Análisis de datos Ómicos - PAC 2

Maria Lucas

2023-05-14

Table of Contents

# Introducción

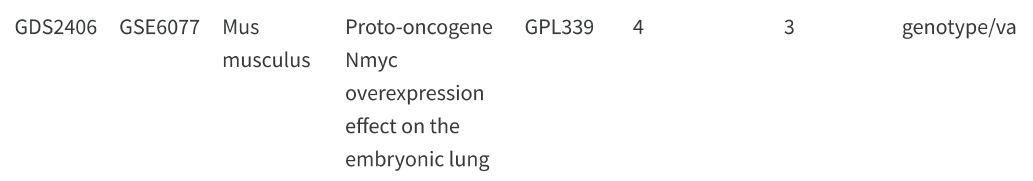
# Objetivos

* Evaluar la calidad de los datos obtenidos mediante microarrays Affymetrix para asegurar la fiabilidad de los resultados.
* Identificar y cuantificar los genes que presentan expresión diferencial entre la línea celular con Nmyc sobreexpresado y la Wild type.
* Analizar las anotaciones funcionales y vías biológicas asociadas a los genes diferencialmente expresados, con el fin de comprender los procesos biológicos afectados por la sobreexpresión de Nmyc.
* Determinar la frecuencia y relevancia de las anotaciones funcionales identificadas en los genes de interés, para comprender mejor su implicación en la función celular y el desarrollo pulmonar.
* Investigar las posibles implicaciones clínicas de los genes diferencialmente expresados, examinando su asociación con enfermedades pulmonares

# Métodos

### Obtención de los datos

Los datos utilizados en este estudio corresponden al conjunto de datos GSE6077, disponible en el repositorio público del Gene Expression Omnibus (GEO) de NCBI. Para acceder a estos datos, se visitó el sitio web del GEO (<https://www.ncbi.nlm.nih.gov/geo/>) y se realizó una búsqueda utilizando el identificador GSE6077. Posteriormente, se descargaron los archivos de datos relevantes para el estudio, que incluyeron los archivos de expresión génica en formato CEL.



Con el fin de organizar la información de las muestras experimentales, se procedió a la creación de un objeto llamado “targets” en el entorno de programación R. Se analizó el objeto phenoData que contiene información relevante sobre las características de cada muestra, como la condición experimental o la procedencia de las muestras. Se creó manualmente un dataframe que contenía los detalles de cada muestra: nombre del archivo, el grupo experimental al que pertenecía (Wild Type o Sobre expresado) y información para la posterior visualización de los datos (color y nombre de las muestras) (**Código 1**). La creación de este objeto permitió asignar las diferentes variables a columnas específicas, facilitando así el acceso y la manipulación de los datos durante el análisis.

La carga de los datos se realizó utilizando el paquete “oligo” en R, que proporciona funciones específicas para el manejo de datos de microarrays Affymetrix. Los archivos CEL fueron importados mediante la función “read.celfiles()”, que permitió leer y procesar los archivos de datos del microarray (**Código 2**).

# Código 1  
  
# Load GEOquery package  
library(GEOquery)

## Loading required package: Biobase

## Loading required package: BiocGenerics

##   
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:stats':  
##   
## IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':  
##   
## anyDuplicated, aperm, append, as.data.frame, basename, cbind,  
## colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,  
## get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,  
## match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,  
## Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,  
## table, tapply, union, unique, unsplit, which.max, which.min

## Welcome to Bioconductor  
##   
## Vignettes contain introductory material; view with  
## 'browseVignettes()'. To cite Bioconductor, see  
## 'citation("Biobase")', and for packages 'citation("pkgname")'.

## Setting options('download.file.method.GEOquery'='auto')

## Setting options('GEOquery.inmemory.gpl'=FALSE)

# Set GEO accession number  
gse = "GSE6077"  
  
# Download and load series matrix file and extract phenoData  
gse\_data = getGEO(gse, GSEMatrix = TRUE)

## Found 1 file(s)

## GSE6077\_series\_matrix.txt.gz

pheno = pData(phenoData(gse\_data[[1]]))  
print(pheno[8:12]) # Print some of the more relevant data

## source\_name\_ch1 organism\_ch1  
## GSM140827 whole lung E18 normal Mus musculus  
## GSM140863 whole lung E18 normal Mus musculus  
## GSM140864 whole lung E18 sftpc-nmyc Mus musculus  
## GSM140865 whole lung E18 sftpc-nmyc Mus musculus  
## characteristics\_ch1  
## GSM140827 normal embryonic day 18 lungs  
## GSM140863 normal lung  
## GSM140864 lungs were over expressing nmyc-gfp fusion protein using the lung surfactant c promoter.  
## GSM140865 whole lungs overexpressing an nmyc-gfp fusion protein via the lung surfactant c promoter  
## molecule\_ch1 extract\_protocol\_ch1  
## GSM140827 total RNA RNeasy kit from QIagen  
## GSM140863 total RNA RNeasy kit from QIagen  
## GSM140864 total RNA RNeasy kit from QIagen  
## GSM140865 total RNA RNeasy kit from QIagen

# Define targets  
filename = c('GSM140827.CEL', 'GSM140863.CEL', 'GSM140864.CEL', 'GSM140865.CEL')  
class = c('WT', 'WT', 'OE', 'OE')  
shortName = c('27\_WT', '63\_WT', '64\_OE', '65\_OE')  
color = c('red', 'red', 'blue', 'blue')  
# Create dataframe  
targetsDF = data.frame(Filename = filename, Class = class, ShortName = shortName, Color = color)  
print(targetsDF) # Print dataframe

## Filename Class ShortName Color  
## 1 GSM140827.CEL WT 27\_WT red  
## 2 GSM140863.CEL WT 63\_WT red  
## 3 GSM140864.CEL OE 64\_OE blue  
## 4 GSM140865.CEL OE 65\_OE blue

# Create object AnnotatedDataFrame with targets  
targets = AnnotatedDataFrame(targetsDF)  
  
# Save some variables to plot  
sampleNames = as.character(targetsDF$ShortName)  
sampleColor = as.character(targetsDF$Color)

# Código 2  
  
# BiocManager::install("affy")  
library(oligo)

## Loading required package: oligoClasses

## Warning: replacing previous import 'S4Arrays::read\_block' by  
## 'DelayedArray::read\_block' when loading 'SummarizedExperiment'

## Welcome to oligoClasses version 1.62.0

## Loading required package: Biostrings

## Loading required package: S4Vectors

## Loading required package: stats4

##   
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:utils':  
##   
## findMatches

## The following objects are masked from 'package:base':  
##   
## expand.grid, I, unname

## Loading required package: IRanges

##   
## Attaching package: 'IRanges'

## The following object is masked from 'package:grDevices':  
##   
## windows

## Loading required package: XVector

## Loading required package: GenomeInfoDb

##   
## Attaching package: 'Biostrings'

## The following object is masked from 'package:base':  
##   
## strsplit

## ================================================================================

## Welcome to oligo version 1.64.1

## ================================================================================

# Set file paths  
setwd("D:/Antiguos estudios/MASTER2/Sem2/Ómica/PAC2/OMICA-2")  
workingDir = getwd()  
celfilesDir = file.path(workingDir,"celfiles")  
  
# Load rawData  
rawData = read.celfiles(filenames=file.path(celfilesDir, targetsDF$Filename), phenoData = targets)

## Loading required package: pd.moe430a

## Loading required package: RSQLite

## Loading required package: DBI

## Platform design info loaded.

## Reading in : D:/Antiguos estudios/MASTER2/Sem2/Ómica/PAC2/OMICA-2/celfiles/GSM140827.CEL  
## Reading in : D:/Antiguos estudios/MASTER2/Sem2/Ómica/PAC2/OMICA-2/celfiles/GSM140863.CEL  
## Reading in : D:/Antiguos estudios/MASTER2/Sem2/Ómica/PAC2/OMICA-2/celfiles/GSM140864.CEL  
## Reading in : D:/Antiguos estudios/MASTER2/Sem2/Ómica/PAC2/OMICA-2/celfiles/GSM140865.CEL

## Warning in read.celfiles(filenames = file.path(celfilesDir,  
## targetsDF$Filename), : 'channel' automatically added to varMetadata in  
## phenoData.

print(rawData)

## ExpressionFeatureSet (storageMode: lockedEnvironment)  
## assayData: 506944 features, 4 samples   
## element names: exprs   
## protocolData  
## rowNames: 1 2 3 4  
## varLabels: exprs dates  
## varMetadata: labelDescription channel  
## phenoData  
## rowNames: 1 2 3 4  
## varLabels: Filename Class ShortName Color  
## varMetadata: labelDescription channel  
## featureData: none  
## experimentData: use 'experimentData(object)'  
## Annotation: pd.moe430a

### Exploración y control de calidad

Antes de realizar el análisis de expresión diferencial, se llevó a cabo un riguroso control de calidad para asegurar la fiabilidad de los datos obtenidos a partir de los microarrays. Se utilizaron diversas herramientas y técnicas para evaluar la calidad global de los datos y detectar posibles problemas técnicos o artefactos que pudieran afectar los resultados del análisis.

Se generó un histograma para visualizar la distribución de las intensidades de fluorescencia de las sondas en los microarrays. Esto permitió evaluar la simetría y la forma de la distribución de los datos, así como identificar posibles valores atípicos (**Código 3**). Además, se construyeron boxplots para examinar la distribución de las intensidades de fluorescencia en función de las diferentes muestras (**Código 4**). Estos gráficos proporcionaron información sobre la variabilidad entre las muestras y permitieron detectar posibles muestras que se desviaran significativamente de las demás.

Se realizó un análisis de componentes principales para evaluar la variabilidad general de los datos y detectar patrones de agrupación o separación entre las muestras (**Código 5**). El ACP permitió reducir la dimensionalidad de los datos y visualizar la estructura global de los datos en un espacio de menor dimensión. Esto facilitó la identificación de posibles efectos de confusión o agrupación no deseados en los datos.

Se llevó a cabo un análisis de cluster jerárquico para examinar la similitud o agrupación entre las muestras en función de sus perfiles de expresión génica (**Código 6**). Se utilizó la distancia euclidiana y el método de agrupación promedio (“average”) para construir un dendrograma que representara las relaciones de similitud entre las muestras. Esto permitió identificar posibles agrupaciones o subgrupos de muestras con perfiles de expresión similares o diferentes.

Finalmente, se utilizó el paquete “ArrayQualityMetrics” en R para realizar un control de calidad exhaustivo y automatizado de los datos de microarrays que complementó los análisis mencionados anteriormente (**Código 7**).

Como veremos a continuación, tras llevar a cabo el control de calidad de los datos, no se detectaron problemas significativos que pudieran comprometer la fiabilidad de los resultados. Por lo tanto, se decidió realizar el análisis de calidad antes de la normalización y no se consideró necesario repetirlo posteriormente.

### Pre-procesamiento

Los datos de expresión génica obtenidos a partir de los microarrays se sometieron a un proceso de normalización para ajustar las diferencias sistemáticas entre las muestras y garantizar una comparabilidad adecuada. Se utilizó el método de normalización Robust Multi-array Average (RMA) mediante la función “rma” en R, aplicada a los datos brutos de los microarrays (“rawData”) (**Código 8**).

Para el filtrado de genes, se utilizó el paquete “genefilter” en R. Primero, se asignó el tipo de anotación de genes mediante la función “annotation”, estableciéndolo como “moe430a”. Esto permitió la asociación de las sondas de expresión con las anotaciones de genes correspondientes. A continuación, se aplicó un filtro basado en la variabilidad de la expresión génica utilizando la función “nsFilter”. Se utilizó el Rango Intercuartílico (IQR) como medida de variabilidad y se estableció un umbral de corte del 75% para seleccionar el 25% de los genes con mayor variabilidad (**Código 9**).

Con estos métodos de preprocesamiento, se logró la normalización de los datos brutos y el filtrado de genes con el objetivo de obtener un conjunto de datos de expresión génica de mayor calidad y relevancia para el análisis de expresión diferencial entre la línea celular con sobreexpresión de Nmyc y la línea Wild type.

### Análisis

Para llevar a cabo el análisis de expresión diferencial entre la línea celular Wild type y la línea con sobreexpresión de Nmyc, se construyó una matriz de diseño que refleja las condiciones experimentales de cada muestra (**Código 10**). Para nuestro estudio, se dispuso de un total de 4 muestras, lo que implicó la construcción de una matriz de diseño con 4 filas. Dado que nuestro factor experimental se basó en el tipo de línea celular, el cual tiene dos opciones posibles (WT u OE), la matriz de diseño se configuró con 2 columnas para representar estas condiciones experimentales distintas.

Además, se creó una matriz de contrastes que especifica la comparación de interés en términos de los coeficientes del modelo lineal (**Código 11**). En este caso, se estableció un contraste entre Wild type y sobreexpresado, lo que permitirá identificar los genes que muestran una expresión diferencial significativa entre estas dos condiciones.

Para realizar el análisis de expresión diferencial, se utilizó la función “lmFit” del paquete de Bioconductor en R. Se aplicó esta función a la matriz de expresión génica normalizada y filtrada (“expression”), utilizando la matriz de diseño y la matriz de contrastes previamente creadas (**Código 12**). La función “lmFit” se encargó de ajustar un modelo lineal a los datos de expresión génica y estimar los coeficientes del modelo asociados con los contrastes de interés. Estos coeficientes proporcionan estimaciones de la diferencia de expresión entre las condiciones Wild type y sobreexpresado para cada gen. La estimación del modelo lineal es un paso esencial para llevar a cabo el análisis de expresión diferencial y detectar los genes que presentan una expresión diferencial significativa entre las condiciones comparadas. Se genera una tabla de resultados que contiene información sobre los genes seleccionados en función del fold change (8) y el valor de p ajustado (0.05).

Seguidamente, se añadieron los identificadores de ENTREZID, SYMBOL y ENSEMBL a los genes resultantes de la tabla (**Código 13**). La anotación de los genes proporciona información relevante sobre la identidad y características de los genes diferencialmente expresados, lo que facilita su interpretación biológica y su posterior análisis funcional.

Finalmente, con el objetivo de visualizar de manera gráfica los resultados del análisis de expresión diferencial, se generó un volcano plot (**Código 14**). Este tipo de gráfico representa los valores de logaritmo en base 2 del fold change en el eje x y los valores de logaritmo en base 10 del valor de p ajustado en el eje y. En el volcano plot, los genes con una expresión significativamente diferencial se resaltan mediante puntos que se sitúan en las regiones superiores y laterales del gráfico, dependiendo de su fold change y su valor de p ajustado.

Adicionalmente, se generó un heatmap para visualizar la expresión diferencial de los genes en las diferentes condiciones experimentales (**Código 15**). En este gráfico, cada fila representa un gen y cada columna representa una muestra. Los valores de expresión de los genes se representan mediante colores, lo que permite identificar patrones de expresión diferencial entre las condiciones.

Con el objetivo de comprender mejor la función biológica de los genes diferencialmente expresados, se llevó a cabo un análisis de sobrerepresentación de categorías GO (Gene Ontology) (**Código 16**). Las categorías GO proporcionan anotaciones funcionales a los genes en términos de procesos biológicos, funciones moleculares y componentes celulares.

# Resultados

# Discusión

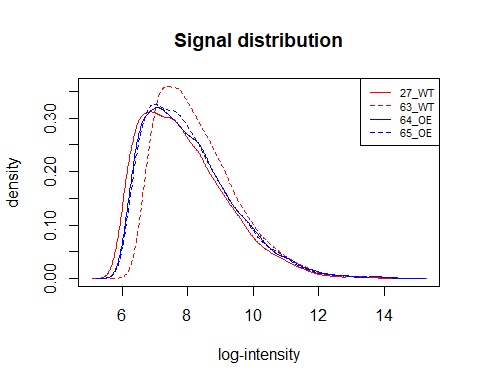
# Referencias

# Apéndice

Como veremos a continuación, no detectamos ningún problema con los datos. Así que tan sólo ejecutaremos el análisis de calidad antes de la normalización y no después.

### Histograma

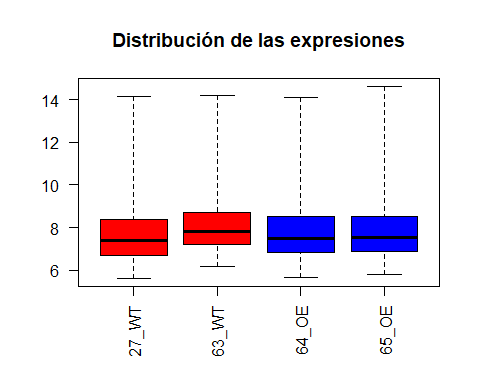
# Código 3  
  
affyLineas <- c(1,2,1,2) # Set line type  
hist(rawData, main="Signal distribution", col=sampleColor, lty=affyLineas)  
legend (x="topright", legend=sampleNames , col=sampleColor, lty=affyLineas, cex=0.7)



Como podemos ver, los 4 arrays dibujan una distribución de similar forma y posición. Esto sugiere que las expresiones no han sido afectadas por ningún aspecto técnico, y que la ligera variación que vemos en GSM140863 se debe a efectos biológicos.

### Boxplot

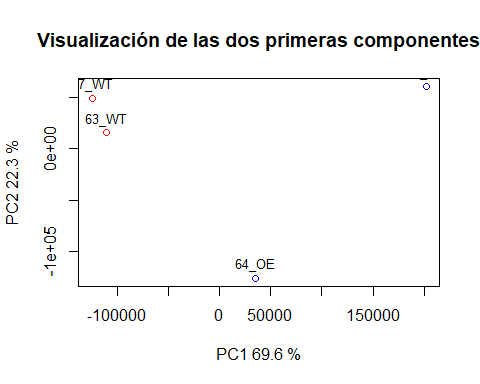
# Código 4  
boxplot(rawData, which = "all", las = 2, main="Distribución de las expresiones", col=sampleColor, names = sampleNames, cel.axis = 0.6)



Nuevamente, el boxplot nos da una idea de la intensidad de la expresión, que es muy similar en todos los arrays.

### Análisis de componentes principales

# Código 5  
  
plotPCA <- function ( X, labels=NULL, colors=NULL, dataDesc="", scale=FALSE)  
{  
 pcX<-prcomp(t(X), scale=scale) # o prcomp(t(X))  
 loads<- round(pcX$sdev^2/sum(pcX$sdev^2)\*100,1)  
 xlab<-c(paste("PC1",loads[1],"%"))  
 ylab<-c(paste("PC2",loads[2],"%"))  
 if (is.null(colors)) colors=1  
 plot(pcX$x[,1:2],xlab=xlab,ylab=ylab, col=colors,   
 xlim=c(min(pcX$x[,1])-10, max(pcX$x[,1])+10),  
 ylim=c(min(pcX$x[,2])-10, max(pcX$x[,2])+10),  
 )  
 text(pcX$x[,1],pcX$x[,2], labels, pos=3, cex=0.8)  
 titulo <- ifelse(dataDesc=="", "Visualización de las dos primeras componentes", dataDesc)  
 title(titulo, cex=0.8)  
}  
  
plotPCA(exprs(rawData), labels=sampleNames, colors = sampleColor)

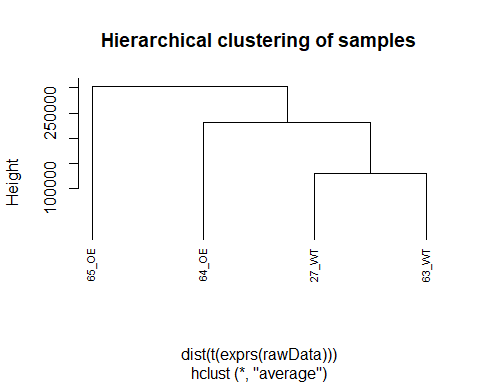


El análisis de componentes principales nos permite detectar si las muestras se agrupan de forma “natural”, ya que éste busca las principales fuentes de variabilidad en los datos. En este caso vemos como las líneas “Wild Type” parecen estar agrupadas, mientras que las que presentan sobreexpresión se alejan del resto. Esto no tiene porque ser indicativo de la existencia de un problema, pero sí deberemos estar al cuidado de un posible efecto batch. Éste sucede cuando se observan diferencias entre muestras que fueron procesadas y analizadas separadamente, podría deberse a problemas técnicos en la hibridación, preparación o escaneo de las muestras. Por este tipo de efectos, es conveniente normalizar las muestras para así eliminar las diferencias sistemáticas entre muestras.

Los axis del gráfico nos muetran que el primer componente explica casi el 70% de la variabilidad de los datos, y el segundo componente el 22%. Este porcentaje es muy elevado, sobretodo para el primer componente, si nos fijamos bien, podríamos dibujar una línea vertical que nos separa los datos, así que con tan solo el primer componente podemos explicar la variabilidad de los datos y separar en las dos categorías los datos.

### Cluster Jerárquico

# Código 6  
clust.euclid.average = hclust(dist(t(exprs(rawData))),method="average")  
plot(clust.euclid.average, labels=sampleNames, main="Hierarchical clustering of samples", hang=-1, cex=0.7)



Al igual que en el análisis de componentes principales, vemos como las muestras Wild Type son más similares entre ellas, mientras que las Sobreexpresadas difieren más. Una posible explicación de ésta variación sería que efecto de las condiciones experimentales afecte tan sólo a un reducido número de genes, o bien que haya algún error en la sobreexpresión de una de las réplicas.

# Control de calidad con ArrayQualityMetrics

# Código 7  
library(arrayQualityMetrics)  
arrayQualityMetrics(rawData, reporttitle = "Calidad\_RawData", force = TRUE)

## The report will be written into directory 'Calidad\_RawData'.

## Warning in svgStyleAttributes(style, svgdev): Removing non-SVG style attribute  
## name(s): subscripts, group.number, group.value  
  
## Warning in svgStyleAttributes(style, svgdev): Removing non-SVG style attribute  
## name(s): subscripts, group.number, group.value  
  
## Warning in svgStyleAttributes(style, svgdev): Removing non-SVG style attribute  
## name(s): subscripts, group.number, group.value  
  
## Warning in svgStyleAttributes(style, svgdev): Removing non-SVG style attribute  
## name(s): subscripts, group.number, group.value

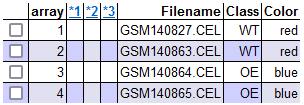
## (loaded the KernSmooth namespace)

## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'

## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'  
  
## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'  
  
## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'  
  
## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'  
  
## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'  
  
## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'  
  
## Warning in KernSmooth::bkde2D(x, gridsize = nbin, bandwidth = bandwidth):  
## Binning grid too coarse for current (small) bandwidth: consider increasing  
## 'gridsize'

El paquete arrayQualityMetrics también nos da información sobre la calidad de las muestras, que es guardada en un directorio “Calidad\_RawData”. Parte de la información que nos muestra ya la hemos analizado mediante el boxplot, histograma, análisis de cluster, etc. Sin embargo, me parece adecuado mencionar la existencia de este paquete e incluir algunos de los análisis que aportan nueva información.

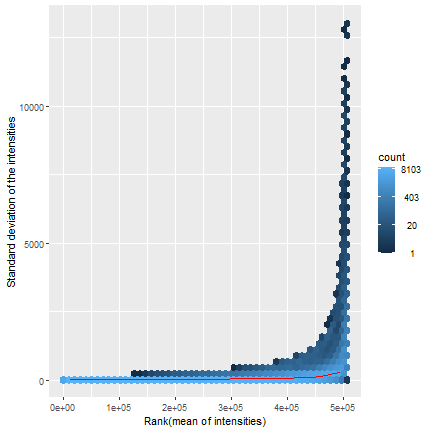
### Outliers



Outliers

No observamos ningún array problemático. Se han realizado detección de outliers tanto por distancias entre arrays, boxplots y Ma plots, y en ninguno de los tres casos se supera el umbral.

### msd

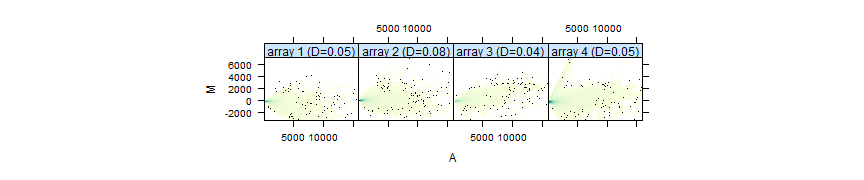


msd

Los gráficos MSD son útiles para estudiar la calidad y variabilidad de los datos. Pueden detectar sesgos técnicos, outliers y efectos de batch. En ellos, el eje Y representa la desviación estándar de la intensidad (medida de la variablidad), y en el X la media de la intensidad (medida de expresión).

En nuestro caso, la gran mayoría de genes presentan poca variabilidad en las muestras y observamos la línea roja prácticamente horizontal. Ciertos genes, sobretodo los de más expresión, sí presentan variabilidad. Esto puede deberse a que al contar con múltiples líneas celulares, en alguna de las líneas haya genes sobreexpresados y por lo tanto observemos variabilidad. Tiene sentido, pues contamos con dos líneas celulares con nmyc sobreexpresado que al ser un factor de transcripción, podría alterar la transcripción de múltiples genes.

### MA-plot

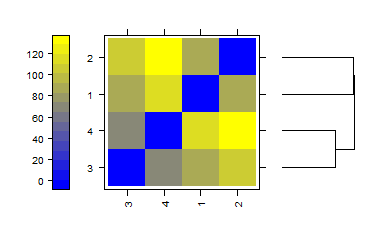


ma

El MA-plot nos permite ver la calidad individual de cada array. En el eje M vemos el la dirección e intensidad del cambio en la expresión génica, mientras que en el eje A vemos la expresión media de cada gen entre las dos condiciones. Este tipo de gráficos son útiles para detectar sesgos sistemáticos o artefactos.

En nuestro caso, no vemos ninguna anormalidad, podemos ver que los genes se distribuyen alrededor de M = 0 sin ver ningún patrón o tendencia clara.

### Heatmap



hm

Heatmap falso del array de expresiones que muestra las distancias entre arrays. No encontramos ninguna anormalidad. En este caso si vemos como las muestras sobreexpresadas son más similares entre ellas en cuanto a expresión que las WT en el cluster jerárquico.

# Normalización

Antes de empezar los análisis de expresión es muy importante normalizar los datos para hacerlos comparables e intentar eliminar la variabilidad que no tenga origen biológico.

# Código 8  
eset = rma(rawData) # Normalize rawData

## Background correcting  
## Normalizing  
## Calculating Expression

print(eset)

## ExpressionSet (storageMode: lockedEnvironment)  
## assayData: 22690 features, 4 samples   
## element names: exprs   
## protocolData  
## rowNames: 1 2 3 4  
## varLabels: exprs dates  
## varMetadata: labelDescription channel  
## phenoData  
## rowNames: 1 2 3 4  
## varLabels: Filename Class ShortName Color  
## varMetadata: labelDescription channel  
## featureData: none  
## experimentData: use 'experimentData(object)'  
## Annotation: pd.moe430a

# Filtrado

Realizaremos un filtrado de genes no específico, eliminando genes por características no relacionadas con el tema del estudio. En específico, eliminaremos genes sin identificador ENTREZID y genes que varían poco usando el rango interquartil quedándonos con el 25% más variable que tienen ENTREZID. El filtrado nos permite reducir el número de genes de estudio y por tanto reducir el número de tests estadísticos a realizar, reduciendo así el error estadístico.

# Código 9  
library(genefilter)  
annotation(eset) = "moe430a" # Set annotation type  
# Filter 25% most changing genes  
eset\_filtered = nsFilter(eset, var.func = IQR, var.cutoff = 0.75, var.filter = TRUE, require.entrez = TRUE, filterByQuantile = TRUE)

##

##

print(eset\_filtered)

## $eset  
## ExpressionSet (storageMode: lockedEnvironment)  
## assayData: 3273 features, 4 samples   
## element names: exprs   
## protocolData  
## rowNames: 1 2 3 4  
## varLabels: exprs dates  
## varMetadata: labelDescription channel  
## phenoData  
## rowNames: 1 2 3 4  
## varLabels: Filename Class ShortName Color  
## varMetadata: labelDescription channel  
## featureData: none  
## experimentData: use 'experimentData(object)'  
## Annotation: moe430a   
##   
## $filter.log  
## $filter.log$numDupsRemoved  
## [1] 8298  
##   
## $filter.log$numLowVar  
## [1] 9818  
##   
## $filter.log$numRemoved.ENTREZID  
## [1] 1288  
##   
## $filter.log$feature.exclude  
## [1] 13

# Extract matrix of filtered data  
filteredEset = eset\_filtered$eset  
expressionSet = exprs(filteredEset)  
colnames(expressionSet) = pData(filteredEset)$ShortName

En este caso vemos como se han eliminado 8298 valores repetidos, 9818 por baja variabilidad y 1288 por falta de identificador ENTREZ. Ahora contamos con 3273 genes.

# Análisis

### Matriz de diseño

La matriz de diseño es una tabla que contiene la información de cada muestra con respecto a su grupo experimental. En nuestro caso tenemos 4 muestras así que contaremos con 4 filas, y sólo tenemos un factor experimental (tipo de línea celular) que tiene dos posibles opciones (WT o OE), así que tendremos 2 columnas.

# Código 10  
  
library(limma)

##   
## Attaching package: 'limma'

## The following object is masked from 'package:oligo':  
##   
## backgroundCorrect

## The following object is masked from 'package:BiocGenerics':  
##   
## plotMA

# Create design matrix  
treat = pData(filteredEset)$Class  
lev = factor(treat, levels = unique(treat))  
designMat<- model.matrix(~0+lev)  
colnames(designMat) = levels(lev)  
rownames(designMat) = sampleNames  
print(designMat)

## WT OE  
## 27\_WT 1 0  
## 63\_WT 1 0  
## 64\_OE 0 1  
## 65\_OE 0 1  
## attr(,"assign")  
## [1] 1 1  
## attr(,"contrasts")  
## attr(,"contrasts")$lev  
## [1] "contr.treatment"

### Matriz de contrastes

La matriz de contrastes describe las comparaciones entre grupos que queremos realizar. Tiene tantas columnas como comparaciones y tantas filas como grupos. Cada comparación o contraste se representa por un 1 y un -1 en las filas a comparar.

En nuestro caso tenemos un diseño muy simple, queremos comparar los efectos de sobreexpresar nmyc, así que compararemmos WT vs OE.

# Código 11  
  
# Create matrix contrast  
cont.matrix = makeContrasts(OEvsWT = OE-WT, levels = designMat)  
comparisonName = "Efecto de la sobreexpresion"  
print(cont.matrix)

## Contrasts  
## Levels OEvsWT  
## WT -1  
## OE 1

### Estimación del modelo

# Código 12  
  
# Fit model  
fit = lmFit(expressionSet, designMat)  
fit.main = contrasts.fit(fit, cont.matrix)  
fit.main = eBayes(fit.main)  
  
# Extract top genes  
topTab = topTable(fit.main, number = nrow(fit.main), coef = "OEvsWT", adjust = "fdr", lfc = 3, p.value = 0.05)  
  
dim(topTab)

## [1] 15 6

head(topTab)

## logFC AveExpr t P.Value adj.P.Val B  
## 1434278\_at 4.654641 7.945429 31.26917 1.013569e-07 0.0002676103 8.603817  
## 1436936\_s\_at -3.525688 8.650697 -27.53764 2.117990e-07 0.0002676103 8.015027  
## 1419407\_at -3.977768 8.147541 -25.81893 3.076400e-07 0.0002676103 7.699454  
## 1437621\_x\_at 3.185596 7.622592 25.23744 3.510019e-07 0.0002676103 7.585360  
## 1418190\_at -3.040917 7.315273 -24.58112 4.088151e-07 0.0002676103 7.451788  
## 1426438\_at 3.004130 5.138115 21.81163 8.156975e-07 0.0003342442 6.825770

# Anotación de resultados

Extraemos los identificadores de los genes a ENTREZID, SYMBOL y ENSEMBL.

# Código 13  
  
library(moe430a.db)  
# Set annotations in ENTREZ, SYMBOL AND ENSEMBL  
keytypes(moe430a.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"

anotaciones = AnnotationDbi::select(moe430a.db, keys = rownames(expressionSet), columns = c("ENTREZID", "SYMBOL", "ENSEMBL"))

## 'select()' returned 1:many mapping between keys and columns

# Add annotations to toptable  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following object is masked from 'package:AnnotationDbi':  
##   
## select

## The following object is masked from 'package:oligo':  
##   
## summarize

## The following objects are masked from 'package:Biostrings':  
##   
## collapse, intersect, setdiff, setequal, union

## The following object is masked from 'package:GenomeInfoDb':  
##   
## intersect

## The following object is masked from 'package:XVector':  
##   
## slice

## The following objects are masked from 'package:IRanges':  
##   
## collapse, desc, intersect, setdiff, slice, union

## The following objects are masked from 'package:S4Vectors':  
##   
## first, intersect, rename, setdiff, setequal, union

## The following object is masked from 'package:Biobase':  
##   
## combine

## The following objects are masked from 'package:BiocGenerics':  
##   
## combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

topTabAnotada = topTab %>% # Copiamos la topTable  
 mutate(PROBEID = rownames(topTab)) %>%   
 left\_join(anotaciones) %>% # Añadimos las anotaciones  
 arrange(P.Value) %>%  
 select(7:10, 1:6)

## Joining with `by = join\_by(PROBEID)`

head(topTabAnotada)

## PROBEID ENTREZID SYMBOL ENSEMBL logFC AveExpr t  
## 1 1434278\_at 17772 Mtm1 ENSMUSG00000031337 4.654641 7.945429 31.26917  
## 2 1436936\_s\_at 213742 Xist ENSMUSG00000086503 -3.525688 8.650697 -27.53764  
## 3 1419407\_at 15139 Hc ENSMUSG00000026874 -3.977768 8.147541 -25.81893  
## 4 1437621\_x\_at 236539 Phgdh ENSMUSG00000053398 3.185596 7.622592 25.23744  
## 5 1418190\_at 18979 Pon1 ENSMUSG00000002588 -3.040917 7.315273 -24.58112  
## 6 1426438\_at 26900 Ddx3y ENSMUSG00000069045 3.004130 5.138115 21.81163  
## P.Value adj.P.Val B  
## 1 1.013569e-07 0.0002676103 8.603817  
## 2 2.117990e-07 0.0002676103 8.015027  
## 3 3.076400e-07 0.0002676103 7.699454  
## 4 3.510019e-07 0.0002676103 7.585360  
## 5 4.088151e-07 0.0002676103 7.451788  
## 6 8.156975e-07 0.0003342442 6.825770

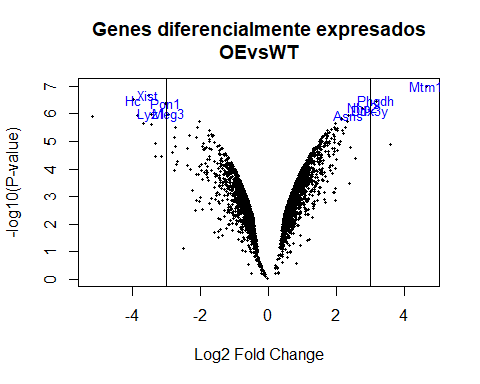
# Resultados

### Volcano plot

# Código 14  
genenames = AnnotationDbi::select(moe430a.db, rownames(fit.main), c("SYMBOL"))$SYMBOL

## 'select()' returned 1:1 mapping between keys and columns

volcanoplot(fit.main, highlight = 10, names = genenames, main = paste("Genes diferencialmente expresados", colnames(cont.matrix), sep = "\n"))  
abline(v = c(-3, 3))



Recordamos que los genes más diferencialmente expresados son aquellos más a los extremos y arriba.

### Heatmap

# Código 15  
# Select rows  
selectedRows = rownames(expressionSet) %in% rownames(topTab)  
selectedData = expressionSet[selectedRows,]  
  
# Heatmap  
library(gplots)

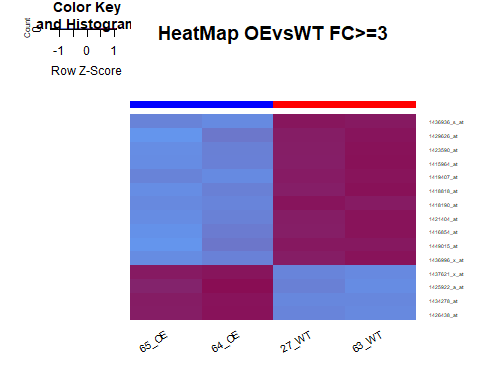
##   
## Attaching package: 'gplots'

## The following object is masked from 'package:IRanges':  
##   
## space

## The following object is masked from 'package:S4Vectors':  
##   
## space

## The following object is masked from 'package:stats':  
##   
## lowess

my\_palette <- colorRampPalette(c("cornflowerblue", "deeppink4"))(n = 299)  
heatmap.2(selectedData,  
 Rowv = TRUE,  
 Colv = TRUE,  
 main = "HeatMap OEvsWT FC>=3",  
 scale = "row",  
 col = my\_palette,  
 sepcolor = "white",  
 sepwidth = c(0.05,0.05),  
 cexRow = 0.5,  
 cexCol = 0.9,  
 key = TRUE,  
 keysize = 1.5,  
 density.info = "histogram",  
 ColSideColors = sampleColor,   
 tracecol = NULL,  
 dendrogram = "none",  
 srtCol = 30)



Cabe destacar que nosotros ya hemos realizado un filtraje de los genes más variables, por ello no es sorpresa que los resultados muestren tantas diferencias entre ambos grupos.

### Análisis de sobrerepresentación

Éste tipo de análisis muestra si entre las anotaciones de los genes, hay alguna categoría de GO que se repita más frecuentemente que si no hubiera expresión diferencial. Esto nos da una idea de a qué proceso biológico estan asociados los genes más variables en el estudio.

# Código 16  
library(moe430a.db)  
# Set probes  
probesUniverse = rownames(expressionSet)  
entrezUniverse = AnnotationDbi::select(moe430a.db, probesUniverse, "ENTREZID")$ENTREZID

## 'select()' returned 1:1 mapping between keys and columns

topProbes = rownames(selectedData)  
entrezTop = AnnotationDbi::select(moe430a.db, topProbes, "ENTREZID")$ENTREZID

## 'select()' returned 1:1 mapping between keys and columns

# Delete duplicates  
topGenes = entrezTop[!duplicated(entrezTop)]  
entrezUniverse = entrezUniverse[!duplicated(entrezUniverse)]  
  
# Use GOstats to analize  
library(GOstats)

## Loading required package: Category

## Loading required package: Matrix

##   
## Attaching package: 'Matrix'

## The following object is masked from 'package:S4Vectors':  
##   
## expand

## Loading required package: graph

##   
## Attaching package: 'graph'

## The following object is masked from 'package:Biostrings':  
##   
## complement

## Warning: replacing previous import 'utils::findMatches' by  
## 'S4Vectors::findMatches' when loading 'AnnotationForge'

##

##   
## Attaching package: 'GOstats'

## The following object is masked from 'package:AnnotationDbi':  
##   
## makeGOGraph

GOparams = new("GOHyperGParams", geneIds = topGenes, universeGeneIds = entrezUniverse, annotation = "moe430a.db", ontology = "BP", pvalueCutoff = 0.01)  
  
GOhyper = hyperGTest(GOparams) # Fisher test  
head(summary(GOhyper))

## GOBPID Pvalue OddsRatio ExpCount Count Size  
## 1 GO:0019835 0.0001062734 265.166667 0.017510944 2 4  
## 2 GO:0043009 0.0012526283 8.333333 0.893058161 5 204  
## 3 GO:0009792 0.0013983153 8.115468 0.914946842 5 209  
## 4 GO:0050830 0.0035645447 27.763158 0.091932458 2 21  
## 5 GO:0046486 0.0037448337 11.787879 0.328330206 3 75  
## 6 GO:0006566 0.0043777361 Inf 0.004377736 1 1  
## Term  
## 1 cytolysis  
## 2 chordate embryonic development  
## 3 embryo development ending in birth or egg hatching  
## 4 defense response to Gram-positive bacterium  
## 5 glycerolipid metabolic process  
## 6 threonine metabolic process

dim(summary(GOhyper))

## [1] 19 7

Como podemos ver, el parámetro “<GO:0019835>” se encuentra 265 veces más (folds) de las esperadas. Éste corresponde a citolisis.

# Random BS GO

<https://support.bioconductor.org/p/41945/> <https://support.bioconductor.org/p/64888/> <https://support.bioconductor.org/p/46513/> <https://rdrr.io/bioc/Biobase/man/phenoData.html> <https://support.bioconductor.org/p/64888/>

Ultimate <https://aspteaching.github.io/Analisis_de_datos_omicos-Materiales_para_un_curso/exploraci%C3%B3n-de-los-datos-control-de-calidad-y-preprocesado.html>