Script análisis microARNs TFM

Belén Mollá Moliner

21/06/2021

Contents

Resumen de los pasos ejecutados en el pipeline para el análisis de small RNA-seq de la empresa EpiDisease.

La primera parte del pipeline que comprende el manejo de los archivos fasq, el filtrado, el análisis de calidad y el mapeo está codificado principalmente en LINUX y pyhton, por ello vamos a ejecutar los chunks llamando a bash desde R. Ha sido necesaria la instalación de una máquina virtual (VMware workstation 16 player) para crear un entorno LINUX (Ubuntu 20.04 LTS) para ejecutar bash y python desde R y RMnd.

La segunda parte del pipeline que comprende el análisis de expresión diferencial y la anotación de los microRNAs está codificada en R.

1 Unión de 4 fastqc por muestra

Se muestra el código para analizar una muestra nombrada como **Pac 2 o 1673**. Unimos los 4 fastq de una misma muestras (L001, L002, L003 y L004) en un archivo fastq descomprimido con la función de bash *cat*:

```
> cat Pac-2-T1-miRNA_S2_L00[1,2,3,4]_R1_001.fastq.gz > 1673.fastq.gz
+ gzip -d 1673.fastq.gz
```

2 Prefiltrado: Análisis de calidad de la muestra con el programa FastQC.

Primero hay que instalar el programa FastQC v0.11.9 (https://www.bioinformatics.babraham.ac.uk/projects/fastqc/). Después hay que llamarlo desde el comando bash y le indicamos el archivo sobre el que queremos hacer el análisis (1673.fastq) y dónde queremos guardar el análisis (Resultados/Pre-filtering_quality)

```
> /home/Downloads/fastqc_v0.11.9/FastQC/fastqc 1673.fastq -o../Resultados/Pre-filtering_quality
```

3 Filtrado

Para el filtrado se ha utilizado la función **cutadapt**, que ha sido instalada desde el terminal de LINUX de R con el comando *sudo apt install cutadapt*. En el proceso de filtrado se va a eliminar el adptador (trimming) y las secuencias con longitud inferior a 18 y calidad inferior a 20:

```
> cutadapt -a TGGAATTCTCGGGTGCCAAGG -m 18 -q 20 -o../Resultados/Processed_data/1673.fastq 1673.fastq
```

4 Postfiltrado: Análisis de calidad de los datos filtrados con el programa FastQC.

```
> /home/Downloads/fastqc_v0.11.9/FastQC/fastqc 1673.fastq -o../Resultados/Post-filtering_quality
```

5 Mapeo de las lecturas sobre el genoma de referencia

Descargamos e instalamos el programa subread 2.0.0 (http://subread.sourceforge.net/). Llamamos al programa desde el comando bash e indicamos las opciones (https://bioconductor.org/packages/release/bioc/vignettes/Rsubread/inst/doc/SubreadUsersGuide.pdf):

subread-align -i GRCh38_index para indicar el genoma de referencia

-t 1 para seleccionar el primer archivo del directorio

-n 45 número de sublecturas extraídas de cada lectura

-m 4 mínmo número de sublecturas consenso requeridas para considerar un punto de alineamiento

-M 3 máximo número de bases mal mapeadas permitidas en el alineamiento

-T 10 map reads with 5 threads

-I 0 número de bases INDEL permitidas en el mapeo (de 0 bp a 200bp)

–**multiMapping -B 10** número de árboles con las mejores localizaciones del mapeo

-r reads.txt -o subread_results.bam archivos de entrada (-r) y de salida (-o)

```
> /home/Downloads/subread-2.0.0/bin/subread-align -i /home/BASESDATOS/GRCh38_index/GRCh38_index
+ -t 1 -n 45 -m 4 -M 3 -T 10 -I 0 --multiMapping -B 10 -r $i -o $path/Resultados/bamfiles/1673.bam;
```

6 Análisis de expresión diferencial con método EdgeR

Cargamos los paquetes que vamos a utilizar a lo largo del procesamiento:

```
> BiocManager::install("Rsubread")
> library(Rsubread)
> library(edgeR)
> library(xlsx)
> library(RColorBrewer)
> library(rJava)
> library(org.Hs.eg.db)
> library(openxlsx)
> library(rtracklayer)
```

6.1 Matriz de conteos

Partimos de los bamfiles del mapeo y del archivo de anotación para crear la matriz de *counts* con la función **featureCounts()** del paquete **RSubred** de R:

```
> featureMatrix <- Rsubread::featureCounts(files = "1673.fastq.bam",
+     annot.ext = "anot_mirnas.RData", allowMultiOverlap = TRUE,
+     countMultiMappingReads = TRUE)
> counts <- featureMatrix$counts
```

6.2 Preparación del metadata

Cargamos el metadata con el nombre de la muestra (P2), el ID de la muestra (1673), el grupo al que pertenece (C=control) y las comparaciones en las que participa (Comp1 y Comp2) y guardamos la columna *groups*:

```
> mdata = read.xlsx(metadata, sheetIndex = 1, as.data.frame = T,
+     header = T)
> groups <- as.character(mdata[-nrow(mdata), 3])
```

6.3 DGE list

Primero filtramos la matriz de conteos y eliminamos todas las filas con 0 conteos. Después con la función **DGEList()** del paquete de R **edgeR**, creamos la *lista DGE* y le añadimos la columna de *groups*:

```
> # filtramos las filas que tienen 0 counts
> counts <- count.table[rowSums(counts) > 0, ]
>
> # creamos la lista DGE
> dge <- edgeR::DGEList(counts = counts)
>
> # añadimos la columna de groups a la lista dge
> dge$samples$group <- groups
```

6.4 Normalización TMM

Para eliminar los sesgos de composición (*bias composition*) entre las bibliotecas, se ha procedido a la normalización de los datos con el método TMM. La función **calcNormFactors()** calcula los factores de normalizaciÓn entre bibliotecas definiendo el tamaño efectivo de la biblioteca (*effective library size*).

```
> # normalización
> tmm <- calcNormFactors(dge)
```

6.5 Normalización cpm y filtrado de los datos atípicos

Ahora los datos se normalizan por la profundidad de sescuenciación de cada muestra con la función **cpm()** del paquete **edgeR** de R y se transforman a escala log2. Además se hace un filtrado de valores atípicos, reteniendo sólo los genes que se expresan más de 4 conteos en escala log2 (16 conteos) en al menos el 20% del total de las muestras (4 de 22 muestras):

```
> # recuento de las lecturas en escala log2
> cpmnorm <- cpm(tmm, log = T)
> thresh <- cpmnorm > 4
> table(rowSums(thresh))
> keep <- rowSums(thresh) >= 4
> counts.tmm <- count.table[keep, ]
```

6.6 Análisis de los componentes principales (PCA) antes y después de la normalización y filtrado de outliers

Ahora vamos a visualizar el gráfico de escalado multidimensional (MDS) de las muestras agrupadas en 3 grupos antes y después de la normalización:

```
> col <- c("green", "orange", "purple")[as.factor(groups)]
> plotMDS(counts, col = col)
> legend("topright", fill = c("green", "orange", "purple"), legend = c("C",
+     "SN", "SV"), cex = 1)
> title("MDS Plot before normalization")
>
> plotMDS(counts.tmm.keep, col = col)
> legend("topright", fill = c("green", "orange", "purple"), legend = c("C",
+     "SN", "SV"), cex = 1)
> title("MDS Plot after normalization")
```

6.7 Matriz de diseño

Se define la matriz de diseño en la que cada columna nos remite a las muestras que corresponden a cada grupo:

```
> groups
> # Specify a design matrix without an intercept term
> design <- model.matrix(~0 + groups)
> design
>
> ## Make the column names of the design matrix a bit nicer
> colnames(design) <- levels(groups)
> design
```

6.8 Matriz de contrastes

Ahora definimos la matriz de contrastes indicando las 3 comparaciones entre grupos:

```
> cont.matrix <- makeContrasts(1= C - SV, 2 = C - SN, 3=SN-SV,4=S-C,levels=design)
> cont.matrix
```

6.9 Análisis de expresión diferencial con el paquete EdgeR de R

EdgeR implementa la estrategia de Bayes para estimar la dispersión binomial negativa. El modelo lineal lo ajustamos con la función **glmQLFit()** (FIT QUASI-LIKELIHOOD GENERALIZED LINEAR MODELS)

```
> edgeRTest <- estimateDisp(counts.tmm, design, robust = TRUE)
>
> # QLFTest method
> fit_E <- glmQLFit(edgeRTest, design, robust = TRUE)
>
> # hacemos una comparación por cada uno de los grupos
> qlf_1 <- glmQLFTest(fit_E, coef = 1, contrast = cont.matrix)
> qlf_2 <- glmQLFTest(fit_E, coef = 2, contrast = cont.matrix)
> qlf_3 <- glmQLFTest(fit_E, coef = 3, contrast = cont.matrix)
> qlf_4 <- glmQLFTest(fit_E, coef = 4, contrast = cont.matrix)
>
> # tabla con los genes expresados diferencialmente por cada
+ # una de las comparaciones entre grupos
> edge_1 = topTags(qlf_1, n = Inf, sort.by = "PValue", p.value = 1,
+     adjust.method = "fdr")
> edge_2 = topTags(qlf_2, n = Inf, sort.by = "PValue", p.value = 1,
+     adjust.method = "fdr")
> edge_3 = topTags(qlf_3, n = Inf, sort.by = "PValue", p.value = 1,
+     adjust.method = "fdr")
> edge_4 = topTags(qlf_4, n = Inf, sort.by = "PValue", p.value = 1,
+     adjust.method = "fdr")
> write.csv(edge_1, file = "edge_CvsSV.csv", row.names = FALSE)
> write.csv(edge_2, file = "edge_CvsSN.csv", row.names = FALSE)
> write.csv(edge_3, file = "edge_SNvsSV.csv", row.names = FALSE)
> write.csv(edge_4, file = "edge_SvsC.csv", row.names = FALSE)
```

6.10 Anotación de los genes con el nombre del gen y el código EntrezID

```
> # anotación de las secuencias con expresión diferencial
+ # significativas
> ann_limma_1_sig <- select(hsa_MTI_v21.csv, keys = rownames(limma_1_sig),
+     keytype = "ENSEMBL", columns = c("Target.Gene", "Entrez.Gene.ID."))
> # ... así con los 8 archivos restantes de las distintas
+ # comparaciones y métodos ...
```

6.11 Seleccionamos los genes infraexpresados y sobreexpresados para presentarlo en dos listas separadas:

```
> # seleccionamos por el cambio de expresión logFC
> ann_limma_1_UP = subset(ann_limma_1_sig, logFC > 0)
> ann_limma_1_DN = subset(ann_limma_1_sig, logFC < 0)
> # ... así con los 8 archivos restantes de las distintas
+ # comparaciones y métodos
```

7 ANOTO chr, start, end y strand de los mirnas PRIMARY

```
> # SEL <- read.csv("edge_comp1.csv", fileEncoding = "utf-8", header = T, stringsAsFactors = F, skipNul
>
> # cargamos todas las librerías
> library(openxlsx)
> library(rtracklayer)
```

```
>
> #library(rtracklayer) cargamos la base de datos de mirBase
> hsa_MTI=readGFF("hsa.gff3")
> head(hsa_MTI)
>
> # abrir cada una de las comparaciones de resultados que queremos analizar
> edgeR<-read.xlsx("./edgeR_all_mirnome.xlsx", sheet = "Comp4")
> SEL=subset(edgeR, edgeR$FDR <= 0.1)# seleccionar la significación
> rownames(SEL) <- SEL[,1]
> write.xlsx(SEL, file = "SEL_comp4_primary.xlsx")
> SEL<-read.xlsx("SEL_comp4_primary.xlsx", rowNames = TRUE)
>
>
> # anotamos Chr, Start, end y strand con hg38
> for (j in rownames(SEL)){
>     if(j %in% hsa_MTI$Name){
>       SEL[j , "Chr"] <-  paste(hsa_MTI$seqid[hsa_MTI$Name == j], collapse=",")
>       SEL[j , "Start"] <-  paste(hsa_MTI$start[hsa_MTI$Name == j], collapse=",")
>       SEL[j , "End"] <-  paste(hsa_MTI$end[hsa_MTI$Name == j], collapse=",")
>       SEL[j , "Strand"] <-  paste(hsa_MTI$strand[hsa_MTI$Name == j], collapse=",")
>     }
>   }
>
>
> RES=list()
> RES$sin_posicion=SEL
>
> CHR <- list()
>     for (i in {1:length(RES$sin_posicion$Chr)}) {
>       names=data.frame(rownames(RES$sin_posicion))[i,]
>       logFC <- data.frame(RES$sin_posicion$logFC)[i,]
>       FDR <- data.frame(RES$sin_posicion$FDR)[i,]
>       chr <- unlist(strsplit(RES$sin_posicion$Chr[i], ","))
>       if (!is.na(chr)) {
>         for (c in chr) {
>           r <- data.frame(Chr = c)
>           CHR$sin_posicion$all <- rbind(CHR$sin_posicion$all, r)
>         }
>       }
>     }
> START <- list()
>     for (i in {1:length(RES$sin_posicion$Start)}) {
>       start <- unlist(strsplit(RES$sin_posicion$Start[i], ","))
>       if (!is.na(start)) {
>         for (s in start) {
>           r <- data.frame(Start = s)
>           START$sin_posicion$all <- rbind(START$sin_posicion$all, r)
>         }
>       }
>     }
> END <- list()
>     for (i in {1:length(RES$sin_posicion$End)}) {
>       names=data.frame(rownames(RES$sin_posicion))[i,]
```

```
>       logFC <- data.frame(RES$sin_posicion$logFC)[i,]
>       FDR <- data.frame(RES$sin_posicion$FDR)[i,]
>       end <- unlist(strsplit(RES$sin_posicion$End[i], ","))
>       if (!is.na(end)) {
>         for (e in end) {
>           r <- data.frame(End = e, mirna=names, logFC=logFC,FDR=FDR)
>           END$sin_posicion$all <- rbind(END$sin_posicion$all, r)
>         }
>       }
>     }
>
> STRAND <- list()
>     for (i in {1:length(RES$sin_posicion$Strand)}) {
>       strand <- unlist(strsplit(RES$sin_posicion$Strand[i], ","))
>       if (!is.na(strand)) {
>         for (t in strand) {
>           r <- data.frame(Strand = t)
>           STRAND$sin_posicion$all <- rbind(STRAND$sin_posicion$all, r)
>         }
>       }
>     }
>
> df=data.frame(CHR$sin_posicion$all,START$sin_posicion$all,END$sin_posicion$all,STRAND$sin_posicion$al
> df$Start=as.numeric(df$Start)
> df$End=as.numeric(df$End)
> str(df)
> write.csv(df, file = "mirna_comp4_hg38_primary.csv")
>
> # Diagramas de Venn
>
> https://mkempenaar.github.io/gene_expression_analysis/chapter-5.html
>
> Sólo necesito los listados de los nombres en cada comparación
```

```
> library(openxlsx)
> comp1 <- read.xlsx("./edgeR_all_mirnome.xlsx", sheet = "Comp1",
+     rowNames = TRUE)
> comp1_names <- row.names(subset(comp1, comp1$FDR <= 0.1))
> comp1_names  #7
>
>
> comp2 <- read.xlsx("./edgeR_all_mirnome.xlsx", sheet = "Comp2",
+     rowNames = TRUE)
> comp2_names <- row.names(subset(comp2, comp2$FDR <= 0.1))
> comp2_names  #83
>
> comp3 <- read.xlsx("./edgeR_all_mirnome.xlsx", sheet = "Comp3",
+     rowNames = TRUE)
> comp3_names <- row.names(subset(comp3, comp3$FDR <= 0.1))
> comp3_names  #90
>
>
> comp4 <- read.xlsx("./edgeR_all_mirnome.xlsx", sheet = "Comp4",
+     rowNames = TRUE)
```

```
> comp4_names <- row.names(subset(comp4, comp4$FDR <= 0.1))
> comp4_names  #23
>
>
> # diagrama de venn con 2 comparaciones
> library(VennDiagram)
> # Calculate the intersection of the two sets
> deg.intersect = length(intersect(comp1_names, comp2_names))
> deg.venn <- list(intersect = deg.intersect, comp1 = length(comp1_names),
+     comp2 = length(comp2_names))
> venn.plot <- draw.pairwise.venn(deg.venn$comp1, deg.venn$comp2,
+     deg.venn$intersect, category = c("comp1", "comp2"), scaled = F,
+     fill = c("light blue", "pink"), alpha = rep(0.5, 2), cat.pos = c(0,
+         0))
> grid.draw(venn.plot)
>
> # diagrama de venn con múltiples comparaciones
> library(gplots)
> png(paste("VENN_mirnoma_0.1_SEPSIS.png", sep = ""), width = 20,
+     height = 20, res = 300, units = "cm")
> venn(list(EOS = comp1_names, LOS = comp2_names, SEPSIS = comp4_names))
> dev.off()
>
> # extraemos el listado de mirnas comunes en EOS, LOS y
+ # SEPSIS https://felixfan.github.io/Venn/
>
> universe <- unique(c(comp1_names, comp2_names, comp4_names))
> comp1_names.l <- universe %in% comp1_names
> comp1_names.l
> comp2_names.l <- universe %in% comp2_names
> comp3_names.l <- universe %in% comp4_names
> universe[comp1_names.l & comp2_names.l & comp3_names.l]
>
> # funcion para testar en el listado de nombres directamente
+ # los mirnas comunes en EOS, LOS y SEPSIS
> test_sepsis <- function(x) (x %in% comp1_names) & (x %in% comp2_names) &
+     (x %in% comp4_names)
> universe[test_sepsis(universe)]
>
> # EOS, LOS y EOS_LOS
> png(paste("VENN_mirnoma_0.1_EOS_LOS.png", sep = ""), width = 20,
+     height = 20, res = 300, units = "cm")
> venn(list(EOS = comp1_names, LOS = comp2_names, LOSvsEOS = comp3_names))
> dev.off()
>
> # funcion para testar en el listado de nombres de los
+ # mirnas de EOS
> test_EOS <- function(x) (x %in% comp1_names) & (x %in% comp3_names) &
+     !(x %in% comp2_names) & !(x %in% comp4_names)
> universe[test_EOS(universe)]
>
> # funcion para testar en el listado de nombres de los
+ # mirnas de LOS
```

```
> test_LOS <- function(x) (x %in% comp2_names) & (x %in% comp3_names) &
+     !(x %in% comp1_names) & !(x %in% comp4_names)
> universe[test_LOS(universe)]
```