales de diferentes ejemplos sobre el análisis de microarrays vistos en la asignatura.

Tanto el análisis estadístico como la manipulación de los datos lo haremos mediante el lenguaje de programación `R`. Además, haremos uso de la librerías del framework de `Bioconductor` (https://www.bioconductor.org) para el análisis de microarrays.

Los valores "crudos" de expresión los obtendremos directamente de los archivos `.CEL`, como veremos en un apartado posterior. Después pre-procesaremos los datos utilizando el método de `RMA`.[5] Una vez terminado el pre-procesado obtendremos los genes diferencialmente expresados y terminaremos con el post-procesado con el que realizaremos anotaciones sobre los genes, visualizaremos los perfiles de expresión y realizaremos el análisis de la significación biológica.

2.2 Preparación del entorno

2.2.1 Paquetes a instalar

Para ejecutar el código `R` del presente documento necesitaremos que estén instalados y cargadas en memoria una serie de paquetes `R` y otros específicos de `Bioconductor`, junto con las correspondientes dependencias que estos paquetes puedan tener.

Para gestionar adecuadamente los paquetes de `Bioconductor` necesitaremos tener instalado el paquete `BiocManager` tal y como se especifica a continuación.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install()
```

```
install.packages("dplyr")
install.packages("tidyr")
install.packages("tidyr")
install.packages("ggplot2")
install.packages("kableExtra")
install.packages("gplots")
install.packages("ggrepel")

BiocManager::install("oligo")
BiocManager::install("Biobase")
BiocManager::install("arrayQualityMetrics")
BiocManager::install("pvca")
BiocManager::install("affyio")
BiocManager::install("genefilter")
BiocManager::install("mouse4302.db")
BiocManager::install("limma")
BiocManager::install("annaffy")
BiocManager::install("clusterProfiler")
```

[5] Irizarry et al., "Exploration, normalization, and summaries of high density oligonucleotide array probe level data".

2.3 Preparación de los datos

2.3.1 Obtención de los datos

Para realizar el análisis de datos con microarrays, partiremos de un proyecto en `R` y dentro del mismo, crearemos un estructura de carpetas donde iremos guardando y mantendremos organizados todos los datos que iremos utilizando. Por ello, partiremos de un directorio raíz de trabajo, el cual lo denominaremos como directorio de trabajo (`As Working Directory`). A partir este directorio colgará toda la subestructura de carpetas y se localizarán los diferentes ficheros. Este directorio raíz lo asignaremos a la variable `workingDire` y de éste se encontrarán los subdirectorios:

- `downloads`, donde almacenaremos el fichero original descargado de en formato comprimido.
- `results`, donde almacenaremos los distintos resultados que iremos obteniendo durante el análisis.
- `celfiles`, donde almacenaremos los ficheros `.CEL` ya descomprimidos y el fichero `target.txt` con la información de las covariantes.

```
workingDire <-getwd()
downloadsDire <-file.path(workingDire, "downloads/")
resultsDire <- file.path(workingDire,"results/")
celfilesDire <- file.path(workingDire,"celfiles")
```

```
list.dirs(path = workingDire, full.names = FALSE, recursive = FALSE)
```

```
[1] "celfiles"  "downloads" "results"
```

Los datos crudos para el análisis los descargaremos del sitio de GEO https://www.ncbi.nlm.nih.gov/geo/query/a cc.cgi?acc=GSE38531 donde también encontraremos información sobre el estudio original.

```
list.files(downloadsDire)
```

```
[1] "GSE38531_RAW.tar"
```

Después de descargar el fichero comprimido procederemos a descomprimirlo y almacenar los ficheros `.CEL` en `celfilesDire`.

```
list.files(celfilesDire)
```

```
 [1] "GSM944831.CEL" "GSM944832.CEL" "GSM944833.CEL" "GSM944834.CEL"
 [5] "GSM944835.CEL" "GSM944836.CEL" "GSM944837.CEL" "GSM944838.CEL"
 [9] "GSM944839.CEL" "GSM944840.CEL" "GSM944841.CEL" "GSM944842.CEL"
[13] "GSM944843.CEL" "GSM944844.CEL" "GSM944845.CEL" "GSM944846.CEL"
[17] "GSM944847.CEL" "GSM944848.CEL" "GSM944849.CEL" "GSM944850.CEL"
[21] "GSM944851.CEL" "GSM944852.CEL" "GSM944853.CEL" "GSM944854.CEL"
[25] "GSM944855.CEL" "GSM944856.CEL" "GSM944857.CEL" "GSM944858.CEL"
[29] "GSM944859.CEL" "GSM944860.CEL" "GSM944861.CEL" "GSM944862.CEL"
[33] "GSM944863.CEL" "GSM944864.CEL" "GSM944865.CEL" "targets.txt"
```

Observamos que al descomprimir los ficheros, los nombres de éstos no coinciden exactamente con los nombres proporcionados en `targets`, por lo que realizaremos pequeños ajustes para que estos coincidan.

2.3.2 Selección de muestras para el análisis

Siguiendo el enunciado y con el fin de simplificar el análisis y particularizar en cierta medida el estudio eliminaremos algunas muestras: por un lado prescindiremos de las cinco muestras tomadas a las dos horas, y por otro lado seleccionaremos de manera aleatoria las muestras restantes de manera conservaremos únicamente cuatro muestras de cada grupo respecto a los datos de partida.

Para realizar dicha selección utilizaremos la función `filter_microarray()` de `R` que encontraremos en el fichero `selectSamples.R` que descargaremos de la plataforma de la UOC junto con el propio enunciado para realizar la PEC. Dicha función nos permitirá extraer 24 muestras distintas a cada uno con tan solo llamarla

usando como semilla (argumento "seed") IDP (por responsabilidad en la confidencialidad de datos tan solo utilizaremos los últimos 4 dígitos).

En el fichero `selectSamples.R` encontraremos además, la estructura de datos `allTargets` donde estará almacenada la información de las covariantes.

Aplicaremos la función `filter_microarray()` sobre `allTargets` y obtendremos un nuevo objeto `targets` que nos permitirá crear un nuevo `ExpressionSet` leyendo únicamente aquellos archivos `.CEL` que hayamos seleccionado. A partir de este `ExpressionSet` personalizado, con 24 muestras muestras procederemos a realizar nuestro análisis.

```
list.files(celfilesDire, pattern = ".txt")
```

```
[1] "targets.txt"
```

Table 1: *targets.txt.*

sample	infection	time	agent	groupo	sample_id	short_name
GSM944836.CEL	S. aureus USA300	hour 24	linezolid	inf_lin	GSM944836	GSM944836_inf_lin
GSM944850.CEL	S. aureus USA300	hour 24	linezolid	inf_lin	GSM944850	GSM944850_inf_lin
GSM944857.CEL	S. aureus USA300	hour 24	linezolid	inf_lin	GSM944857	GSM944857_inf_lin
GSM944864.CEL	S. aureus USA300	hour 24	linezolid	inf_lin	GSM944864	GSM944864_inf_lin
GSM944854.CEL	uninfected	hour 0	linezolid	uni_lin	GSM944854	GSM944854_uni_lin
GSM944847.CEL	uninfected	hour 0	linezolid	uni_lin	GSM944847	GSM944847_uni_lin
GSM944840.CEL	uninfected	hour 0	linezolid	uni_lin	GSM944840	GSM944840_uni_lin
GSM944861.CEL	uninfected	hour 0	linezolid	uni_lin	GSM944861	GSM944861_uni_lin
GSM944835.CEL	S. aureus USA300	hour 24	untreated	inf_unt	GSM944835	GSM944835_inf_unt
GSM944849.CEL	S. aureus USA300	hour 24	untreated	inf_unt	GSM944849	GSM944849_inf_unt
GSM944863.CEL	S. aureus USA300	hour 24	untreated	inf_unt	GSM944863	GSM944863_inf_unt
GSM944856.CEL	S. aureus USA300	hour 24	untreated	inf_unt	GSM944856	GSM944856_inf_unt
GSM944845.CEL	uninfected	hour 0	untreated	uni_unt	GSM944845	GSM944845_uni_unt
GSM944852.CEL	uninfected	hour 0	untreated	uni_unt	GSM944852	GSM944852_uni_unt
GSM944859.CEL	uninfected	hour 0	untreated	uni_unt	GSM944859	GSM944859_uni_unt
GSM944838.CEL	uninfected	hour 0	untreated	uni_unt	GSM944838	GSM944838_uni_unt
GSM944865.CEL	S. aureus USA300	hour 24	vancomycin	inf_van	GSM944865	GSM944865_inf_van
GSM944837.CEL	S. aureus USA300	hour 24	vancomycin	inf_van	GSM944837	GSM944837_inf_van
GSM944851.CEL	S. aureus USA300	hour 24	vancomycin	inf_van	GSM944851	GSM944851_inf_van
GSM944858.CEL	S. aureus USA300	hour 24	vancomycin	inf_van	GSM944858	GSM944858_inf_van
GSM944841.CEL	uninfected	hour 0	vancomycin	uni_van	GSM944841	GSM944841_uni_van
GSM944855.CEL	uninfected	hour 0	vancomycin	uni_van	GSM944855	GSM944855_uni_van
GSM944862.CEL	uninfected	hour 0	vancomycin	uni_van	GSM944862	GSM944862_uni_van
GSM944848.CEL	uninfected	hour 0	vancomycin	uni_van	GSM944848	GSM944848_uni_van

2.3.3 Lectura de los datos

El paso inicial que haremos con los datos, partiendo del subconjunto de 24 ficheros (`targets.txt`), será utilizar la librería `oligo` de `R` para leer los datos en bruto (archivos `CEL`) y almacenarlos en la variable `rawData`. Esta variable será un objeto `ExpressionSet` que está diseñado para almacenar y organizar varias fuentes de información diferentes en un único conjunto de datos.[6]

A continuación, mostramos las muestras leídas (ficheros `CEL` leídos) y almacenados en el objeto `rawData`.

[6] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".

```
rownames(pData(rawData))
```

```
 [1] "GSM944836.CEL" "GSM944850.CEL" "GSM944857.CEL" "GSM944864.CEL"
 [5] "GSM944854.CEL" "GSM944847.CEL" "GSM944840.CEL" "GSM944861.CEL"
 [9] "GSM944835.CEL" "GSM944849.CEL" "GSM944863.CEL" "GSM944856.CEL"
[13] "GSM944845.CEL" "GSM944852.CEL" "GSM944859.CEL" "GSM944838.CEL"
[17] "GSM944865.CEL" "GSM944837.CEL" "GSM944851.CEL" "GSM944858.CEL"
[21] "GSM944841.CEL" "GSM944855.CEL" "GSM944862.CEL" "GSM944848.CEL"
```

```
head(rawData)
```

```
ExpressionFeatureSet (storageMode: lockedEnvironment)
assayData: 6 features, 24 samples
  element names: exprs
protocolData
  rowNames: GSM944836.CEL GSM944850.CEL ... GSM944848.CEL (24 total)
  varLabels: exprs dates
  varMetadata: labelDescription channel
phenoData
  rowNames: GSM944836.CEL GSM944850.CEL ... GSM944848.CEL (24 total)
  varLabels: infection time ... short_name (6 total)
  varMetadata: labelDescription channel
featureData: none
experimentData: use 'experimentData(object)'
Annotation: pd.mouse430.2
```

2.4 Análisis exploratorio y control de calidad

2.4.1 Exploración de datos

Empezaremos con una exploración de datos procedentes de la lectura de los microarrays (crudos) realizada en el apartado anterior. Esta primera lectura, nos permitirá decidir si los datos necesitan alguna transformación, si presentan algún problema y si los grupos que deseamos comparar se separan mínimamente.

Para disponer de cierta descripción sobre las muestras que visualizaremos, añadimos al nombre del fichero la información de las covariantes a través del campo `short_name` a partir de `targets.txt`.

```
 [1] "GSM944836_inf_lin" "GSM944850_inf_lin" "GSM944857_inf_lin"
 [4] "GSM944864_inf_lin" "GSM944854_uni_lin" "GSM944847_uni_lin"
 [7] "GSM944840_uni_lin" "GSM944861_uni_lin" "GSM944835_inf_unt"
[10] "GSM944849_inf_unt" "GSM944863_inf_unt" "GSM944856_inf_unt"
[13] "GSM944845_uni_unt" "GSM944852_uni_unt" "GSM944859_uni_unt"
[16] "GSM944838_uni_unt" "GSM944865_inf_van" "GSM944837_inf_van"
[19] "GSM944851_inf_van" "GSM944858_inf_van" "GSM944841_uni_van"
[22] "GSM944855_uni_van" "GSM944862_uni_van" "GSM944848_uni_van"
```

A continuación, realizaremos una primera exploración de los datos de manera visual mediante técnicas univariantes como son gráfico de densidad, diagrama de cajas.

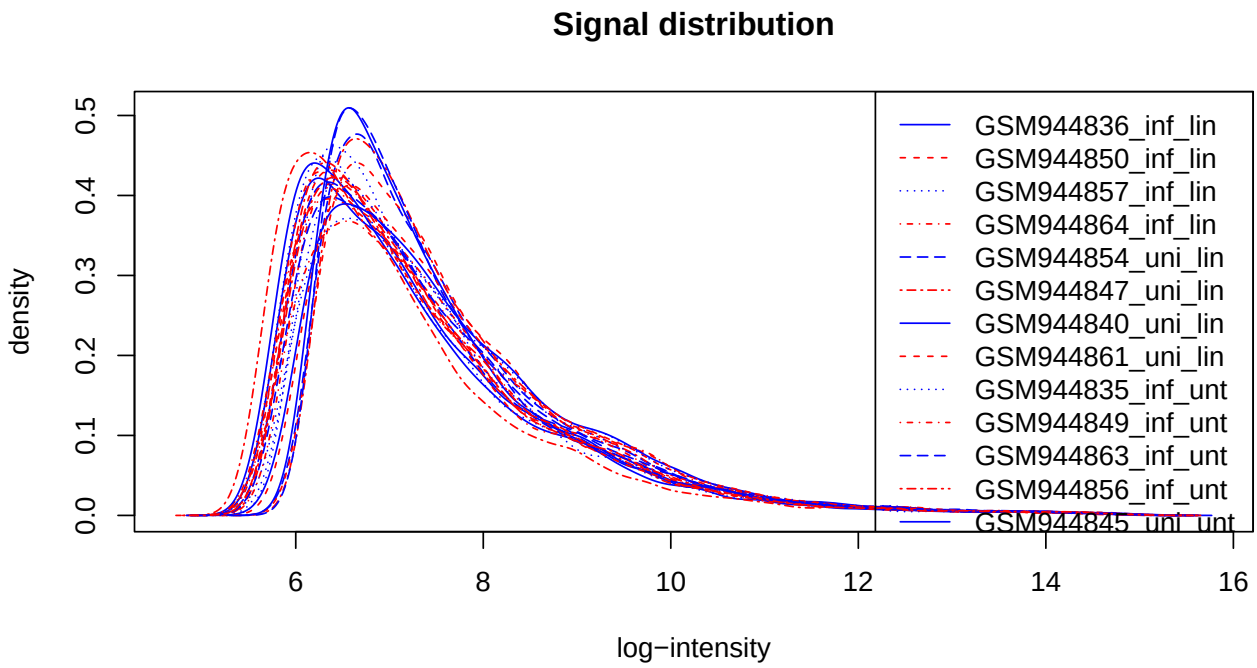


Figure 1: *Diagrama para visualizar la distribución de las expresiones de cada array para los datos crudos.*

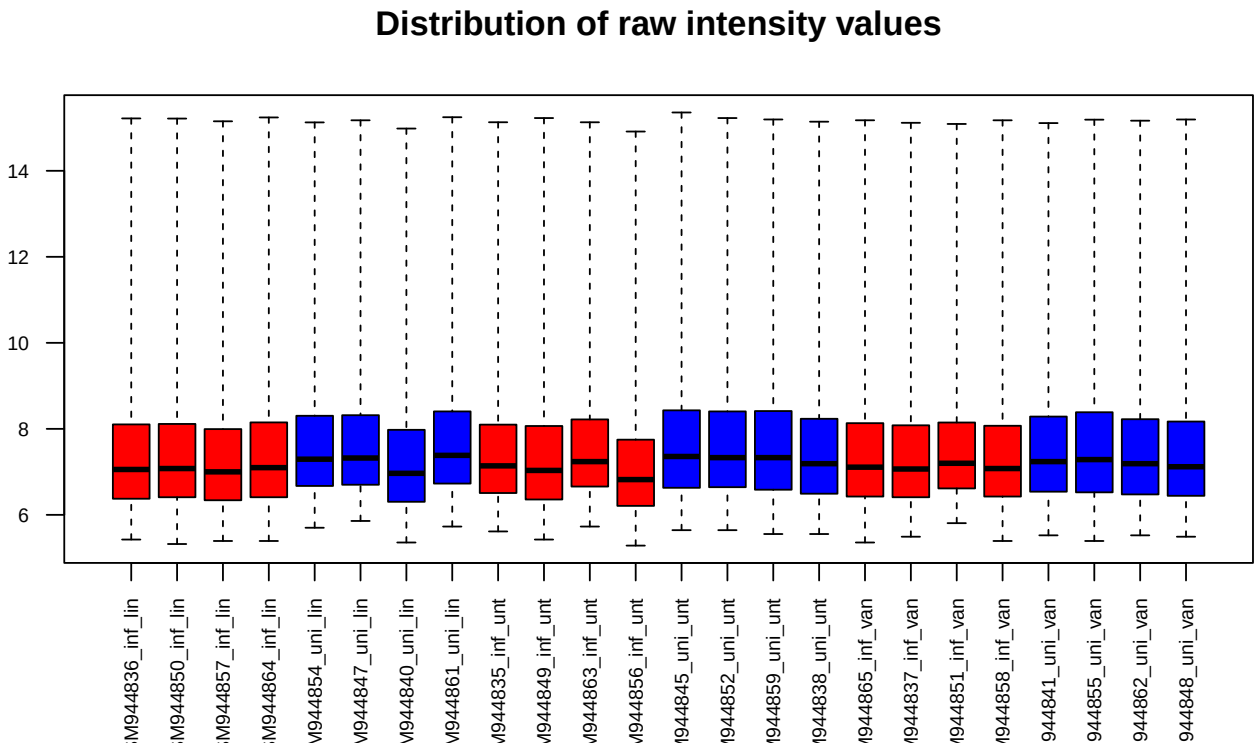


Figure 2: *Diagrama de cajas para los datos crudos de las expresiones de cada array.*

También utilizaremos técnicas multivariantes como son el análisis de conglomerados, `cluster` mediante un agrupamiento jerárquico seguido de un dendrograma, y realizaremos un análisis de distancias visualizando

después la matriz de distancias mediante un mapa de calores.[7]

```
manDist <-dist(t(exprs(rawData)))
```

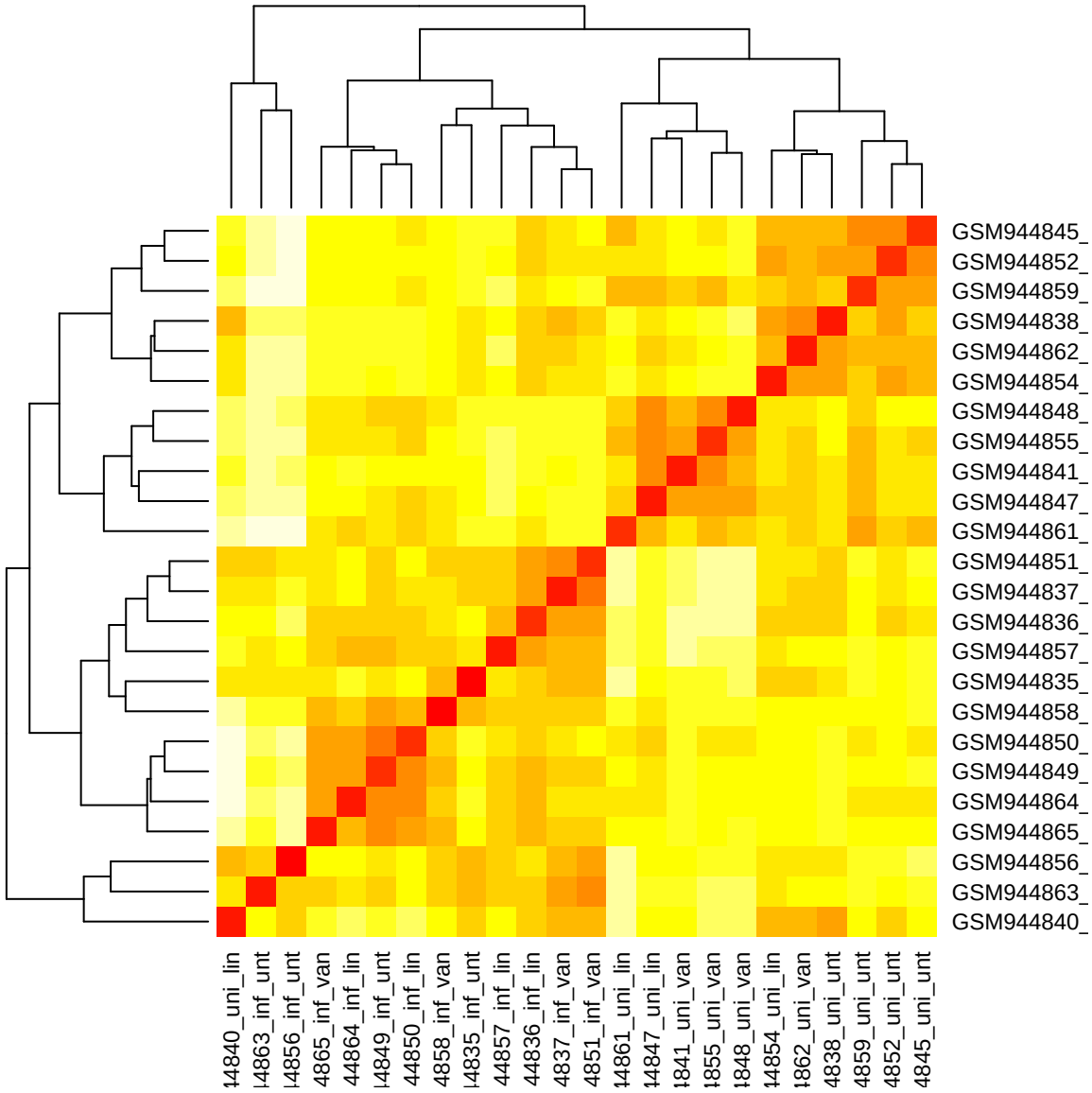


Figure 3: *Diagrama de calor de las distancias euclídeas entre los distintos arrays a partir de los datos crudos.*

[7]Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

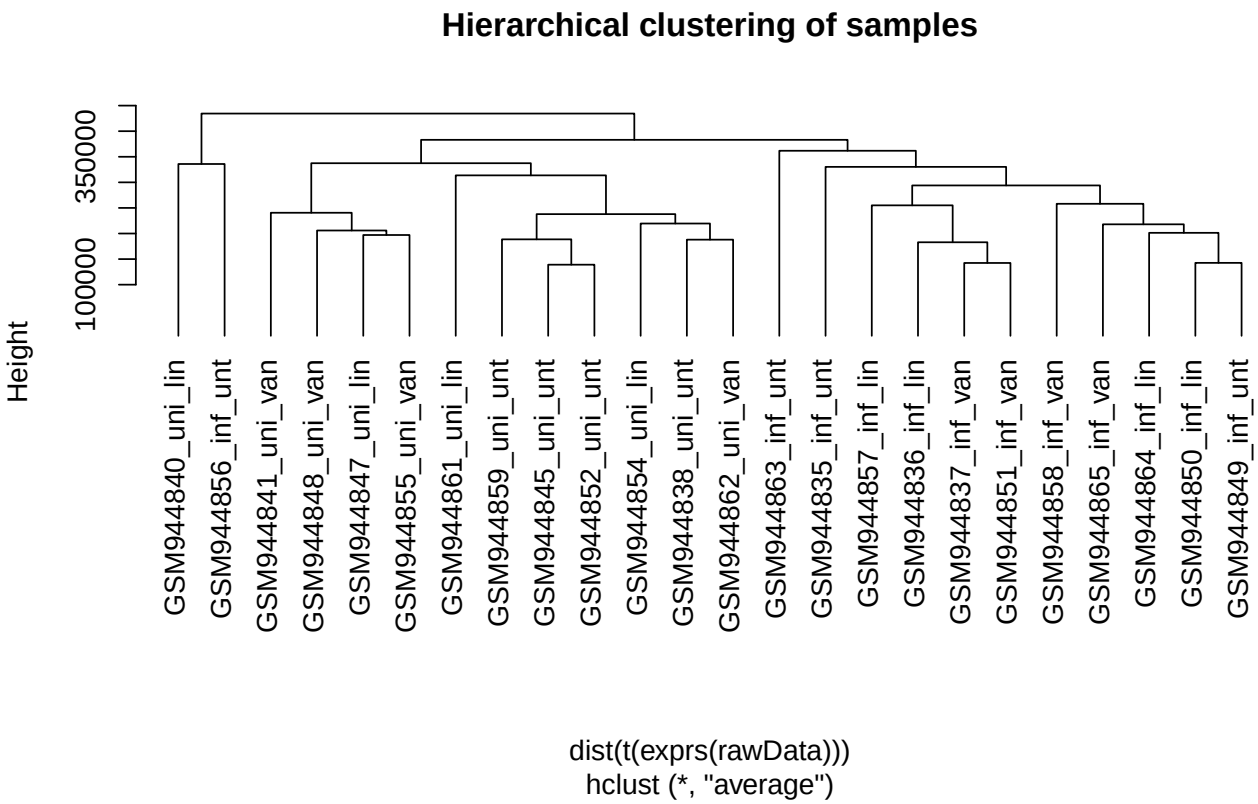


Figure 4: *Dendrograma resultante de un agrupamiento jerárquico entre las muestras a partir de los datos crudos.*

Para evaluar y detectar posibles efectos de lote, `batch`, realizaremos un análisis de los componentes principales PCA y mostraremos en un gráfico los componentes principales mostrando los datos en dimensión reducida de forma que cada componente explicará una dimensión de mayor variabilidad e independiente de la siguiente.[8]

La importancia de los grupos los representaremos a través de variabilidad explicada por cada componente. Según,[9] si la suma de los porcentajes es alta, superior por ejemplo al 50%, las conclusiones que de ellos se deriven serán más fiables que con valores bajos, por ejemplo inferiores al 30% de varianza explicada.

[8]Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".
[9]Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

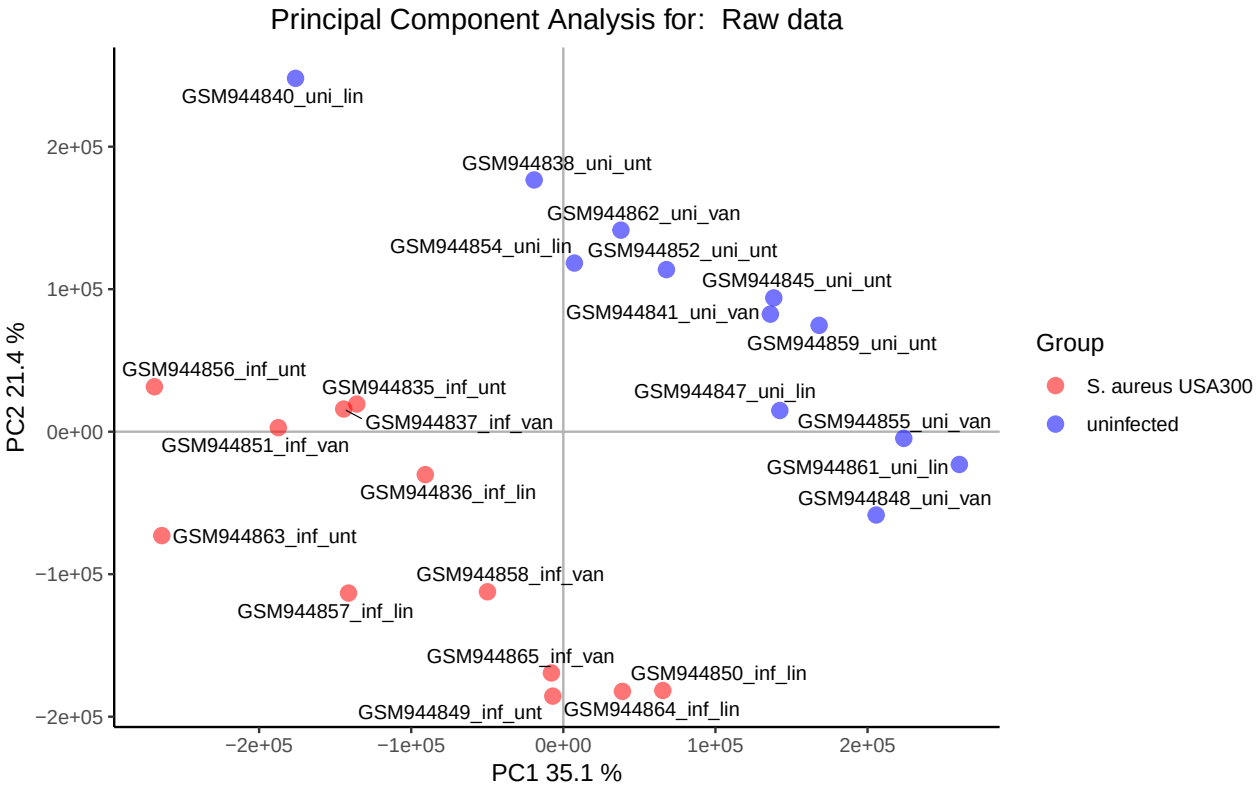


Figure 5: *Diagrama para visualizar los dos primeros componentes de los datos crudos.*

2.4.2 Control de calidad de los datos crudos

A continuación, realizaremos, un control de calidad con el método `arrayQualityMetrics()` del paquete `arrayQualityMetrics` específico para microarrays, que encapsula diferentes análisis generando un subdirectorio en el que guarda todas las salidas. Estas salidas están englobadas en un archivo `index.html`.

```
list.files(paste0(resultsDire,"arrayQuality_raw/"))
```

```
 [1] "arrayQualityMetrics.css" "arrayQualityMetrics.js"
 [3] "box.pdf"                 "box.png"
 [5] "dens.pdf"                "dens.svg"
 [7] "hm.pdf"                  "hm.png"
 [9] "index.html"              "ma.pdf"
[11] "ma.png"                  "msd.pdf"
[13] "msd.png"                 "out box.pdf"
[15] "out box.png"             "out hm.pdf"
[17] "out hm.png"              "out ma.pdf"
[19] "out ma.png"              "pca.pdf"
[21] "pca.svg"
```

A continuación, mostramos los valores (outliers) identificado con los diferentes métodos utilizados por la función `arrayQualityMetrics()`:

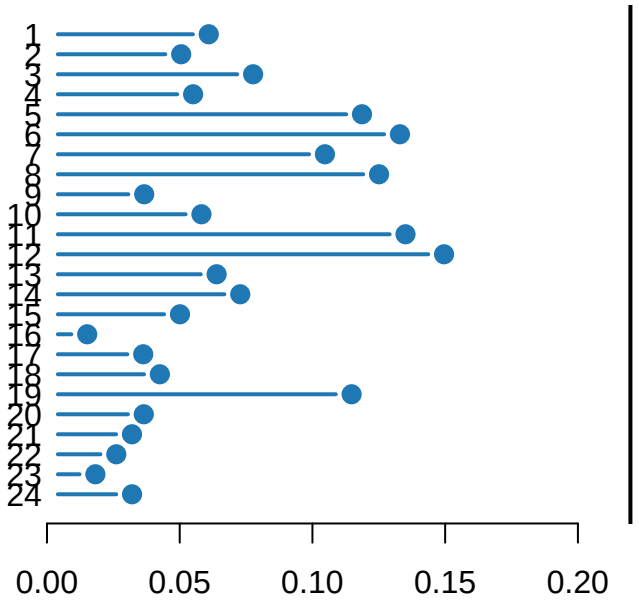


Figure 6: *Diagrama obtenido de 'arrayQualityMetrics()' para detectar outliers (Distancias entre arrays) a partir de los datos crudos.*

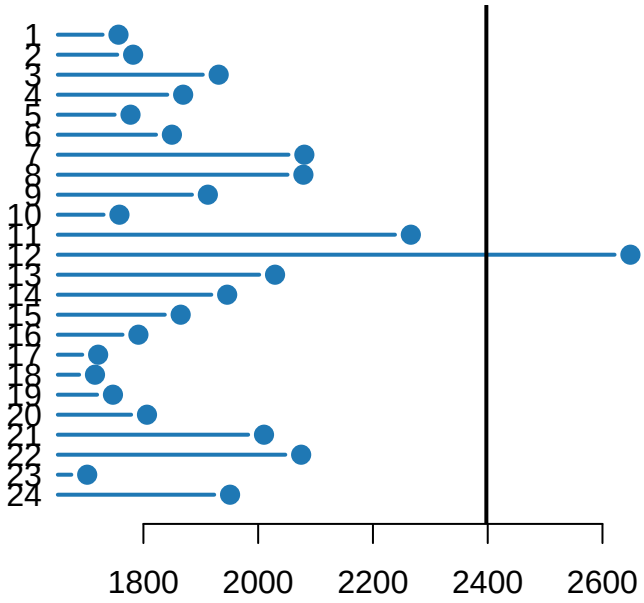


Figure 7: *Diagrama obtenido de 'arrayQualityMetrics()' para detectar outliers (Diagrama de cajas) a partir de los datos crudos.*

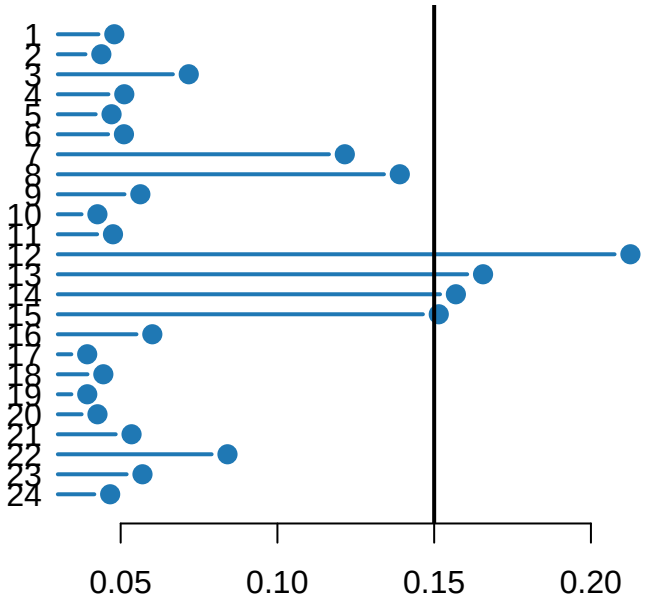


Figure 8: *Diagrama obtenido de 'arrayQualityMetrics()' para detectar outliers (Diagrma MA) a partir de los datos crudos.*

2.4.3 Normalización

Tras una primera exploración y control de calidad, el siguiente paso que haremos será normalizar los datos. Los datos se pueden normalizar mediante diferente herramientas, no obstante, en el presente estudio utilizaremos el método `rma()`.

Este paso es necesario para que cuando hagamos el análisis de los genes diferencialmente expresados, estos sean comparables entre sí e intentar reducir, y si es posible eliminar, toda la variabilidad en las muestras que no se deba a razones biológicas.[10]

Es decir, mediante el proceso de normalización lo que queremos es eliminar sesgos artificiales debidos a cuestiones técnicas.[11] De esta manera aseguraremos que en la matriz se refleja la expresión diferencial de los genes.[12]

Cabe destacar que el proceso de normalización mediante RMA se compone de tres etapas:[13]

- Corrección de fondo
- Normalización para hacer los valores de los arrays comparables.
- Summarización de las diversas sondas asociadas a cada grupo de sondas para dar un único valor.

```
eset_rma <- rma(rawData)
```

```
Background correcting
Normalizing
Calculating Expression
```

[10] Gonzalo Sanz and Sánchez-Pla, "Statistical Analysis of Microarray data".
[11] Gonzalo Sanz and Sánchez-Pla, "Statistical Analysis of Microarray data".
[12] Gonzalo Sanz and Sánchez-Pla, "Statistical Analysis of Microarray data".
[13] Gonzalo Sanz and Sánchez-Pla, "Statistical Analysis of Microarray data".

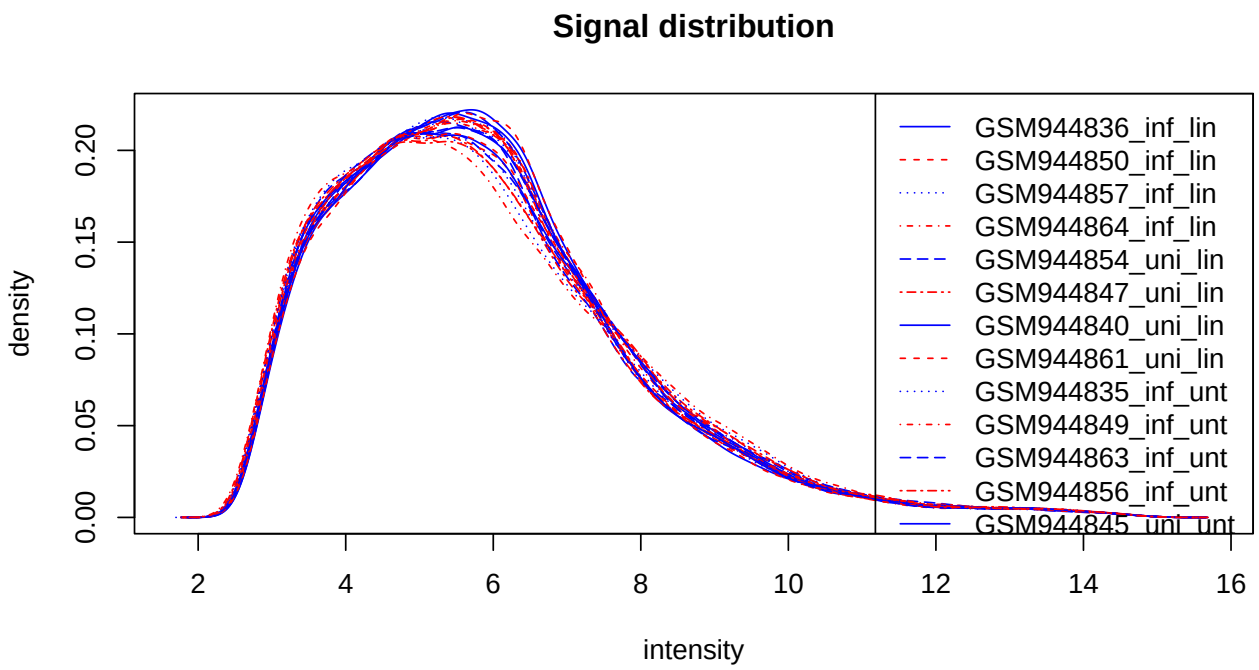


Figure 9: *Diagrama para visualizar la distribución de las expresiones de cada array para los datos normalizado*

A continuación, visualizaremos en un diagrama de cajas con los datos normalizados,

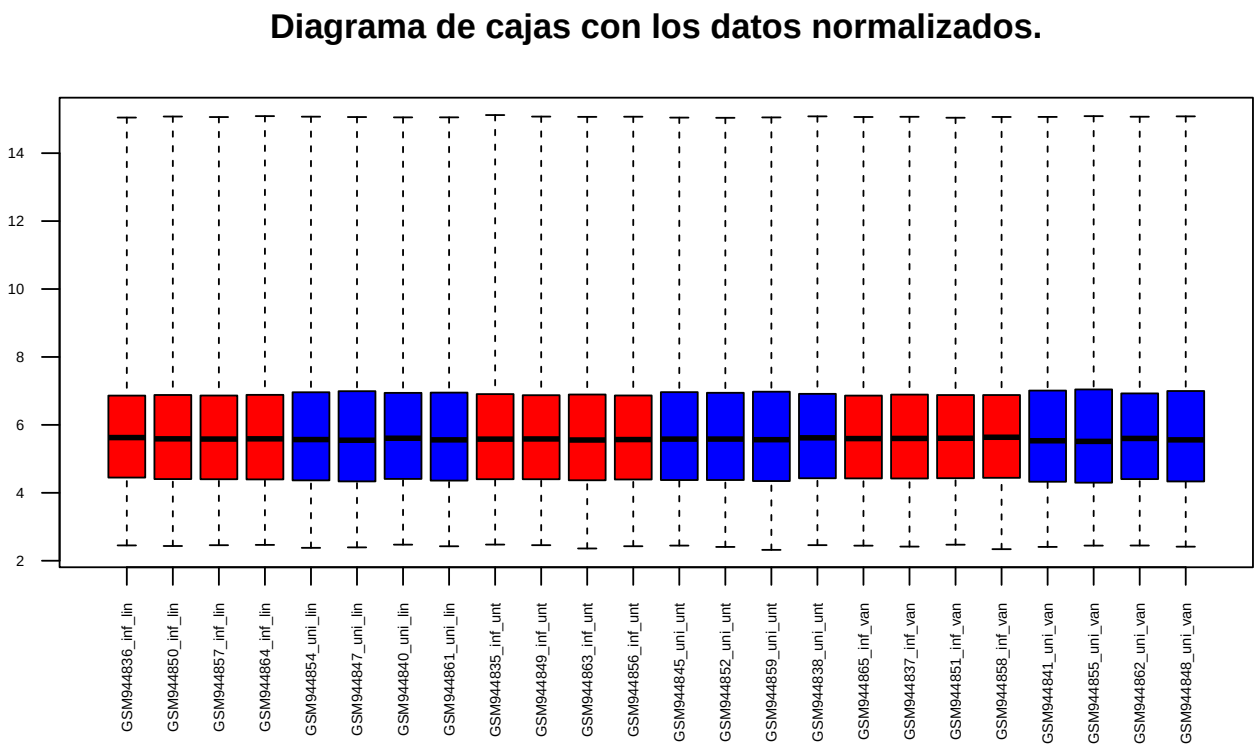


Figure 10: *Diagrama de cajas los datos normalizados.*

Ahora observamos que todos los diagramas de caja tienen el mismo aspecto, están en una escala común en la que se pueden comparar. Por tanto, podemos decir que la normalización ha funcionado correctamente.

2.4.4 Control de calidad de los datos normalizados

Después de normalizar los datos es interesante volver a realizar un control de calidad de los datos para comprobar cómo se ven los datos.[14]

```
list.files(paste0(resultsDire,"arrayQuality_raw/"))
```

```
 [1] "arrayQualityMetrics.css" "arrayQualityMetrics.js"
 [3] "box.pdf"                 "box.png"
 [5] "dens.pdf"                "dens.svg"
 [7] "hm.pdf"                  "hm.png"
 [9] "index.html"              "ma.pdf"
[11] "ma.png"                  "msd.pdf"
[13] "msd.png"                 "out box.pdf"
[15] "out box.png"             "out hm.pdf"
[17] "out hm.png"              "out ma.pdf"
[19] "out ma.png"              "pca.pdf"
[21] "pca.svg"
```

A continuación, mostramos los valores (outliers) identificado con los diferentes métodos utilizados por la función `arrayQualityMetrics()` después de normalizar los datos con la función `rma()`:

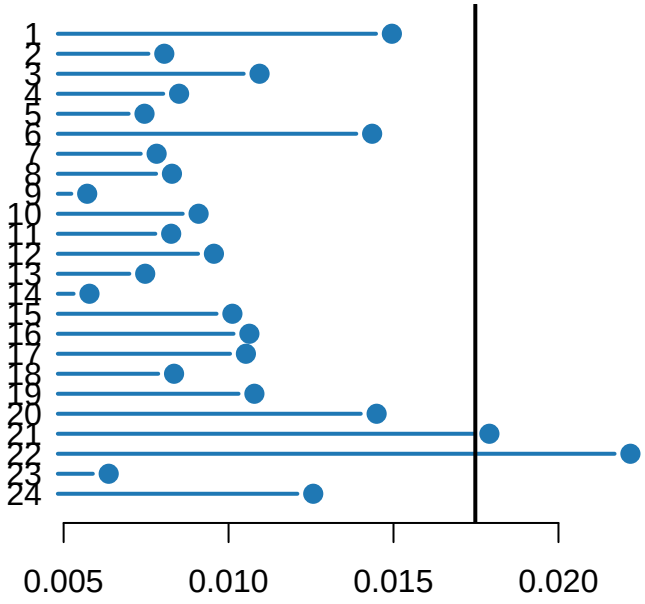


Figure 11: *Diagrama obtenido de 'arrayQualityMetrics()' para detectar outliers (Distancias entre arrays) a partir de los datos normalizados.*

[14] Gonzalo Sanz and Sánchez-Pla, "Statistical Analysis of Microarray data".

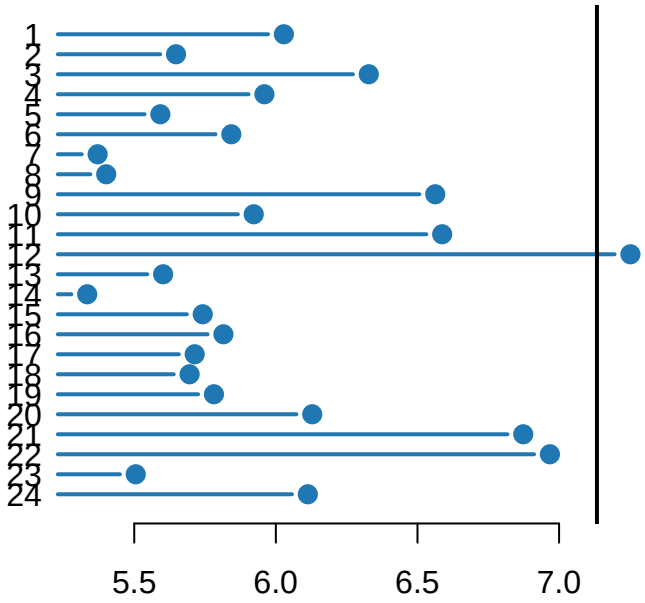


Figure 12: *Diagrama obtenido de 'arrayQualityMetrics()' para detectar outliers (Diagrama de cajas) a partir de los datos normalizados.*

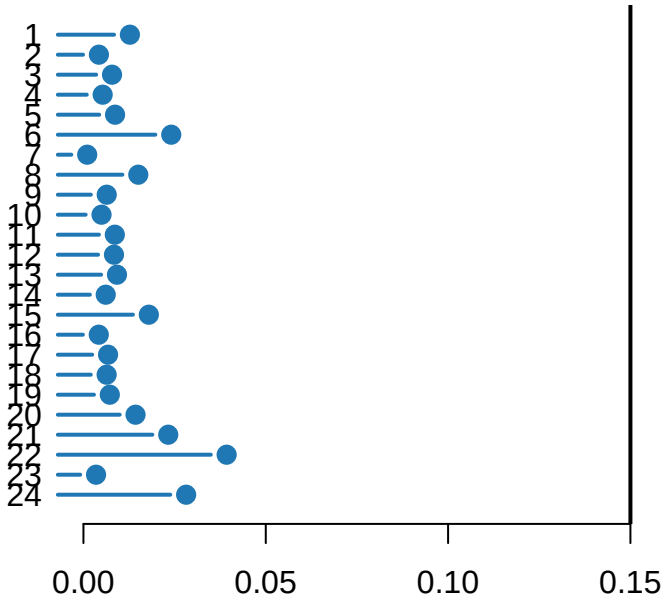


Figure 13: *Diagrama obtenido de 'arrayQualityMetrics()' para detectar outliers (Diagrama MA) a partir de los datos normalizados.*

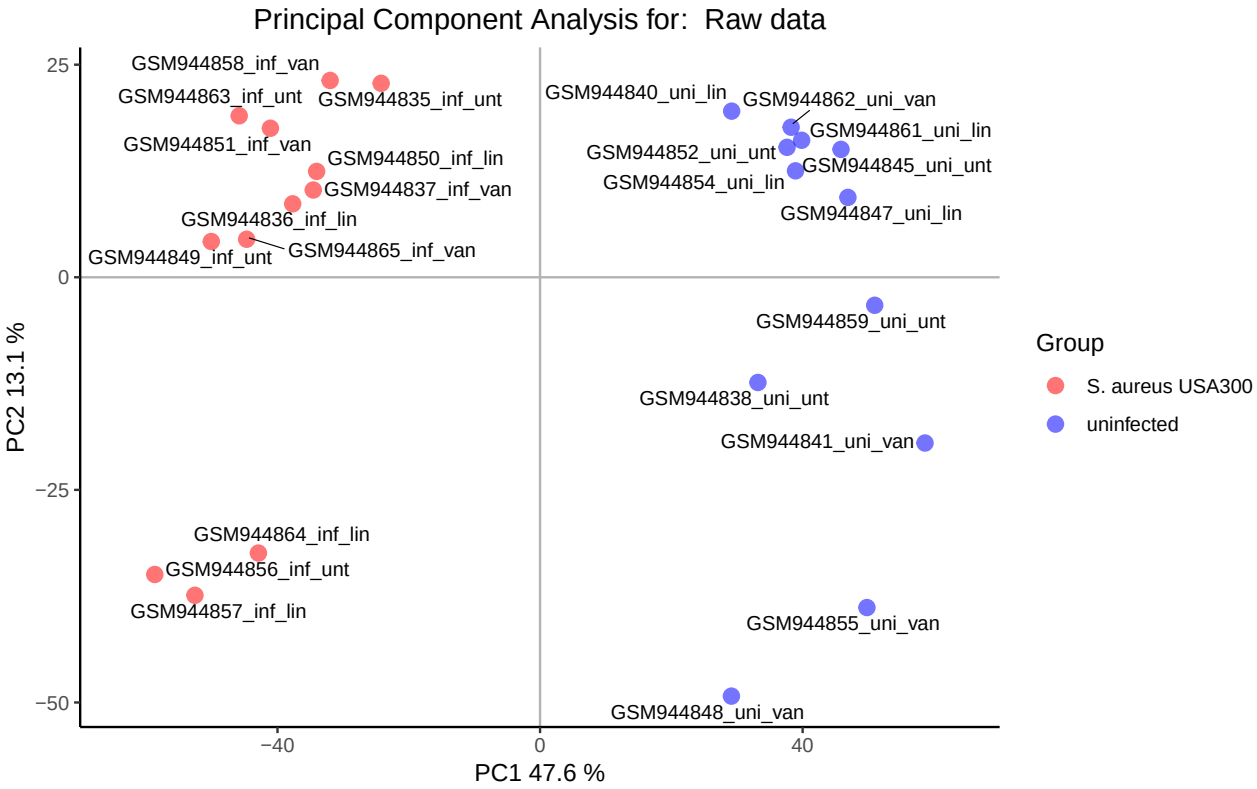


Figure 14: *Diagrama para visualizar los dos primeros componentes de los datos normalizados.*

2.4.5 Detección del efecto lote

Para analizar el efecto lote utilizamos la función `pvcaBatchAssess` del paquete `pvca`.

Según[15] el error acumulativo introducido por variaciones experimentales por causas técnicas (herramienta, técnico, ...) dependientes del tiempo y el lugar se conoce como "efectos de lote".

En el ámbito de los datos de microarrays, existen diferentes enfoques para identificar la existencia o no de este tipo de efecto. En el presente análisis, siguiendo los ejemplos visto,[16] utilizaremos el `PVCA` (Combat and Principal variation component analysis'). Esta herramienta, estima la fuente y la proporción de variación en dos pasos, análisis de componentes principales y análisis de componentes de varianza.

Comprobaremos también si todas las muestras se procesaron el mismo día mediante la función `get.celfile.dates()` del paquete `affyio`,

```
unique(get.celfile.dates(filenames=file.path(celfilesDire,fileNames)))
```

```
[1] "2010-11-10"
```

vemos que no es necesario considerar un factor de lote típico como la "Fecha de procesamiento".

En la siguiente visualización se muestra un diagrama de barras con una barra por cada fuente de variación incluida en el análisis. El tamaño de cada barra indica el porcentaje de variabilidad atribuible a cada fuente.[17] En nuestro caso, el gráfico indica que la principal fuente de variación en las muestras es la condición de `tiempo`. Esto se observó también de manera clara en los gráficos de PCA; tanto en los datos brutos, como en los normalizados.

[15] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".
[16] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".
[17] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

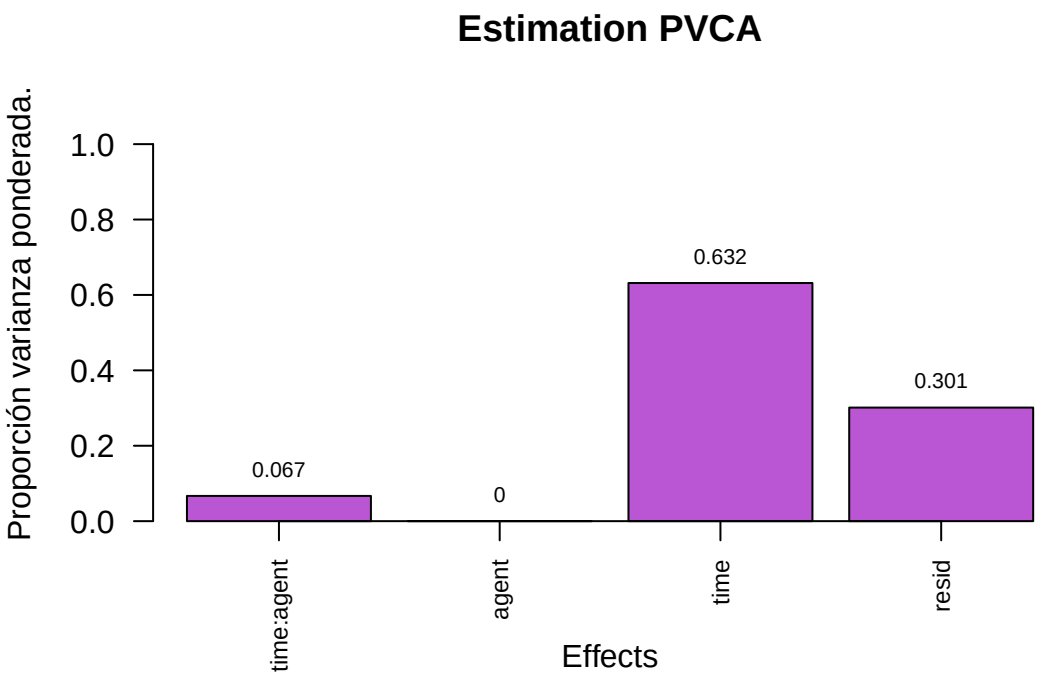


Figure 15: *Diagrama de barras que indica la importancia relativa de los diferentes factores que afectan a la expresión del gen:* `agent`, `time`, `resid`, `interacción time:agent`.

2.4.6 Filtrado de los datos

Según[18] el filtraje es algo discutido, no obstante en el enunciado de la PEC se sugiere la posibilidad de eliminar las sondas menos variables y quedarnos con el 10% de sondas que presenten mayor variabilidad.

A continuación, detectaremos los genes con mayor variabilidad. Para ello, nos fijaremos en la variabilidad global de cada gen.

```
sds <- apply(exprs(eset_rma), 1, sd)
sdsO <- sort(sds)
```

En la siguiente visualización veremos la variabilidad de todos los genes, y esperamos diferenciar los genes que varían más de los que no lo hacen. Se mostrarán los valores de desviación estándar para todas las muestras para todos los genes, ordenados de menor a mayor.

[18] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

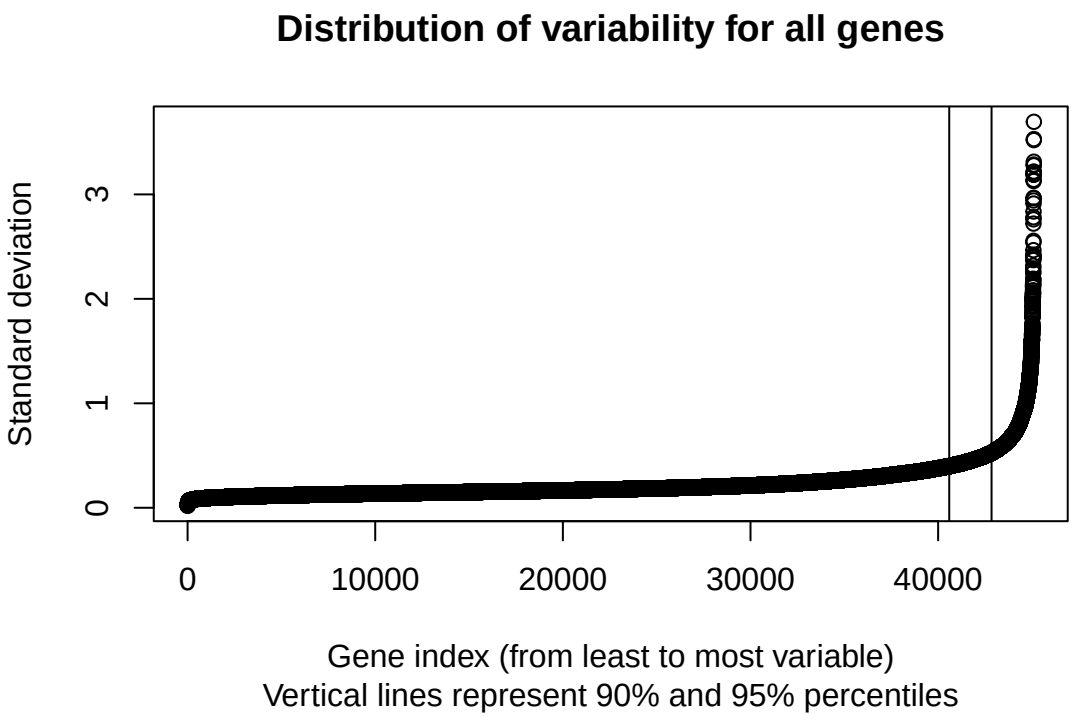


Figure 16: *Diagrama que muestra los valores desviaciones estándar a lo largo de todas las muestras para todos los genes ordenados de menor a mayor.*

Para realizar el filtrado de aquellos genes con menor variabilidad, aquellos genes cuya variabilidad puede atribuirse a una variación aleatoria, utilizaremos la función `nsFilter()` de la librería `genefilter`.[19] Razonablemente, se espera que estos genes no se expresen de forma diferencial.[20]

Antes de realizar el filtrado asociaremos sobre al `ExpressionSet` una serie de anotaciones mediante el conjunto de datos `mouse4302.db`.

```
annotation(eset_rma) <- "mouse4302.db"
```

Este paquete de anotaciones disponible lo utilizaremos para eliminar aquellos conjuntos de sondas que no tengan un identificador de gen asociado. Además de este criterio de filtrado utilizaremos el parámetro `var.cutoff` de la función `nsFilter()` para establecer un umbral de variabilidad.

```
data_nsFilter <- nsFilter(eset_rma,
                          require.entrez=TRUE,
                          remove.dupEntrez=TRUE,
                          var.func=IQR,
                          var.cutoff=0.9,
                          filterByQuantile=TRUE,
                          var.filter=TRUE,
                          feature.exclude="^AFFX"
)
```

la función `nsFilter()` devuelve dos objetos: un `expressionSet` y otro de log:

```
[1] "eset"       "filter.log"
```

con el objeto `filter.log` podemos ver el resultados del filtrado:

[19] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".
[20] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".

```
$numDupsRemoved
[1] 16958

$numLowVar
[1] 18432

$numRemoved.ENTREZID
[1] 7650

$feature.exclude
[1] 13
```

Almacenaremos los genes que sobreviven al filtrado en la variable `eset_rma_filt`.

```
data_eset_rma_filt <- data_nsFilter$eset
```

Después de realizar el filtrado vemos la evolución en cuanto a la dimensión del conjunto de datos.

Table 2: *Conjuntos de datos.*

dataset	Features	Samples
rawData	1004004	24
eset_rma	45101	24
data_eset_rma_filt	2048	24

Guardamos los datos crudos, normalizados y los filtrados en `resultsDire`.

```
list.files(paste0(resultsDire))
```

```
[1] "arrayQuality_norm"            "arrayQuality_raw"
[3] "data_normalized_filtered.csv" "data_normalized.csv"
[5] "data_normalized.Rda"          "data_raw.csv"
```

2.5 Genes diferencialmente expresados para cada comparación

Utilizaremos modelos lineales para microarrays para obtener una lista de genes diferencialmente expresados. El primer paso para el análisis basado en modelos lineales es crear la matriz de diseño[21] y después crearemos la matriz de contrastes y utilizaremos ambas matriz para realizar las comparaciones propuestas en el enunciado:

- Infectados vs no infectados sin tratamiento
- Infectados vs no infectados tratados con LINEZOLID
- Infectados vs no infectados tratados con VANCOMICINA

Es decir, nuestro objetivo será intentar caracterizar, a través del cambio en la expresión génica, el efecto de la infección y del tratamiento con antibióticos así como comparar los efectos de éstos.

2.5.1 Construcción de las matrices de diseño y de contrastes

La matriz de diseño es una tabla que describe la asignación de cada muestra (filas) a un grupo (columnas). Cada fila contiene un uno en la columna del grupo al que pertenece la muestra y un cero en las restantes.[22]

Definimos con la función `model.matrix()` la matriz de diseño para el análisis que estamos realizando de la siguiente manera:

[21] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".
[22] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

Table 3: *Matriz de diseño.*

inf_lin	inf_unt	inf_van	uni_lin	uni_unt	uni_van
1	0	0	0	0	0
1	0	0	0	0	0
1	0	0	0	0	0
1	0	0	0	0	0
0	0	0	1	0	0
0	0	0	1	0	0
0	0	0	1	0	0
0	0	0	1	0	0
0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	0
0	0	0	0	1	0
0	0	0	0	1	0
0	0	0	0	1	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	1

Por otro lado, la matriz de contrastes describe las comparaciones entre grupos.[23] Consta de tantas columnas como comparaciones y tantas filas como grupos. A partir del enunciado, mediante la función `makeContrasts()`, definiremos a continuación la matriz de contrastes:

Table 4: *Matriz de contrastes para las tres comparaciones.*

inf_vs_uni.unt	inf_vs_uni.lin	inf_vs_uni.van
0	1	0
1	0	0
0	0	1
0	-1	0
-1	0	0
0	0	-1

Mediante esta matriz de contrastes recogeremos lo siguiente:

- Infectados vs no infectados sin tratamiento: analizar el efecto del no tratamiento sobre los infectados y no infectados.
- Infectados vs no infectados tratados con LINEZOLID: analizar el efecto del tratamiento `lin` sobre los infectados y no infectados.

[23] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

- Infectados vs no infectados tratados con VANCOMICINA: analizar el efecto del tratamiento `van` sobre los infectados y no infectados.

2.5.2 Estimación del modelo y selección de genes

Como hemos mencionado al inicio de presente apartado, utilizaremos el paquete `limma` para obtener una lista de genes diferencialmente expresados. Para estimar el modelo lineal utilizaremos la función `lmFit()`. Tras ajustar el modelo estimaremos los contrastes y realizaremos las pruebas de significación que nos llevarán a decidir, para cada gen y cada comparación, si éstos pueden considerarse diferencialmente expresados.[24]

```
fit <- lmFit(data_eset_rma_filt, designMat)
fit.main <- contrasts.fit(fit, cont.matrix)
fit.main <- eBayes(fit.main)
```

2.5.3 Obtención de las listas de genes diferencialmente expresados

Los resultados de un análisis de expresión diferencial se pueden extraer con la función `topTable`.[25] Esta función para cada contraste obtiene una tabla de genes ordenados por el p-value de menor a mayor, donde las columnas contienen información acerca de los genes y la diferencia entre los grupos comparados.[26]

A continuación realizaremos el análisis de expresión diferencial para cada comparación que hemos definido.

- inf_vs_uni.unt: genes que cambian su expresión entre `infeccion` y `no infeccion` `sin tratamiento`:

En la siguiente tabla mostramos los 10 genes más expresados diferencialmente en cada comparación `inf_vs_uni.unt`.

Table 5: *Resultado 'topTable()' para inf vs uni unt: genes que cambian su expresión entre 'infeccion' y 'no infeccion' 'sin tratamiento'.*

PROBEID	logFC	AveExpr	t	P.Value	adj.P.Val	B
1422953_at	3.782913	11.285986	19.72971	0	0	24.46709
1421262_at	7.338091	7.116261	19.41303	0	0	24.15921
1427747_a_at	6.253967	10.714264	18.36380	0	0	23.10007
1422438_at	-2.155588	7.557760	-15.33730	0	0	19.66337
1440865_at	4.278966	11.151848	15.20126	0	0	19.49385
1418722_at	6.091476	10.963898	14.84044	0	0	19.03725
1447806_s_at	-3.262810	7.890583	-14.47527	0	0	18.56458
1448213_at	3.329679	12.053344	14.47496	0	0	18.56418
1419681_a_at	5.312061	7.123546	14.29327	0	0	18.32493
1417290_at	4.612120	10.127408	14.06103	0	0	18.01507

- inf_vs_uni.lin: genes que cambian su expresión entre `infeccion` y `no infeccion` con `tratamiento lin`.

En la siguiente tabla mostramos los 10 genes más expresados diferencialmente en cada comparación `inf_vs_uni.lin`.

[24] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".
[25] Mireia Ferrer, "Workflow de análisis de expresión diferencial usando RNA-seq".
[26] Mireia Ferrer, "Workflow de análisis de expresión diferencial usando RNA-seq".

Table 6: *Resultado 'topTable()' para inf vs uni lin: genes que cambian su expresión entre 'infeccion' y 'no infeccion' con 'tratamiento' 'lin'.*

PROBEID	logFC	AveExpr	t	P.Value	adj.P.Val	B
1421262_at	6.226482	7.116261	16.47225	0	0	20.91076
1422953_at	2.734360	11.285986	14.26100	0	0	18.24115
1427747_a_at	4.750369	10.714264	13.94872	0	0	17.83218
1448562_at	4.287875	7.708944	13.01957	0	0	16.56341
1419681_a_at	4.809975	7.123546	12.94230	0	0	16.45424
1422438_at	-1.780585	7.557760	-12.66911	0	0	16.06361
1440865_at	3.499990	11.151848	12.43391	0	0	15.72136
1455132_at	-1.658940	8.445168	-11.64557	0	0	14.53243
1427102_at	3.614436	10.661573	11.58749	0	0	14.44221
1436530_at	4.368038	10.011413	11.50540	0	0	14.31406

- inf_vs_uni.van: genes que cambian su expresión entre `infeccion` y `no infeccion` con `tratamiento van`.

En la siguiente tabla mostramos los 10 genes más expresados diferencialmente en cada comparación `inf_vs_uni.van`.

Table 7: *Resultado 'topTable()' para inf vs uni van: genes que cambian su expresión entre 'infeccion' y 'no infeccion' con 'tratamiento' 'van'.*

PROBEID	logFC	AveExpr	t	P.Value	adj.P.Val	B
1421262_at	6.866506	7.116261	18.16545	0	0	22.82459
1427747_a_at	5.317716	10.714264	15.61464	0	0	19.97618
1419681_a_at	5.556654	7.123546	14.95140	0	0	19.15990
1422953_at	2.848534	11.285986	14.85648	0	0	19.04026
1422438_at	-1.981556	7.557760	-14.09904	0	0	18.05910
1416111_at	-2.356253	7.363186	-14.02464	0	0	17.96012
1447806_s_at	-2.967954	7.890583	-13.16715	0	0	16.78371
1418722_at	5.307705	10.963898	12.93097	0	0	16.44776
1440865_at	3.636685	11.151848	12.91953	0	0	16.43135
1448788_at	-1.978253	6.304205	-12.68318	0	0	16.08954

2.6 Post-procesado de las listas de genes

Después de realizar el análisis de cuales son los genes diferencialmente expresados para cada comparación realizaremos un post-procesado que resulte de ayuda para la interpretación de los resultados. Por ello, realizaremos:

- Comparación entre listas para determinar qué genes cambian simultáneamente o no en las diferentes comparaciones.
- Anotación sobre la lista de genes mediante la información publicada existente en bases de datos.
- Visualización de los genes seleccionados en las distintas comparaciones.
- Análisis de significación biológica de las listas mediante el análisis de enriquecimiento y/o análisis `gene set analysis`.

2.6.1 Comparación múltiple

Cuando se realizan varias comparaciones a la vez puede resultar importante ver qué genes cambian simultáneamente en más de una comparación. Para ello, utilizaremos la función `decideTests()` para realizar las

comparaciones entre las listas de genes.[27]

Realizamos la comparación múltiple con la función `decideTests()` con los siguientes parámetros: `adjust.method="fdr"`, `p.value=0.01`, `lfc=1`. Se seleccionaran los genes que cambian en una o más condiciones.[28] Según[29] este criterio combina a la vez la significación estadística y la significación biológica.

Para cada gen y comparación el resultado del análisis será un 1 si el gen esta sobrexpresado, un 0 si no hay cambio significativo, o un -1 si está infraregulado.

```
res <- decideTests(fit.main, method="separate", adjust.method="fdr", p.value=0.01, lfc=1)
```

A partir de este resultado podemos contar qué filas tienen como mínimo una celda distinta de cero y visualizar los resultados

```
sum.res.rows <- apply(abs(res),1,sum)
res.selected <- res[sum.res.rows!=0,]
```

Table 8: *Comparación múltiple con la función 'decideTests()' para cada gen y comparación el resultado del análisis.*

Var1	Var2	Freq
Down	inf_vs_uni.unt	504
NotSig	inf_vs_uni.unt	1077
Up	inf_vs_uni.unt	467
Down	inf_vs_uni.lin	552
NotSig	inf_vs_uni.lin	1137
Up	inf_vs_uni.lin	359
Down	inf_vs_uni.van	692
NotSig	inf_vs_uni.van	1088
Up	inf_vs_uni.van	268

2.6.2 Anotación de los genes

Para ver las anotaciones disponibles que utilizaremos para asociar a los genes de nuestro estudio, descargaremos el paquete de `R mouse4302.db`.

```
keytypes(mouse4302.db)
```

```
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
 [6] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
[11] "GENETYPE"     "GO"           "GOALL"        "IPI"          "MGI"
[16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
[21] "PROBEID"      "PROSITE"      "REFSEQ"       "SYMBOL"       "UNIPROT"
```

```
columns(mouse4302.db)
```

```
 [1] "ACCNUM"       "ALIAS"        "ENSEMBL"      "ENSEMBLPROT"  "ENSEMBLTRANS"
 [6] "ENTREZID"     "ENZYME"       "EVIDENCE"     "EVIDENCEALL"  "GENENAME"
[11] "GENETYPE"     "GO"           "GOALL"        "IPI"          "MGI"
[16] "ONTOLOGY"     "ONTOLOGYALL"  "PATH"         "PFAM"         "PMID"
[21] "PROBEID"      "PROSITE"      "REFSEQ"       "SYMBOL"       "UNIPROT"
```

Como una primera aproximación de exploración de la base de datos `mouse4302.db` podemos utilizar la función `aafTableAnn()` del paquete `annafy` para generar y exportar una tabla con anotaciones e hipervínculos a la

[27] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".
[28] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".
[29] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

base de datos para cada anotación de cada gen ya seleccionado.

```
list.files(paste0(resultsDire),pattern="data_gen_selected_annoted_db_mouse4302")
```

```
[1] "data_gen_selected_annoted_db_mouse4302.html"
```

Crearemos y usaremos la función `annotatedTopTable()` para asociar las anotaciones de la base de datos a nuestro listado de genes, y añadir así los siguientes identificadores: `SYMBOL`, `ENTREZID`, `GENENAME` para cada comparación:[30]

- inf_vs_uni.unt: genes que cambian su expresión entre `infección` y `no infección sin tratamiento`:

```
topAnnotated_inf_vs_uni.unt <- annotatedTopTable(topTab_inf_vs_uni.unt,anotPackage="mouse4302.db")
```

Table 9: *Anotaciones para el resultado del análisis de expresión diferencial con 'topTable()' para inf vs uni unt: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'unt'.*

PROBEID	SYMBOL	logFC	AveExpr	t	P.Value	adj.P.Val	B
1415677_at	Dhrs1	0.7816	8.0560	4.6485	0.0001	0.0002	0.4750
1415682_at	Xpo7	-1.3052	10.2534	-4.4156	0.0002	0.0004	-0.0784
1415703_at	Huwe1	-0.6480	8.2439	-4.7815	0.0001	0.0002	0.7906
1415758_at	Fryl	-0.7660	8.6732	-6.6394	0.0000	0.0000	5.0749
1415775_at	Rbbp7	-0.7090	8.5715	-3.8691	0.0009	0.0013	-1.3704
1415794_a_at	Spin1	-0.5940	7.8707	-3.3550	0.0030	0.0040	-2.5595
1415802_at	Slc16a1	0.5565	9.8693	1.2706	0.2178	0.2359	-6.4404
1415822_at	Scd2	-0.3231	6.9529	-1.1130	0.2783	0.2986	-6.6237
1415837_at	Klk1	-0.1372	5.2207	-0.3030	0.7649	0.7820	-7.2032
1415839_a_at	Npm1	-0.7367	9.0096	-3.0589	0.0060	0.0076	-3.2220

- inf_vs_uni.lin: genes que cambian su expresión entre `infección` y `no infección` con `tratamiento lin`

```
topAnnotated_inf_vs_uni.lin <- annotatedTopTable(topTab_inf_vs_uni.lin,anotPackage="mouse4302.db")
```

Table 10: *Anotaciones para el resultado del análisis de expresión diferencial con 'topTable()' para inf vs uni lin: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'lin'.*

PROBEID	SYMBOL	logFC	AveExpr	t	P.Value	adj.P.Val	B
1415677_at	Dhrs1	0.6076	8.0560	3.6134	0.0016	0.0023	-1.7892
1415682_at	Xpo7	0.3208	10.2534	1.0851	0.2902	0.2988	-6.4619
1415703_at	Huwe1	-0.7775	8.2439	-5.7376	0.0000	0.0000	3.1884
1415758_at	Fryl	-0.8379	8.6732	-7.2626	0.0000	0.0000	6.5569
1415775_at	Rbbp7	-0.5905	8.5715	-3.2226	0.0041	0.0052	-2.6775
1415794_a_at	Spin1	-0.8315	7.8707	-4.6962	0.0001	0.0002	0.7550
1415802_at	Slc16a1	-0.4422	9.8693	-1.0095	0.3242	0.3322	-6.5401
1415822_at	Scd2	-1.0001	6.9529	-3.4446	0.0024	0.0032	-2.1763
1415837_at	Klk1	-1.1205	5.2207	-2.4742	0.0220	0.0251	-4.2669
1415839_a_at	Npm1	-0.4182	9.0096	-1.7365	0.0971	0.1033	-5.5931

- inf_vs_uni.van: genes que cambian su expresión entre `infección` y `no infección` con `tratamiento van`

[30] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

```
topAnnotated_inf_vs_uni.van <- annotatedTopTable(topTab_inf_vs_uni.van,anotPackage="mouse4302.db")
```

Table 11: *Anotaciones para el resultado del análisis de expresión diferencial con 'topTable()' para inf vs uni van: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'van'.*

PROBEID	SYMBOL	logFC	AveExpr	t	P.Value	adj.P.Val	B
1415677_at	Dhrs1	0.6367	8.0560	3.7867	0.0011	0.0015	-1.4774
1415682_at	Xpo7	0.8290	10.2534	2.8046	0.0106	0.0132	-3.6825
1415703_at	Huwe1	-0.9210	8.2439	-6.7961	0.0000	0.0000	5.4889
1415758_at	Fryl	-0.9497	8.6732	-8.2309	0.0000	0.0000	8.4830
1415775_at	Rbbp7	-0.9116	8.5715	-4.9752	0.0001	0.0001	1.3303
1415794_a_at	Spin1	-1.0922	7.8707	-6.1683	0.0000	0.0000	4.0942
1415802_at	Slc16a1	0.0245	9.8693	0.0560	0.9559	0.9573	-7.1556
1415822_at	Scd2	-0.6805	6.9529	-2.3438	0.0290	0.0342	-4.6184
1415837_at	Klk1	-1.2266	5.2207	-2.7085	0.0132	0.0162	-3.8847
1415839_a_at	Npm1	-1.1979	9.0096	-4.9741	0.0001	0.0001	1.3276

Una vez hayamos asociado a nuestras listas de genes las correspondientes anotaciones para cada una de las tres comparaciones guardaremos el conjunto de datos en `resultsDire`.

```
list.files(paste0(resultsDire),pattern="data_top_annotated")
```

```
[1] "data_top_annotated_inf_vs_uni_lin.csv"
[2] "data_top_annotated_inf_vs_uni_unt.csv"
[3] "data_top_annotated_inf_vs_uni_van.csv"
```

2.6.3 Visualización de los perfiles de expresión

Podemos visualizar la tabla `res.selected` obtenida anteriormente con la función `decideTests()` mediante el diagrama de `Venn`. Este diagrama representa el número de genes que se han denominado expresados diferencialmente en cada comparación con un valor de corte establecido (`p.value=0.01`, `lfc=1`). Podemos identificar mediante la representación cuantos genes son compartidos por una o más selecciones.

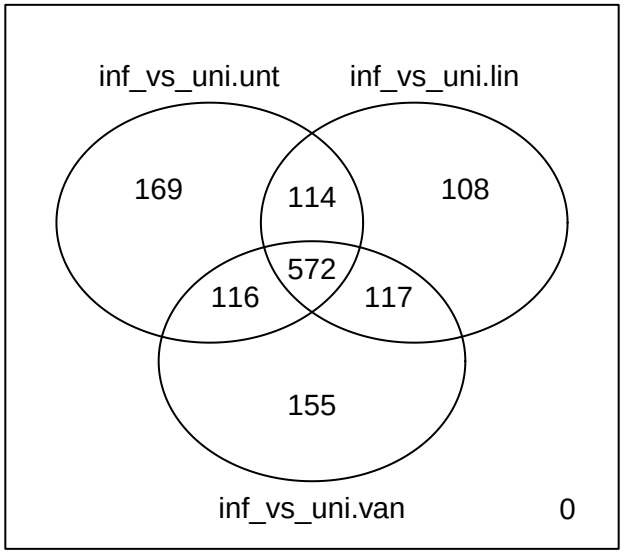


Figure 17: *Diagrama de Venn con los genes expresados diferencialmente en cada comparación.*

Mediante el `volcanoplot()` obtendremos una visualización de la expresión diferencial. Este tipo de figura representa aquellos genes que aparecen en los extremos superiores. El eje X representa los cambios de expresión, es decir el efecto biológico en escala logarítmica y el eje Y representa el efecto estadístico en escala logarítmica negativa.

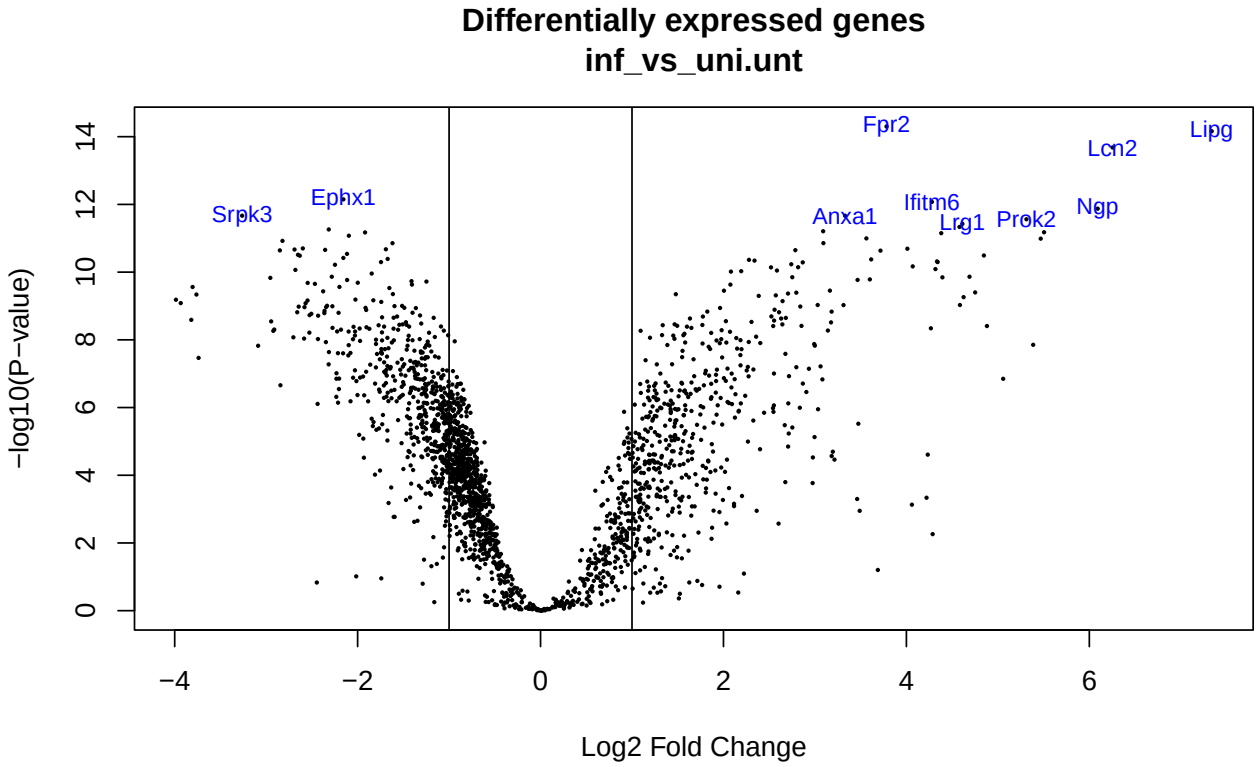


Figure 18: *Diagrama de Venn con los genes expresados diferencialmente en la primera comparación.*

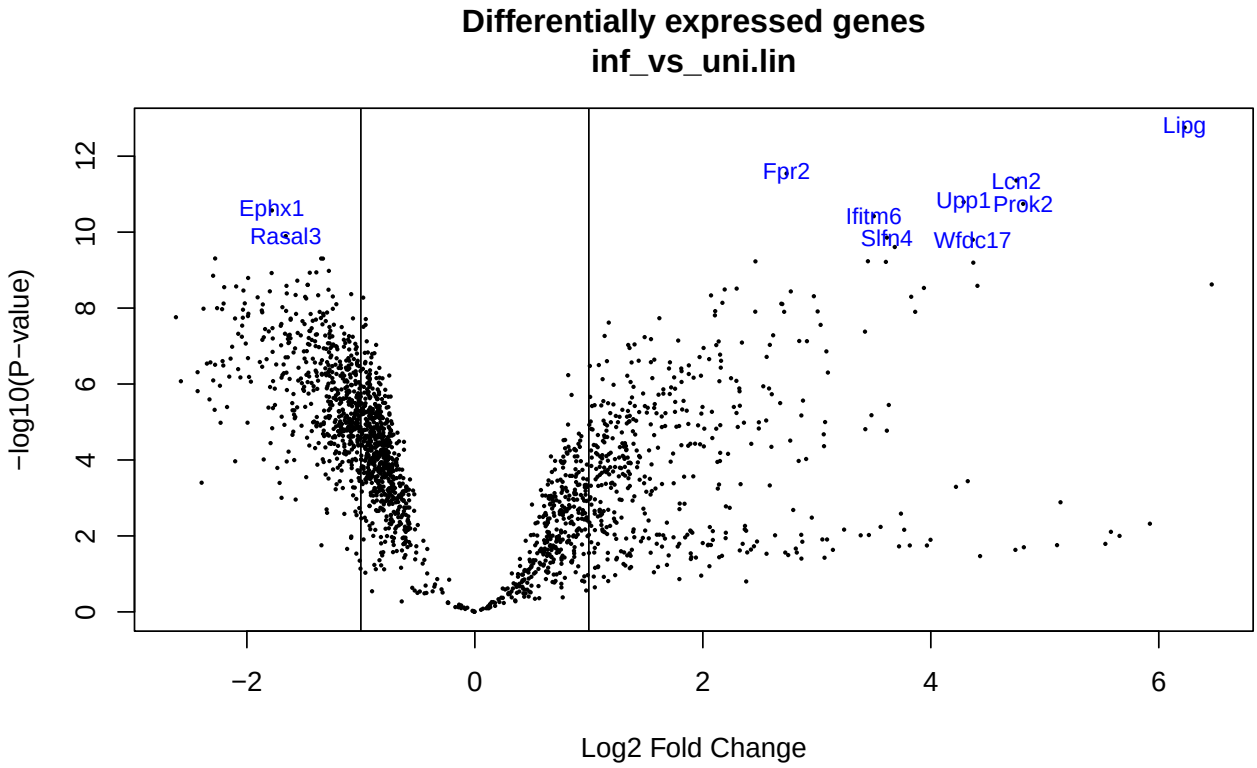


Figure 19: *Diagrama de Venn con los genes expresados diferencialmente en la segunda comparación.*

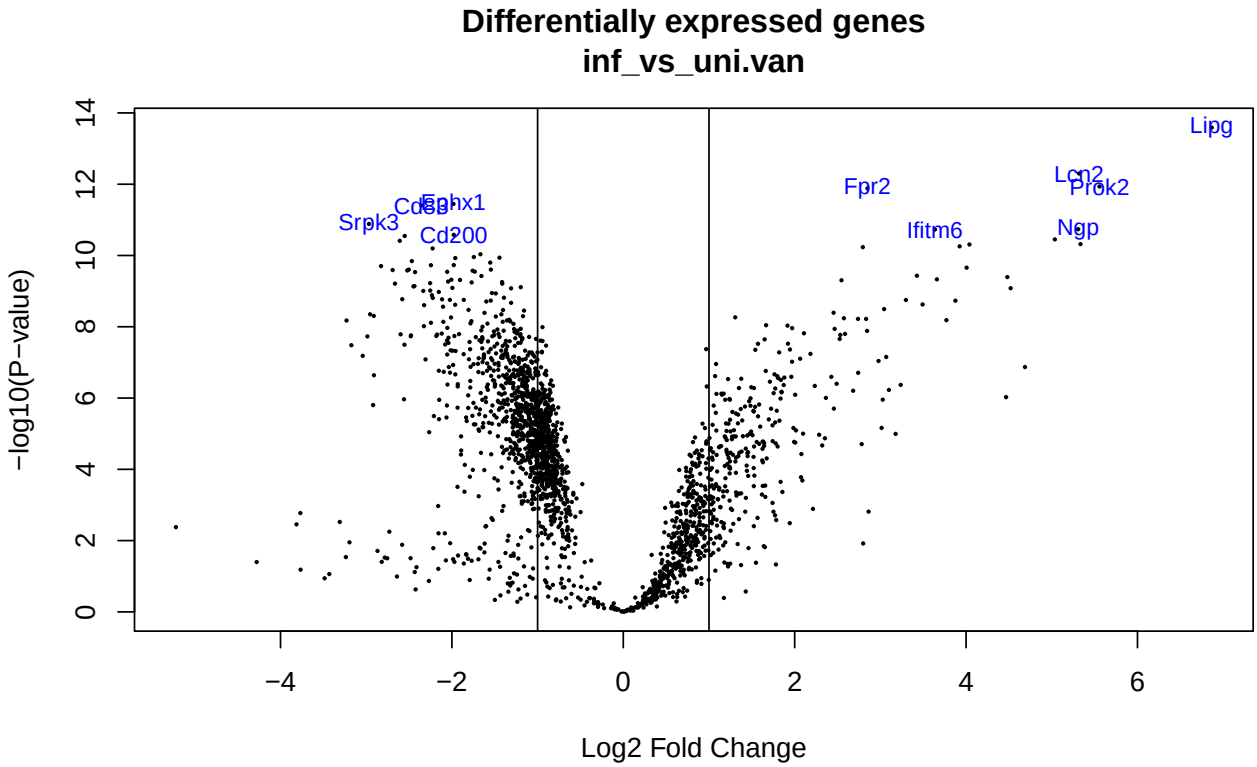
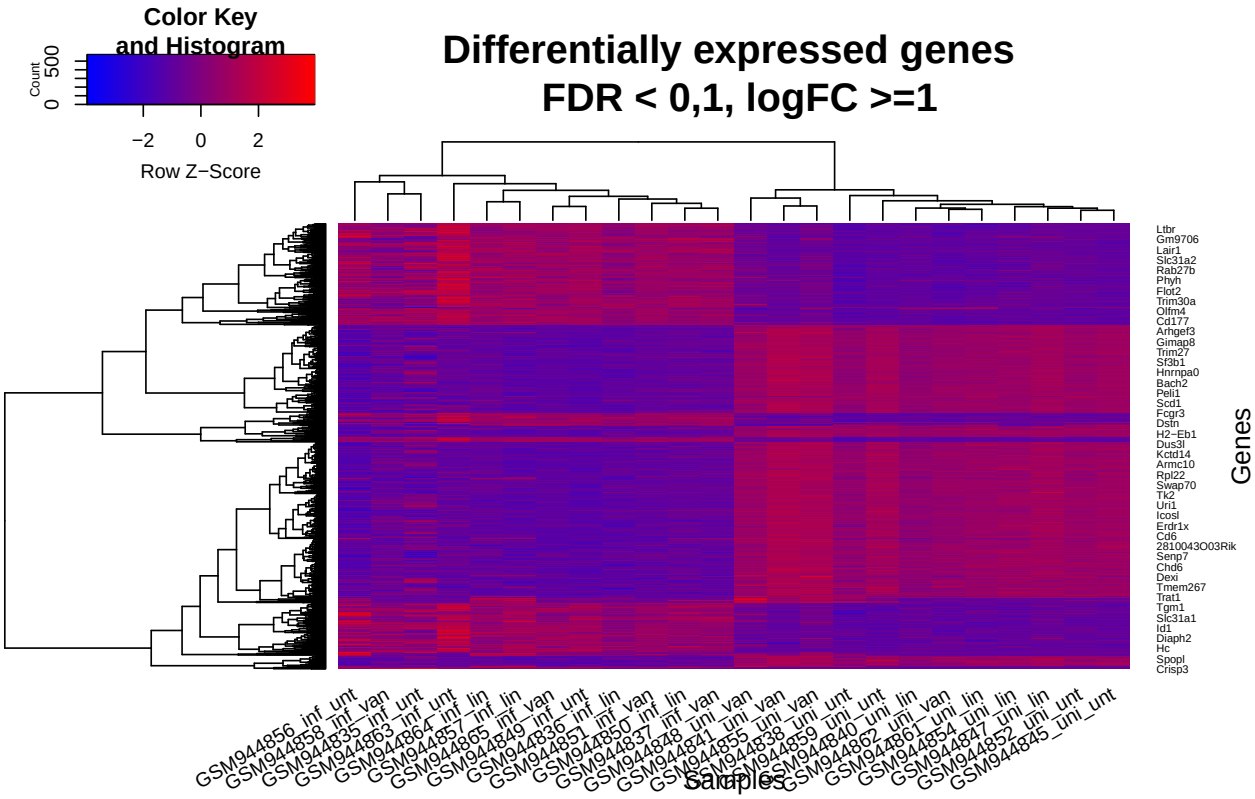


Figure 20: *Diagrama de Venn con los genes expresados diferencialmente en la tercera comparación.*



2.6.4 Análisis de la significación biológica

Una vez anotados los genes podremos intentar interpretar los resultados tratando determinar si las listas se encuentran enriquecidas en algunas categorías biológicas. Para ello podremos llevar a cabo un análisis de sobre-representación o un `Gene Set Enrichment Analysis`.

Por último, visualizaremos de forma sencilla los resultados del análisis de significación biológica, lo que ayudará a comprender y comparar los resultados.

- **Análisis de sobre-representación**

El análisis de sobre-representación toma la lista de genes expresados diferencialmente y busca categorías biológicas en las que algunos genes aparecen con frecuencias inusualmente altas. Es decir, busca genes que aparecen con más frecuencia en una categoría determinada de lo que se esperaría por casualidad.[31]

Para el análisis de enriquecimiento utilizaremos la función `enrichGO()` del paquete `clusterProfiler`. Mediante la función `get_genes_in()` obtendremos la lista de genes seleccionados (parámetro `gene`) y mediante la función `get_gen_universe()` obtendremos el universo de genes es decir todos los genes que se han incluido en el análisis (parámetro `universe`).

Realizaremos el análisis de enriquecimiento para cada una de las comparaciones.

```
universe_EntrezIDs <- get_gen_universe()

listOfTables <- list(inf_vs_uni.unt = topAnnotated_inf_vs_uni.unt,
                     inf_vs_uni.lin = topAnnotated_inf_vs_uni.lin,
                     inf_vs_uni.van = topAnnotated_inf_vs_uni.van)
comparisonsNames <- names(listOfTables)
```

[31] Sánchez-Pla, "An Introduction to Pathway Analysis with R and Bioconductor".

```
get_gen_enriched <- function(genesIn_EntrezIDs,universe_EntrezIDs){
  enrich_go <- enrichGO(gene = genesIn_EntrezIDs,
                        universe = universe_EntrezIDs,
                        keyType = "ENTREZID",
                        OrgDb = org.Mm.eg.db,
                        ont = "BP",
                        pAdjustMethod = "BH",
                        qvalueCutoff = 0.25,
                        readable = TRUE)

  return(enrich_go)
}
```

- Comparación: inf_vs_uni.unt

Table 12: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con 'enrichGO()' para inf vs uni unt: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'unt'.*

Description	GeneRatio	BgRatio	pvalue	p.adjust
regulation of innate immune response	64/475	486/28905	0	0
positive regulation of response to biotic stimulus	57/475	431/28905	0	0
positive regulation of innate immune response	53/475	404/28905	0	0
cytokine-mediated signaling pathway	56/475	471/28905	0	0
myeloid leukocyte migration	43/475	255/28905	0	0
defense response to bacterium	52/475	430/28905	0	0
activation of innate immune response	45/475	303/28905	0	0
immune response-activating signaling pathway	54/475	486/28905	0	0
immune response-regulating signaling pathway	54/475	497/28905	0	0
leukocyte migration	48/475	402/28905	0	0

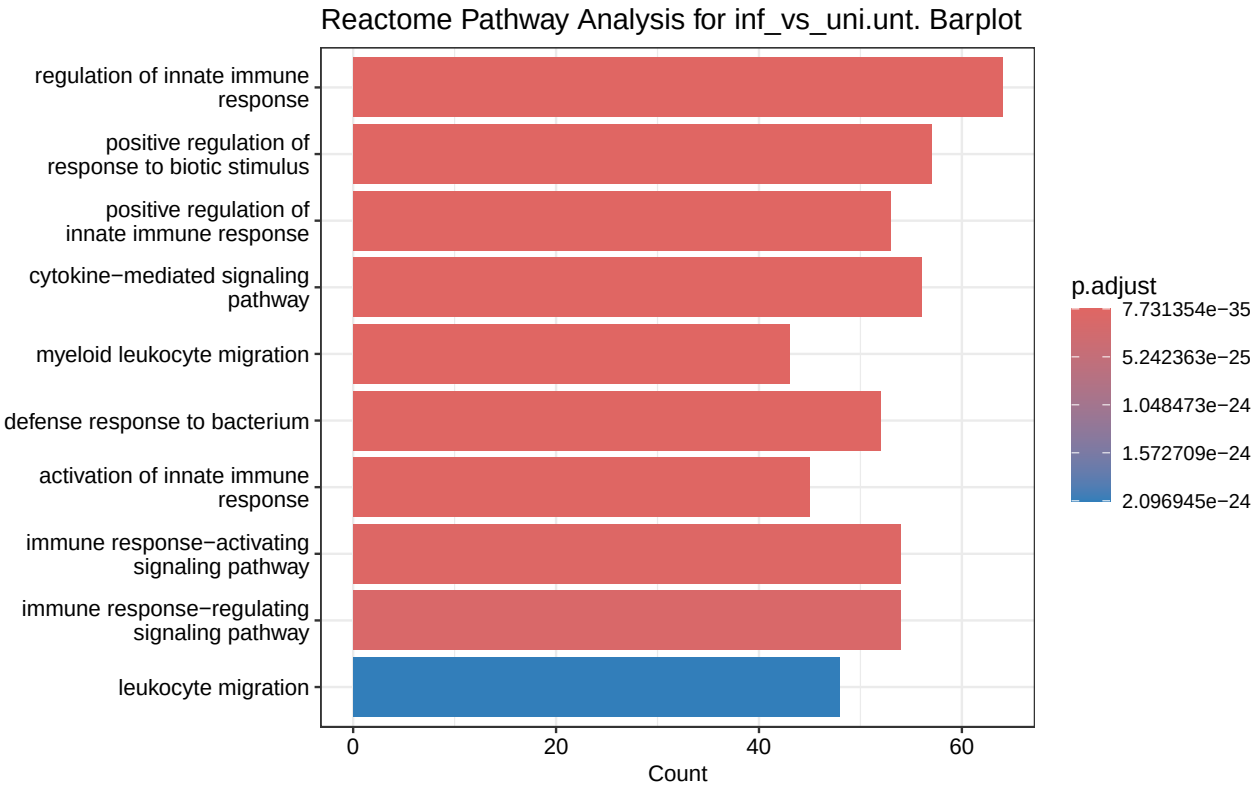


Figure 21: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con* `enrichGO()` *para inf vs uni unt: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento unt`.

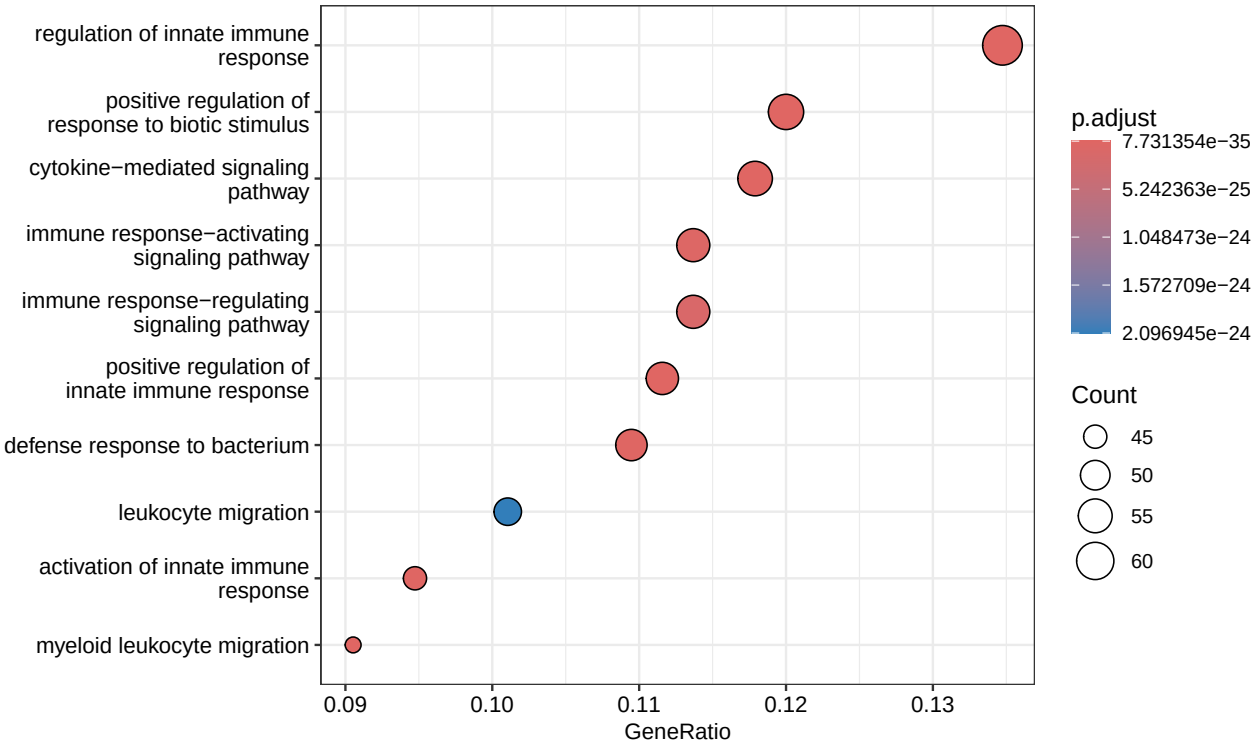


Figure 22: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con* `enrichGO()` *para inf vs uni unt: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento unt`.

- Comparación: inf_vs_uni.lin

Table 13: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con 'enrichGO()' para inf vs uni lin: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'lin'.*

Description	GeneRatio	BgRatio	pvalue	p.adjust
regulation of innate immune response	63/431	486/28905	0	0
defense response to bacterium	56/431	430/28905	0	0
positive regulation of response to biotic stimulus	55/431	431/28905	0	0
positive regulation of innate immune response	49/431	404/28905	0	0
activation of innate immune response	42/431	303/28905	0	0
response to interferon-beta	25/431	72/28905	0	0
response to virus	44/431	377/28905	0	0
negative regulation of hydrolase activity	41/431	329/28905	0	0
humoral immune response	41/431	339/28905	0	0
cytokine-mediated signaling pathway	47/431	471/28905	0	0

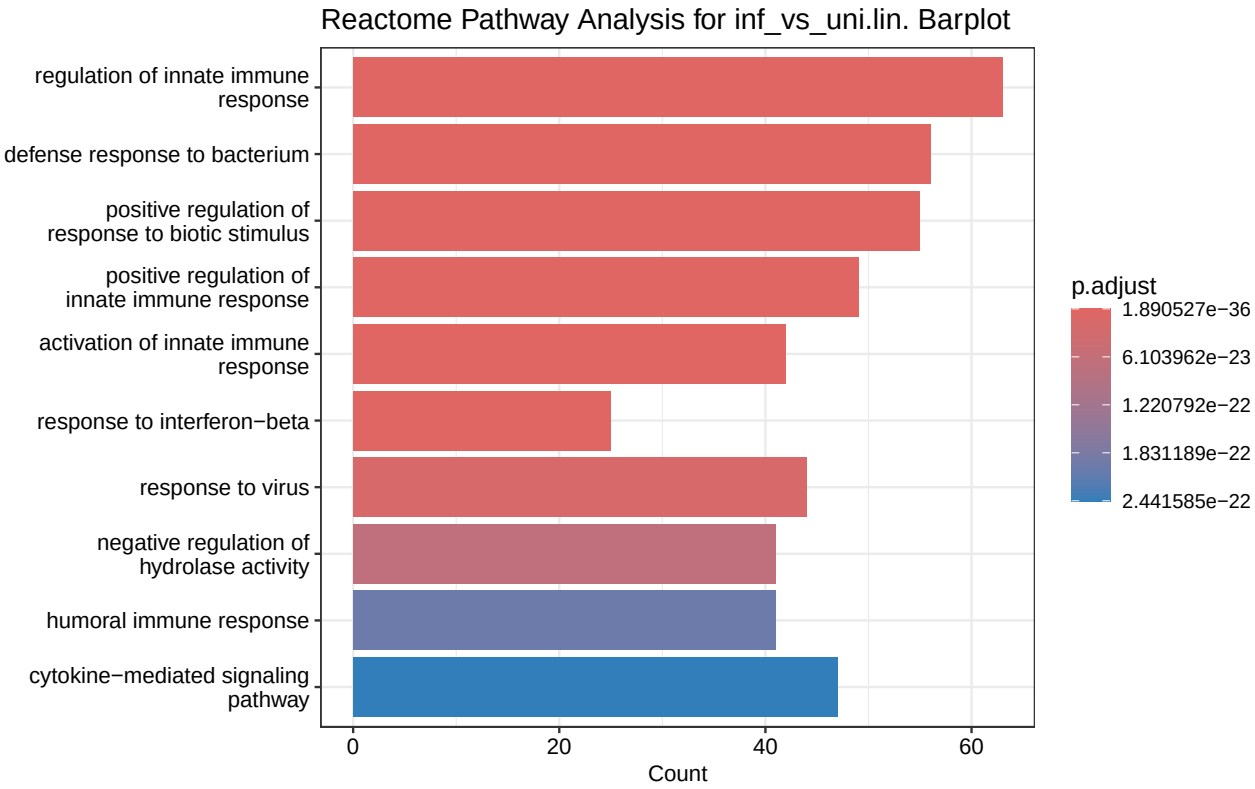


Figure 23: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con* `enrichGO()` *para inf vs uni lin: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento lin`.

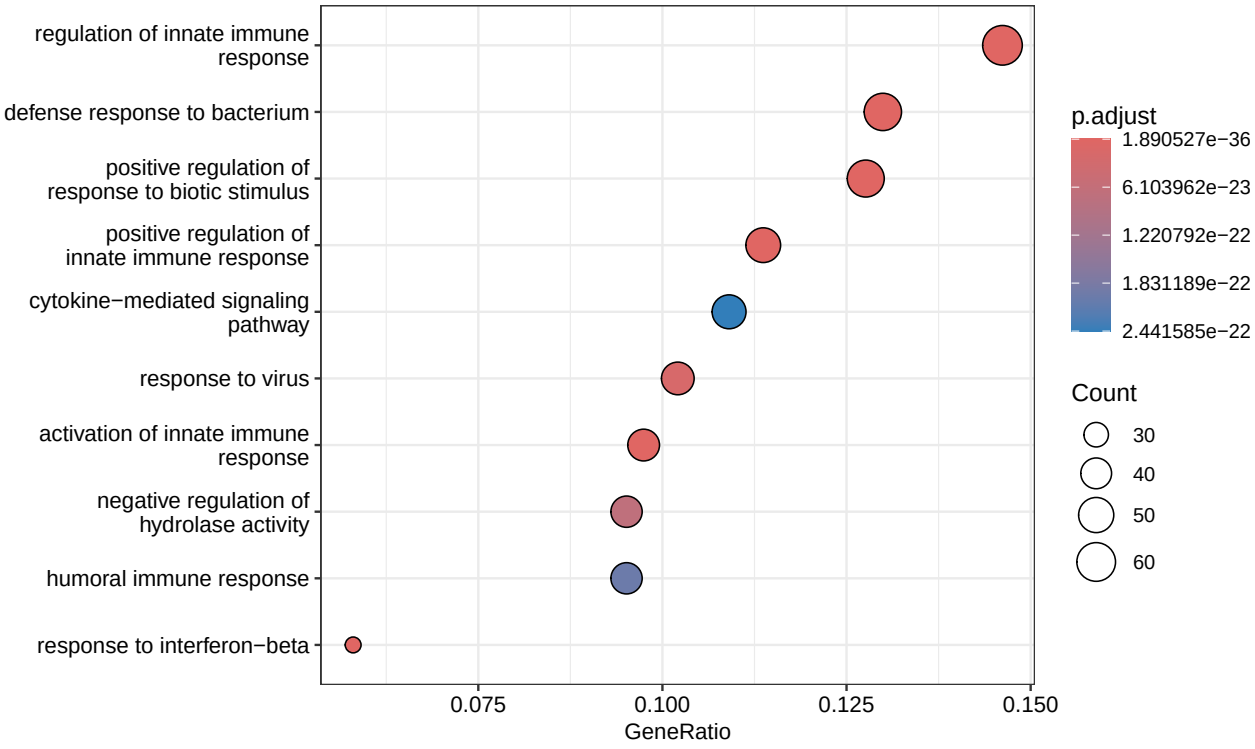


Figure 24: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con* `enrichGO()` *para inf vs uni lin: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento lin`.

- Comparación: inf_vs_uni.van

Table 14: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con 'enrichGO()' para inf vs uni van: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'van'.*

Description	GeneRatio	BgRatio	pvalue	p.adjust
myeloid leukocyte migration	31/275	255/28905	0	0
regulation of innate immune response	39/275	486/28905	0	0
cell chemotaxis	32/275	334/28905	0	0
immune response-activating signaling pathway	37/275	486/28905	0	0
immune response-regulating signaling pathway	37/275	497/28905	0	0
leukocyte chemotaxis	28/275	242/28905	0	0
leukocyte migration	33/275	402/28905	0	0
positive regulation of innate immune response	33/275	404/28905	0	0
activation of innate immune response	29/275	303/28905	0	0
positive regulation of response to biotic stimulus	33/275	431/28905	0	0

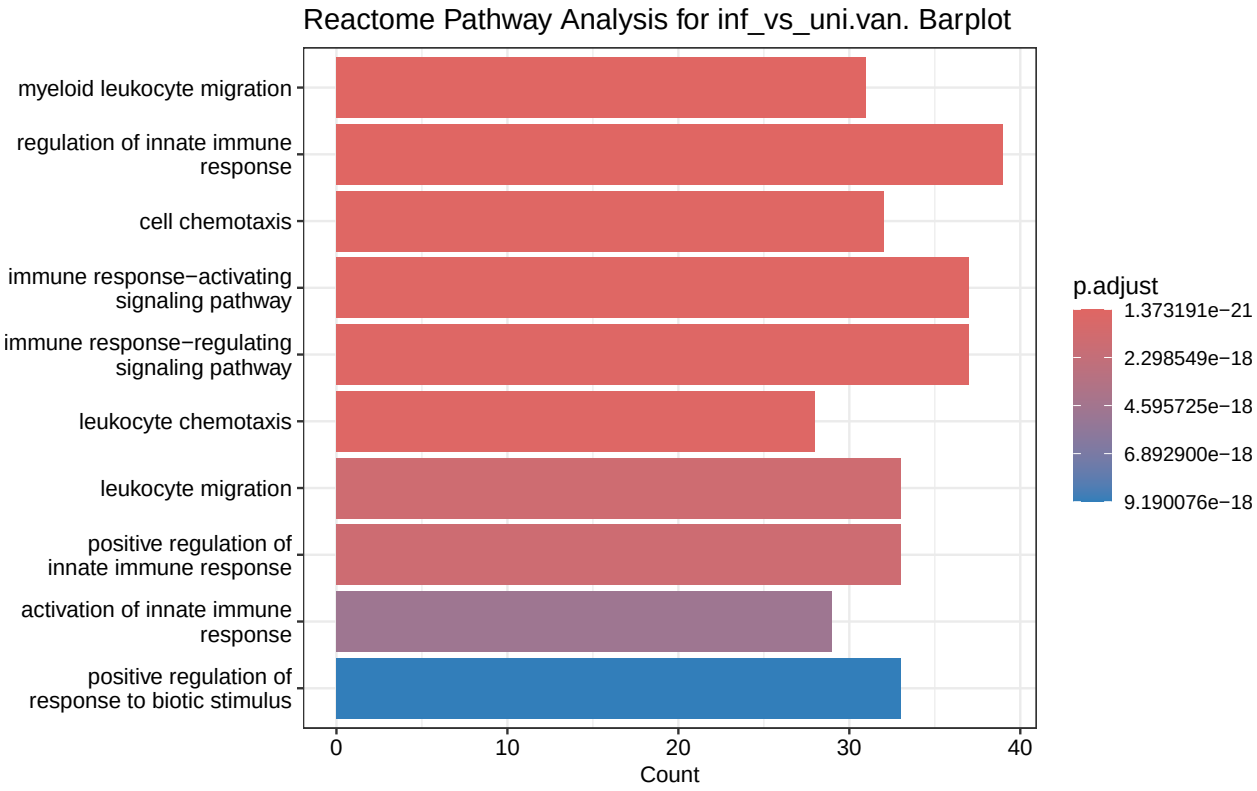


Figure 25: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con* `enrichGO()` *para inf vs uni van: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento van`.

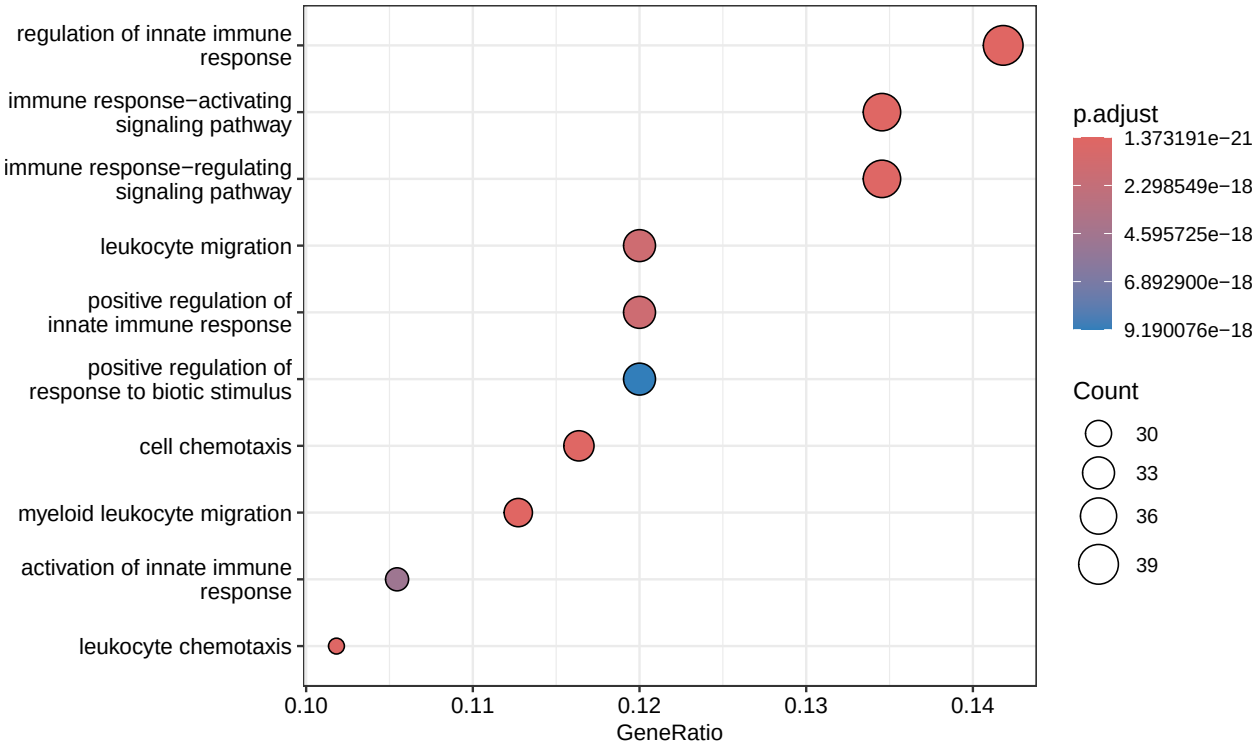


Figure 26: *Análisis de sobre-representación toma la lista de genes expresados diferencialmente con* ***enrichGO()*** *para inf vs uni van: genes que cambian su expresión entre* ***infección*** *y* ***no infección*** *con* ***tratamiento van.***

- **Análisis de enriquecimiento del conjunto de genes (GSEA)**

Los análisis de expresión de conjuntos de genes funcionan con todos los genes y buscan conjuntos (categorías) de genes expresados diferencialmente. Es decir, buscan categorías que estén relacionadas con genes en las partes superior o inferior de la lista de genes.[32]

Para el análisis de conjunto de genes utilizaremos la función `gseKEGG()` del paquete `clusterProfiler`. Mediante la función `get_gen_vector()` obtenemos el parámetro `geneList`.

- Comparación: inf_vs_uni.unt

Table 15: *Análisis de conjunto de genes toma la lista de genes expresados diferencialmente con 'gseKEGG()' para inf vs uni unt: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'unt'.*

Description	setSize	NES	p.adjust
Hematopoietic cell lineage - Mus musculus (house mouse)	34	-2.7346	0e+00
Antigen processing and presentation - Mus musculus (house mouse)	20	-2.6803	1e-04
Neutrophil extracellular trap formation - Mus musculus (house mouse)	41	2.4317	1e-04
Amoebiasis - Mus musculus (house mouse)	23	2.3995	1e-04
NOD-like receptor signaling pathway - Mus musculus (house mouse)	44	2.3488	1e-04
Primary immunodeficiency - Mus musculus (house mouse)	18	-2.4925	3e-04
IL-17 signaling pathway - Mus musculus (house mouse)	17	2.2740	5e-04
Asthma - Mus musculus (house mouse)	11	-2.3458	5e-04
Autoimmune thyroid disease - Mus musculus (house mouse)	13	-2.3829	7e-04
Viral myocarditis - Mus musculus (house mouse)	16	-2.4120	7e-04

[32] Sánchez-Pla, "An Introduction to Pathway Analysis with R and Bioconductor".

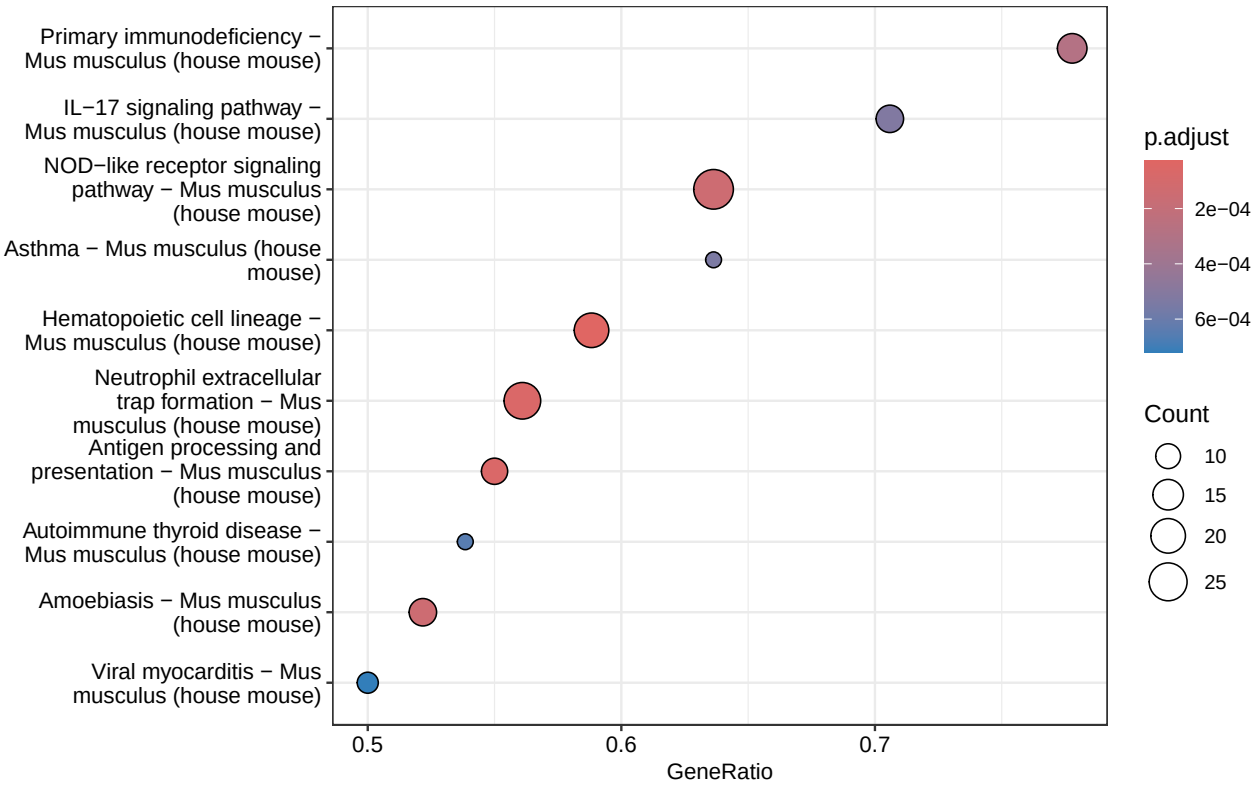


Figure 27: *Análisis de conjunto de genes toma la lista de genes expresados diferencialmente con* `gseKEGG()` *para inf vs uni unt: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento unt`.

- Comparación: inf_vs_uni.lin

Table 16: *Análisis de conjunto de genes toma la lista de genes expresados diferencialmente con 'gseKEGG()' para inf vs uni lin: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'lin'.*

Description	setSize	NES	p.adjust
Complement and coagulation cascades - Mus musculus (house mouse)	36	3.2293	0e+00
Hematopoietic cell lineage - Mus musculus (house mouse)	34	-3.1213	0e+00
Coronavirus disease - COVID-19 - Mus musculus (house mouse)	55	2.4955	0e+00
Pertussis - Mus musculus (house mouse)	28	2.4568	1e-04
Primary immunodeficiency - Mus musculus (house mouse)	18	-2.6936	1e-04
NOD-like receptor signaling pathway - Mus musculus (house mouse)	44	2.3789	2e-04
Neutrophil extracellular trap formation - Mus musculus (house mouse)	41	2.3350	2e-04
Th17 cell differentiation - Mus musculus (house mouse)	36	-2.4183	3e-04
Staphylococcus aureus infection - Mus musculus (house mouse)	32	2.3645	4e-04
Intestinal immune network for IgA production - Mus musculus (house mouse)	18	-2.5350	4e-04

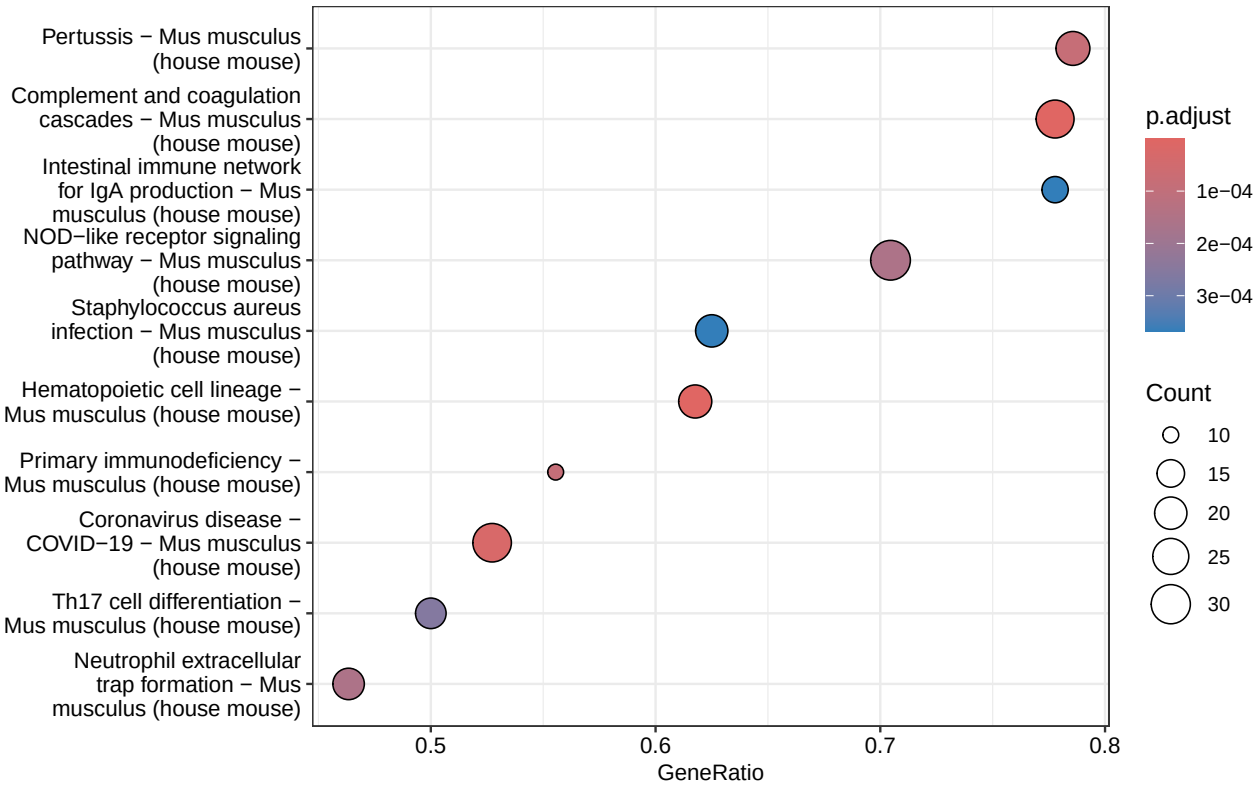


Figure 28: *Análisis de conjunto de genes toma la lista de genes expresados diferencialmente con* `gseKEGG()` *para inf vs uni lin: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento lin`.

- Comparación: inf_vs_uni.van

Table 17: *Análisis de conjunto de genes toma la lista de genes expresados diferencialmente con 'gseKEGG()' para inf vs uni van: genes que cambian su expresión entre 'infección' y 'no infección' con 'tratamiento' 'van'.*

Description	setSize	NES	p.adjust
Hematopoietic cell lineage - Mus musculus (house mouse)	34	-2.7323	0.0000
Graft-versus-host disease - Mus musculus (house mouse)	14	-2.4662	0.0001
Viral myocarditis - Mus musculus (house mouse)	16	-2.4605	0.0002
Antigen processing and presentation - Mus musculus (house mouse)	20	-2.4326	0.0003
Autoimmune thyroid disease - Mus musculus (house mouse)	13	-2.3956	0.0004
NOD-like receptor signaling pathway - Mus musculus (house mouse)	44	2.3610	0.0005
Asthma - Mus musculus (house mouse)	11	-2.3092	0.0005
Intestinal immune network for IgA production - Mus musculus (house mouse)	18	-2.3048	0.0011
Type I diabetes mellitus - Mus musculus (house mouse)	15	-2.2785	0.0011
IL-17 signaling pathway - Mus musculus (house mouse)	17	2.3919	0.0012

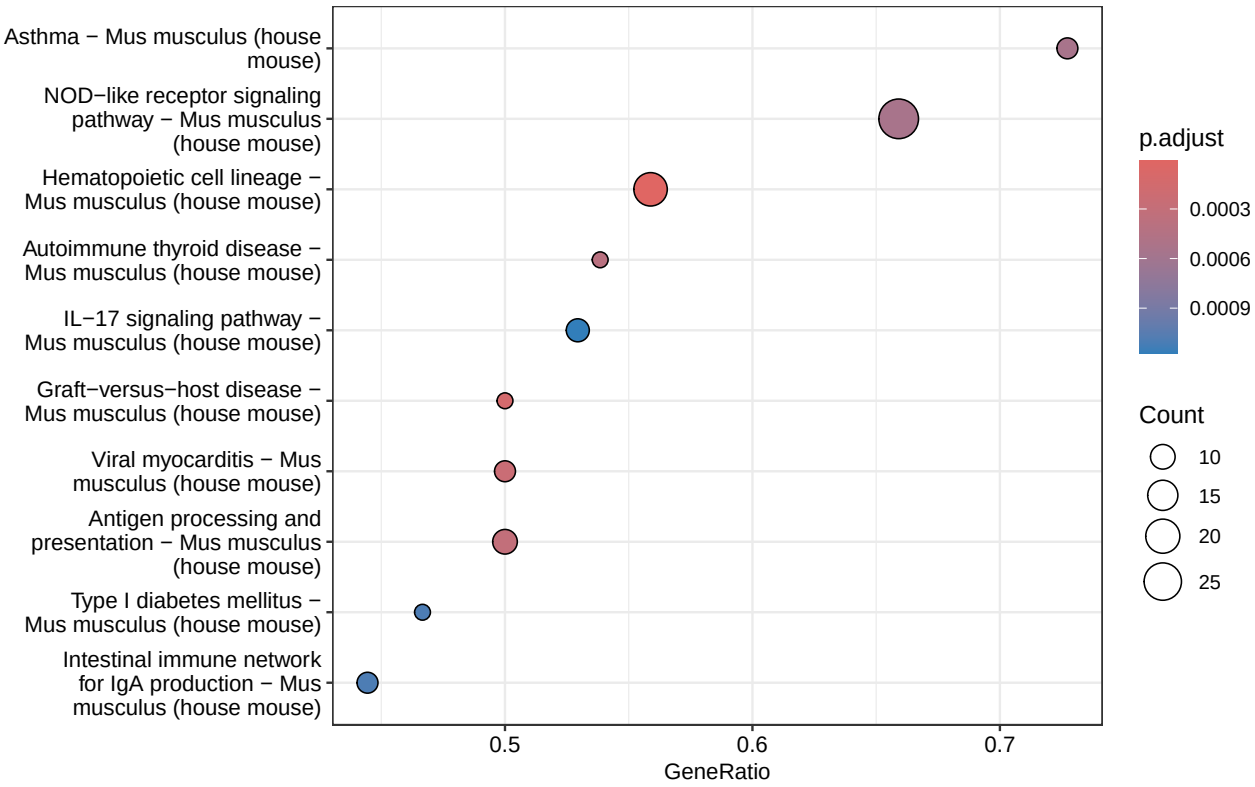


Figure 29: *Análisis de conjunto de genes toma la lista de genes expresados diferencialmente con* `gseKEGG()` *para inf vs uni van: genes que cambian su expresión entre* `infección` *y* `no infección` *con* `tratamiento van`.

3 Resumen de resultados y discusión

3.1 Resumen

Inicialmente hemos preparado el entorno necesario para la elaboración de este informe y la descarga de los datos para su estudio. Una vez los datos estaban disponibles, durante el **pre-procesado** hemos preparado los datos seleccionando la muestra para el análisis. Posteriormente hemos realizado una primera exploración y hemos visto la necesidad de normalizar (función `rma()`) los datos y realizado control de calidad sobre los mismos.

Sobre los datos ya normalizados hemos aplicado un filtrado y hemos dejado los datos preparados para realizar un análisis de los **genes diferencialmente expresados** para cada **comparación** a la que hemos querido dar respuesta

A partir del enunciado de la PEC, hemos identificado las siguientes tres comparaciones:

- Infectados vs no infectados sin tratamiento
- Infectados vs no infectados tratados con LINEZOLID
- Infectados vs no infectados tratados con VANCOMICINA

Utilizando el paquete `limma` hemos obtenido una lista de genes diferencialmente expresados. Para ello hemos seguido los siguientes tres pasos:

- Estimar el modelo: `lmFit()`.
- Estimar los contrastes: `constrasts.fit()`.
- Realizar la prueba de significación: `eBayes()`.

La estimación del modelo lo hemos realizado sobre los datos normalizados y filtrados (función `nsFilter()`) `data_eset_rma_filt`. Para ello hemos diseñado y utilizado la matriz de diseño. Para realizar los contrastes del modelo hemos definido la matriz de contrastes, que recoge las tres comparaciones que queríamos analizar. Para obtener el listado de genes diferencialmente expresados a partir ahí. Los resultados del análisis de expresión diferencial lo hemos extraído con la función `topTable()`.

Después de realizar el análisis de cuales son los genes diferencialmente expresados para cada comparación, hemos realizado un **post-procesado** para tratar de interpretar los resultados siguiendo los siguientes pasos:

- Comparar entre listas para determinar qué genes cambian simultáneamente o no en las diferentes comparaciones.
- Anotar sobre la lista de genes mediante la información publicada existente en bases de datos.
- Visualizar los genes seleccionados en las distintas comparaciones.
- Analizar la significación biológica de las listas mediante el **análisis de enriquecimiento** y/o **gene set analysis**.

3.2 Resultados

Durante la creación del presente documento hemos generado diferentes conjuntos de datos que hemos ido almacenando en el directorio `resultsDire`. Una vez hemos llegado a este punto del proceso, mostraremos la lista de dichos conjuntos de datos que han sido resultado del proceso:

Table 18: *Listado de los conjuntos datos que hemos generado en análisis del presente documento.*

dataset
data_gen_enriched_inf_vs_uni.lin.csv
data_gen_enriched_inf_vs_uni.unt.csv
data_gen_enriched_inf_vs_uni.van.csv
data_gen_selected_annoted_db_mouse4302.html
data_gse_inf_vs_uni.lin.csv
data_gse_inf_vs_uni.unt.csv
data_gse_inf_vs_uni.van.csv
data_normalized_filtered_selected.csv
data_normalized_filtered.csv
data_normalized.csv
data_normalized.Rda
data_raw.csv
data_top_annotated_inf_vs_uni_lin.csv
data_top_annotated_inf_vs_uni_unt.csv
data_top_annotated_inf_vs_uni_van.csv
data_top_inf_vs_uni_lin.csv
data_top_inf_vs_uni_unt.csv
data_top_inf_vs_uni_van.csv

Table 19: *Los dos subdirectorios, raw y normalizado de control de calidad generados.*

dataset
arrayQuality_norm
arrayQuality_raw

Table 20: *Los ficheros contenidos dentro que cada control de calidad generado.*

x
arrayQualityMetrics.css
arrayQualityMetrics.js
box.pdf
box.png
dens.pdf
dens.svg
hm.pdf
hm.png
index.html
ma.pdf
ma.png
msd.pdf
msd.png
out box.pdf
out box.png
out hm.pdf
out hm.png
out ma.pdf
out ma.png
pca.pdf
pca.svg

En las primera exploraciones, en las Figuras: 1, 2 se aprecia que los valores de las muestras están fuera de los valores normales. Se pueden apreciar algunos valores atípicos mediante las visualizaciones obtenidas del la función `arrayQualityMetrics()` en las Figuras: 6, 7, 8.

No obstante, tras normalizar los datos, en las Figuras: 9, 10, podemos ver cierta normalidad y en las Figuras: 12, 13, 11 que los valores atípicos se han reducido, y en general podemos decir que los datos son correctos.

Tras normalizar y filtrar los datos en post-procesado, en Figura 17 mediante el diagrama de Venn hemos podido detectar un cierto número de genes diferencialmente expresados. Este diagrama representa el número de genes que se han denominado expresados diferencialmente en cada comparación con un valor de corte establecido (`p.value=0.01`, `lfc=1`). Además, podemos identificar mediante la representación cuantos genes son compartidos por una o más selecciones. En las Figuras 18, 19, 20 se muestras los 10 genes principales para cada condición.

También hemos podido representar la comparación múltiple mediante los resultados obtenidos en la Tabla 8 que representa para condición los genes diferencialmente expresados con los valores de corte establecidos (`p.value=0.01`, `lfc=1`). Las listas completas de genes seleccionados (ordenados de mayor a menor p–valor) pueden verse en los en las Tablas: 5, 6, 7

Por último, hemos realizado el análisis de significación biológica mediante el análisis de sobre-representación que se puede visualizar en las Figuras: 21, 23 y 25. Además el análisis de GSE se puede visualizar en las Figuras: 27, 28 y 29.

3.3 Discusión

Como primer análisis de microarrays, lo que podemos destacar el es lo complicado que resulta tener conclusiones claras, sobre todo en el apartado del análisis de la significación biológica, más haya de obtener los listados para cada uno de los casos y ver que posiblemente comparten categorías biológica en las 3 condiciones.

Por otro lado, los criterios de corte (`threshold`) tanto para el filtrado de los datos, como para la selección del listado genes diferencialmente expresados, han sido totalmente arbitrarios y lo hemos establecido siguiendo el criterio de los ejemplos vistos en la asignatura. No obstante, sería interesante experimentar con estos criterios algo más para ver como varía con éste el listado de genes diferencialmente expresados.

Por último, sí que podemos destacar, también en la linea de lo visto en los apuntes, que el número de muestras es limitado.

4 References

[GS19] Ricardo Gonzalo Sanz and Alex Sánchez-Pla. "Case Study 1Microarrays Analysis.html". In: (2019). Ed. by Verónica Bolón-Canedo and Amparo Alonso-Betanzos. URL: https://doi.org/10.1007/978-1-4939-9442-7_5.

[GS20] Ricardo Gonzalo Sanz and Alex Sánchez-Pla. "Statistical Analysis of Microarray data". In: (2020). URL: https://github.com/ASPteaching/Omics_Data_Analysis-Case_Study_1-Microarrays/blob/43479badd7cab68a7e8d7ed695285fee16bcba41/Case_Study_1-Microarrays_Analysis.html.

[Iri+03] Rafael A. Irizarry et al. "Exploration, normalization, and summaries of high density oligonucleotide array probe level data". In: *Biostatistics* 4.2 (Apr. 2003), pp. 249–264. ISSN: 1465-4644. DOI: 10.1093/biostatistics/4.2.249.

[Mir21] Ricardo Gonzalo y Alex Sanchez Mireia Ferrer. "Workflow de análisis de expresión diferencial usando RNA-seq". In: (Dec. 2021). URL: https://github.com/ASPteaching/Analisis_de_datos_omicos-Ejemplo_2-RNASeq/blob/fb52d73853485353187cf2be813d3cb1e6b1900e/Workflow_basico_de_RNASeq.html.

[Sán20] Alex Sánchez-Pla. "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado". In: (May 2020). URL: https://github.com/alexsanchezpla/Ejemplo_de_Analisis_de_Microarrays_con_Bioconductor.

[Sán23] Alex Sánchez-Pla. "An Introduction to Pathway Analysis with R and Bioconductor". In: (Feb. 2023). URL: https://github.com/ASPteaching/An-Introduction-to-Pathway-Analysis-with-R-and-Bioconductor.

5 Appendix

Se pueden consultar el presente documento y así como las exportaciones realidas en la siguiete dirección git: https://github.com/azenzano/Zenzano-Sarasola-Ager-PEC2.

Carga de librerías

```
library(dplyr)
library(oligo)
library(Biobase)
library(ggplot2)
library(ggrepel)
library(kableExtra)
library(arrayQualityMetrics)
library(pvca)
library(affyio)
library(genefilter)
library(mouse4302.db)
library(limma)
library(annaffy)
library(annotate)
library(tidyr)
library(gplots)
library(clusterProfiler)
```

Directorios y subdirectorios de los datos

```
workingDire <-getwd()
downloadsDire <-file.path(workingDire, "downloads/")
dataDire <-file.path(workingDire, "data/")
resultsDire <- file.path(workingDire,"results/")
celfilesDire <- file.path(workingDire,"celfiles")
```

Simulamos el dataset targets basándonos en la descripción proporcionada

```
allTargets <- data.frame(
  sample = c("GSM944831", "GSM944838", "GSM944845", "GSM944852", "GSM944859",
             "GSM944833", "GSM944840", "GSM944847", "GSM944854", "GSM944861",
             "GSM944834", "GSM944841", "GSM944848", "GSM944855", "GSM944862",
             "GSM944832", "GSM944839", "GSM944846", "GSM944853", "GSM944860",
             "GSM944835", "GSM944842", "GSM944849", "GSM944856", "GSM944863",
             "GSM944836", "GSM944843", "GSM944850", "GSM944857", "GSM944864",
             "GSM944837", "GSM944844", "GSM944851", "GSM944858", "GSM944865"),
  infection = c(rep("uninfected", 15), rep("S. aureus USA300", 20)),
  time = c(rep("hour 0", 15), rep("hour 2", 5), rep("hour 24", 15)),
  agent = c(rep("untreated", 5), rep("linezolid", 5), rep("vancomycin", 5),
            rep("untreated", 5), rep("untreated", 5), rep("linezolid", 5),
            rep("vancomycin", 5)),
  groupo = c(rep("uni_unt", 5), rep("uni_lin", 5), rep("uni_van", 5),
             rep("inf_unt", 5), rep("inf_unt", 5), rep("inf_lin", 5), rep("inf_van", 5))
)
```

Descarga de los datos de la base de datos de NCBI.

```
file_path <- paste0(downloadsDire,"GSE38531_RAW.tar")
if (file.exists(file_path)) {
```

```
  print("The file exists!")
} else {
  print("The file does not exist, therefore it will be downloaded")
  url <- "https://www.ncbi.nlm.nih.gov/geo/download/?acc=GSE38531&format=file"
  utils::download.file(url, destfile=file_path, mode="wb")
}
```

Descomprimimos el fichero GSE38531_RAW.tar para obtener los ficheros `.CEL`.

```
check_files = list.files(celfilesDire)
if (length(check_files) == 0){
  utils::untar(file_path, exdir = celfilesDire)
  files <-paste("celfiles",  dir("celfiles",pattern = "*.gz"), sep="/")
  for(filename in files){
    R.utils::gunzip(filename,
                    destname = paste0(gsub("[_].*","\\2",filename),".CEL"),
                    overwrite=FALSE)
  }
}
```

Función `filter_microarray()` para generar la selección para la muestra de estudio.

```
filter_microarray <- function(allTargets, seed) {
  # Esta función permitirá extraer 24 muestras distintas a cada uno con tan solo
  # llamarla usando como

  # Configurar la semilla aleatoria
  set.seed(seed)

  # Filtrar las filas donde 'time' no sea 'hour 2'
  filtered <- subset(allTargets, time != "hour 2")

  # Dividir el dataset por grupos únicos de 'infection' + 'agent'
  filtered$group <- interaction(filtered$infection, filtered$agent)

  # Seleccionar 4 muestras al azar de cada grupo
  selected <- do.call(rbind, lapply(split(filtered,filtered$group),function(group_data){
    if (nrow(group_data) > 4) {
      group_data[sample(1:nrow(group_data), 4), ]
    } else {
      group_data
    }
  }))

  # Obtener los índices originales como nombres de las filas seleccionadas
  original_indices <- match(selected$sample, allTargets$sample)

  # Modificar los rownames usando 'sample' y los índices originales
  rownames(selected) <- paste0(selected$sample, ".", original_indices)

  # Eliminar la columna 'group' y devolver el resultado
  selected$group <- NULL
  return(selected)
}
```

Generación de selección para la muestra de estudio.

```
result <- filter_microarray(allTargets, seed=1935) %>%
  mutate(sample_id = sample) %>%
  mutate(short_name = paste0(sample,"_",groupo)) %>%
  mutate(sample=paste0(sample,".CEL"))

write.table(result, file = "./celfiles/targets.txt", sep="\t",
            row.names=FALSE, quote=FALSE)
```

Creación del fichero `targets.txt` con las covariables del estudio.

```
sampleInfo <- read.AnnotatedDataFrame(file.path(celfilesDire,"targets.txt"),
                                      header = TRUE, row.names = 1, sep="\t")

fileNames <- rownames(pData(sampleInfo))
celFiles <- file.path(celfilesDire,fileNames)
rawData <- read.celfiles(celFiles, phenoData = sampleInfo, verbose = FALSE)
```

Función `plotPCA3()` para visualizar PCA.[33]

```
plotPCA3 <- function (datos,labels,factor,title,scale,colores,size=1.5,glineas=0.25){
  data <- prcomp(t(datos),scale=scale)
  # plot adjustments
  dataDf <- data.frame(data$x)
  Group <- factor
  loads <- round(data$sdev^2/sum(data$sdev^2)*100,1)
  # main plot
  p1 <- ggplot(dataDf,aes(x=PC1, y=PC2)) +
    theme_classic() +
    geom_hline(yintercept = 0, color = "gray70") +
    geom_vline(xintercept = 0, color = "gray70") +
    geom_point(aes(color = Group), alpha = 0.55, size = 3) +
    coord_cartesian(xlim = c(min(data$x[,1])-5,max(data$x[,1])+5)) +
    scale_fill_discrete(name = "Group")
  # avoiding labels superposition
  p1 + geom_text_repel(aes(y = PC2 + 0.25, label = labels),
                       segment.size = 0.25, size = size) +
    labs(x = c(paste("PC1",loads[1],"%")),y=c(paste("PC2",loads[2],"%"))) +
    ggtitle(paste("Principal Component Analysis for: ",title,sep=" "))+
    theme(plot.title = element_text(hjust = 0.5)) +
    scale_color_manual(values=colores)
}
```

Control de calidad con el método `arrayQualityMetrics()` del paquete `arrayQualityMetrics` específico para microarrays.

```
if(!dir.exists(file.path(resultsDire, "arrayQuality_raw"))){
  arrayQualityMetrics(rawData,outdir=file.path(resultsDire,"arrayQuality_raw"),force=TRUE)
}
```

Normalizamos las muestras mediante la función `rma()`.

```
eset_rma <- rma(rawData)
```

Analizamos el efecto lote mediante la función `pvcaBatchAssess()` del paquete `pvca`.

[33] Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

```
pData(eset_rma) <- result
pct_threshold <- 0.6
batch.factors <- c("time","agent")
pvcaObj <- pvcaBatchAssess (eset_rma, batch.factors, pct_threshold)
```

Analizamos la variabilidad de los genes.

```
sds <- apply(exprs(eset_rma), 1, sd)
sdsO<- sort(sds)
```

Filtramos mediante la función `nsFilter()` del parque `genefilter`.

```
data_nsFilter <- nsFilter(eset_rma,
                          require.entrez=TRUE,
                          remove.dupEntrez=TRUE,
                          var.func=IQR,
                          var.cutoff=0.9,
                          filterByQuantile=TRUE,
                          var.filter=TRUE,
                          feature.exclude="^AFFX"
)
```

Guardamos los datos normalizados y los filtrados.

```
write.csv(exprs(eset_rma), file="./results/normalized.Data.csv")
write.csv(exprs(eset_rma_filt), file="./results/normalized.Filtered.Data.csv")
save(eset_rma, eset_rma_filt, file="./results/normalized.Data.Rda")
```

Definimos la matriz de diseño.

```
designMat<- model.matrix(~0+groupo, pData(data_eset_rma_filt))
colnames(designMat) <- c("inf_lin", "inf_unt", "inf_van",
                         "uni_lin", "uni_unt", "uni_van")
```

Definimos la matriz de contrastes:

```
cont.matrix <- makeContrasts (inf_vs_uni.unt = inf_unt-uni_unt,
                              inf_vs_uni.lin = inf_lin-uni_lin,
                              inf_vs_uni.van = inf_van-uni_van,
                              levels=designMat)
```

Realizamos el análisis de expresión diferencial para: `inf_vs_uni.unt`, `inf_vs_uni.lin` y `inf_vs_uni.van` con la función `topTable`

```
topTab_inf_vs_uni.unt <- (topTable(fit.main, number=nrow(fit.main),
                                   coef="inf_vs_uni.unt", adjust="fdr") )
topTab_inf_vs_uni.lin<- (topTable(fit.main, number=nrow(fit.main),
                                  coef="inf_vs_uni.lin", adjust="fdr"))
topTab_inf_vs_uni.van<- (topTable(fit.main, number=nrow(fit.main),
                                  coef="inf_vs_uni.van", adjust="fdr"))
```

Realizamos la comparación múltiple con la función `decideTests()` con los siguientes parámetros: `adjust.method="fdr"`, `p.value=0.01`, `lfc=1`. Se seleccionaran los genes que cambian en una o más condiciones.[34]

```
res <- decideTests(fit.main, method="separate", adjust.method="fdr", p.value=0.01, lfc=1)
sum.res.rows <- apply(abs(res),1,sum)
```

[34]Sánchez-Pla, "Análisis de datos de microarrays con R y Bioconductor, Selección de genes diferencialmente expresado".

```
res.selected <- res[sum.res.rows!=0,]
```

Exportamos las anotaciones asociadas a través de la base de datos `mouse4302.db` con la función `aafTableAnn()` del paquete `annafy` para nuestros genes seleccionados

```
genesSelected <- rownames(res.selected)
at <- aafTableAnn(genesSelected, "mouse4302.db")
check_files = list.files(paste0(resultsDire),pattern="data_gen_selected_annoted_db_mouse4302")
if (length(check_files) == 0){
  saveHTML(at, file.path(resultsDire, "data_gen_selected_annoted_db_mouse4302.html"),
           "Annotations for selected genes")
}
```

Crearemos la función `annotatedTopTable()` para asociar las anotaciones de la base de datos a nuestro listado de genes, y añadir así los siguientes identificadores:[35] `SYMBOL`, `ENTREZID`, `GENENAME`.

```
annotatedTopTable <- function(topTab, anotPackage){
  topTab <- cbind(PROBEID=rownames(topTab), topTab)
  myProbes <- rownames(topTab)
  thePackage <- eval(parse(text = anotPackage))
  geneAnots <- AnnotationDbi::select(thePackage,myProbes,c("SYMBOL","ENTREZID","GENENAME"))
  annotatedTopTab<- merge(x=geneAnots, y=topTab, by.x="PROBEID", by.y="PROBEID")
  return(annotatedTopTab)
}
```

Creamos el conjunto de datos para visualizarlo con un mapa de calor.

```
probesInHeatmap <- rownames(res.selected)
data_eset_rma_filt_sel <- exprs(
  data_eset_rma_filt)[rownames(exprs(data_eset_rma_filt)) %in% probesInHeatmap,]
genes <- rownames(data_eset_rma_filt_sel)
geneSymbols <- AnnotationDbi::select(mouse4302.db,keys = genes, columns = c("SYMBOL"))

# Determine the indices for the non-NA genes
non_na_idx <- which(is.na(geneSymbols$SYMBOL) == FALSE)
# Return only the genes with annotations using indices
geneSymbols <- geneSymbols[non_na_idx, ]
# Determine the indices for the non-duplicated genes
non_duplicates_idx <- which(duplicated(geneSymbols$SYMBOL) == FALSE)
# Return only the non-duplicated genes using indices
geneSymbols <- geneSymbols[non_duplicates_idx, ]
data_eset_rma_filt_sel_anota <- exprs(
  data_eset_rma_filt)[rownames(exprs(data_eset_rma_filt)) %in% geneSymbols$PROBEID,]
rownames(data_eset_rma_filt_sel_anota) <- geneSymbols$SYMBOL

write.csv(data_eset_rma_filt_sel_anota,
          file = file.path("./results/data_normalized_filtered_selected.csv"))
```

Para el análisis de enriquecimiento utilizaremos la función `enrichGO()` del paquete `clusterProfiler`. Mediante la función `get_genes_in()` obtenemos el parámetro `gene` y mediante la función `get_gen_universe()` el parámetro `universe`.

Función para obtener los genes del estudio.

```
get_genes_in <- function(topTab){
  topGenes <- topTabAnno %>%
```

[35] Gonzalo Sanz and Sánchez-Pla, "Case Study 1Microarrays Analysis.html".

```
    filter(adj.P.Val<0.05 & logFC>1) %>%
    dplyr::arrange(desc(logFC)) %>%
    distinct(ENTREZID, .keep_all = TRUE)

  genesIn_EntrezIDs <- topGenes$ENTREZID
  return(genesIn_EntrezIDs)

}
```

Función para obtener los genes del universo mediante las base de datos `org.Mm.egGO` y `org.Mm.egPATH`.

```
get_gen_universe <- function(){
  mapped_genes2GO <- mappedkeys(org.Mm.egGO)
  mapped_genes2KEGG <- mappedkeys(org.Mm.egPATH)
  mapped_genes <- union(mapped_genes2GO , mapped_genes2KEGG)
  universe_EntrezIDs <- mapped_genes
  return(universe_EntrezIDs)

}
```

Análisis de sobre-representación

```
for (i in 1:length(listOfTables)){

  topTabAnno <- listOfTables[[i]]
  genesIn_EntrezIDs <- get_genes_in(topTabAnno)
  enrich_go = get_gen_enriched(genesIn_EntrezIDs,universe_EntrezIDs)

  cat("\n* Comparación: ", comparisonsNames[i],"\n")

  write.csv(as.data.frame(enrich_go),
            file =paste0("./results/","data_gen_enriched_",comparisonsNames[i],".csv"),
            row.names = FALSE)

  print(barplot(enrich_go, showCategory = 15, font.size = 8,
                title = paste0("Reactome Pathway Analysis for ",
                               comparisonsNames[i],". Barplot")))

  print(dotplot(enrich_go, showCategory=9))
}
```

Función para realizar y obtener en análisis enriquecido con la función `enrichGO()` del paquete `clusterProfiler`.

```
get_gen_enriched <- function(genesIn_EntrezIDs,universe_EntrezIDs){
  enrich_go <- enrichGO(gene = genesIn_EntrezIDs,
                        universe = universe_EntrezIDs,
                        keyType = "ENTREZID",
                        OrgDb = org.Mm.eg.db,
                        ont = "BP",
                        pAdjustMethod = "BH",
                        qvalueCutoff = 0.25,
                        readable = TRUE)

  return(enrich_go)
}
```

Función para obtener los genes del estudio.

```
get_gen_vector <- function(topTab){
  geneList <- topAnnotated_inf_vs_uni.unt %>%
    dplyr::arrange(desc(logFC)) %>%
    distinct(ENTREZID, .keep_all = TRUE) %>%
    dplyr::arrange(desc(logFC))

  genesVector <- geneList$logFC
  names(genesVector) <- geneList$ENTREZ

  return(genesVector)
}
```

Función para realizar y obtener en análisis `GSE` con la función `gseKEGG()` del paquete `clusterProfiler`.

```
set.seed(1935)
get_gen_set_enriched <- function(genesVector){
  gse_kegg <- gseKEGG(geneList = genesVector,
                      organism = "mouse",
                      keyType = "kegg",
                      exponent = 1,
                      minGSSize = 10,
                      maxGSSize = 500,
                      pvalueCutoff = 0.05,
                      pAdjustMethod = "BH",
                      # nPerm = 10000, #augmentem permutacions a 10000
                      verbose = TRUE,
                      use_internal_data = FALSE,
                      seed = TRUE,
                      eps=0,
                      by="fgsea"
  )

  return(gse_kegg)

}
```

Análisis de `GSE`.

```
for (i in 1:length(listOfTables)){
  topTabAnno <- listOfTables[[i]]

  genesVector <- get_gen_vector(topTabAnno)
  gse_kegg <- get_gen_set_enriched(genesVector)

  cat("\n* Comparison: ", comparisonsNames[i],"\n")

  df_gsea_result <- as.data.frame(setReadable(gse_kegg, OrgDb = org.Mm.eg.db, keyType ="ENTREZID" ))
  print(kable(df_gsea_result[,c("Description","setSize","NES","p.adjust")]))


  print(head(gse_kegg))

  write.csv(as.data.frame(gse_kegg),
            file =paste0("./results/","data_gse_",comparisonsNames[i],".csv"),
```

```
            row.names = FALSE)

  p<- dotplot(gse_kegg, showCategory = 10, font.size = 10,
              title =paste("Enriched Pathways\n", comparisonsNames[1] ,
                           split=".sign") + facet_grid(.~.sign))
  print(p)
}
```