Análisis 16s rRNA utilizando DADA2

Tu nombre

2023-10-23

Contents

Chapter 1

Introducción a DADA2

Repositorio de GitHub aquí

Cita:

> Callahan, B. J., McMurdie, P. J., Rosen, M. J., Han, A. W., Johnson, A. J. A., y Holmes, S. P. (2016). DADA2: High-resolution sample inference from Illumina amplicon data. Nature methods, 13(7), 581-583.

Descripción: DADA2 es un paquete de software de código abierto utilizado para modelar y corregir errores en secuencias de amplicones secuenciados con el protocolo Illumina. Infiere secuencias de una muestra de manera exacta y resuelve diferencias con una resolución de un nucleótido.

Los datos utilizados pueden ser consultados en el European Nucleotide Archive (ENA) bajo el nombre de proyecto PRJNA428495.

Descargamos los archivos FASTQ en la sección "Generated FASTQ files: FTP" y los guardamos en una carpeta.

Los binarios del paquete DADA2 están disponibles a través de Bioconductor

1.1 Instalación de DADA2 en R (versión 4.3)

Este bloque instala el paquete DADA2 a través de Bioconductor. Es importante tener instalado BiocManager para poder acceder a los paquetes de Bioconductor.

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

BiocManager::install("dada2")
library(dada2)
```

1.2 Configuramos el directorio en el que se ubican las reads

```
path <- "/Users/" # Cambiamos al directorio que contiene las reads
list.files(path)
```

1.3 Ordenamos los archivos y extraemos el nombre de las muestras

Este bloque ordena los archivos FASTQ y extrae el nombre de las muestras. Asegúrate de que los patrones de búsqueda (pattern="_1.fastq.gz" y pattern="_2.fastq.gz") coincidan con la nomenclatura de tus archivos.

```
fnFs <- sort(list.files(path, pattern="_1.fastq.gz"))
fnRs <- sort(list.files(path, pattern="_2.fastq.gz"))
sample.names <- sapply(strsplit(fnFs, "_"), '[', 1)
fnFs <- file.path(path, fnFs)
fnRs <- file.path(path, fnRs)
```

1.4 Establecer ruta para archivos filtrados

Define una ruta basada en la variable `sample.names` donde guardar los archivos filtrados. El código genera nombres de archivo con un sufijo ("_F_filt.fastq.gz" o "_R_filt.fastq.gz") para diferenciarlos.

```
filt_path <- file.path(path, "filtered")
filtFs <- file.path(filt_path, paste0(sample.names, "_F_filt.fastq.gz"))
filtRs <- file.path(filt_path, paste0(sample.names, "_R_filt.fastq.gz"))
print(fnFs)
print(fnRs)
print(filtFs)
print(filtRs)
```

1.5 Filtrar y recortar lecturas

La función filterAndTrim se utiliza para el filtrado y recorte de las lecturas, es decir, se eliminan las lecturas de baja calidad y las recorta a una longitud específica. Se especifican varios parámetros, como las longitudes de truncamiento, la calidad máxima permitida, la eliminación de secuencias de fagos (`rm.phix`), la compresión y el uso de múltiples hilos (`multithread`). El resultado se almacena en la variable `out`. Windows 10 permite el 'multi-theading', excepto en el comando `filterAndTrim`.

```
out <- filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen=c(250, 200), maxN=0, maxEE=c(2,2), trun
```

1.6 Estimación de la tasa de error

Se estima la tasa de error para las lecturas adelante (errF) y las lecturas reversas (errR) utilizando la función `learnErrors`. La opción `multithread=TRUE` se utiliza para acelerar el proceso.

```
errF <- learnErrors(filtFs, multithread=TRUE)
errR <- learnErrors(filtRs, multithread=TRUE)
```

1.7 Dereplicar lecturas

Se realiza la eliminación de duplicados de lecturas (dereplicación) para las lecturas adelante (derepFs) y las lecturas reversas (derepRs) utilizando la función `derepFastq`. Se asignan nombres a las muestras con `sample.names`.

```
derepFs <- derepFastq(filtFs, verbose=TRUE)
derepRs <- derepFastq(filtRs, verbose=TRUE)
names(derepFs) <- sample.names
names(derepRs) <- sample.names
```

1.8 Inferencia de secuencias

Se utilizan las tasas de error estimadas en el paso anterior para la inferencia de secuencias únicas utilizando la función `dada` para las lecturas adelante (dadaFs) y las lecturas reversas (dadaRs).

```
dadaRs <- dada(derepRs, err=errR, multithread=TRUE)
dadaFs <- dada(derepFs, err=errF, multithread=TRUE)
```

1.9 Fusionar lecturas emparejadas

Aquí se fusiona las secuencias emparejadas utilizando la función `mergePairs`. Las secuencias únicas y las lecturas originales se utilizan para la fusión, y los resultados se almacenan en la variable mergers.

```
mergers <- mergePairs(dadaFs, derepFs, dadaRs, derepRs, verbose=TRUE)
```

1.10 Crear tabla de secuencias

Se crea una tabla de secuencias utilizando la función `makeSequenceTable`. Esta tabla contendrá información sobre las secuencias y su abundancia.

```
seqtab <- makeSequenceTable(mergers)
```

1.11 Eliminar quimeras

Se realiza la eliminación de quimeras utilizando la función `removeBimeraDenovo`. Se especifica el método de eliminación como "consensus".

```
seqtab.nochim <- removeBimeraDenovo(seqtab, method="consensus", multithread=TRUE, verbo
```

1.12 Asignar taxonomía

Se asigna la taxonomía a las secuencias utilizando la función `assignTaxonomy`. Se proporciona un archivo de referencia para la asignación taxonómica, y se utiliza el `multithread` para acelerar el proceso. La base de datos taxonómica utilizada será un archivo fasta de entrenamiento derivado de la versión 138.1 del Proyecto Silva y formateado para su uso con DADA2. Este archivo puede ser descargado de Zenodo.

```
taxa <- assignTaxonomy(seqtab.nochim, "C:/Users/UBBBC/Desktop/Samuel/PRJNA428495/silva_
```

1.13 Impresión de resultados

Finalmente, se imprime en la consola las primeras filas de la asignación taxonómica resultante.

```
unname(head(taxa))
```