Process Single Cell RNA-Seq reads using *scruff*

Zhe Wang

2017-12-11

`scruff` is a toolkit for processing single cell RNA-seq fastq reads. It does demultiplexing, alignment, Unique Molecular Identifier (UMI) correction, and transcript counting in an automated fashion and outputs expression table. This vignette provides a brief introduction to the `scruff` package by walking through the demultiplexing, alignment, and UMI-counting of a built-in example dataset (*van den Brink, et al. 2017*).

Quick Start

```r
# Run scruff on example dataset
# NOTE: Requires Rsubread index and TxDb objects for reference genome.
# For generation of these files, please refer to Stepwise Tutorial.

library(scruff)

# Export example fastq files
dir.create("./scruff_example/examplefastq",
           showWarnings = FALSE,
           recursive = TRUE)
setwd("./scruff_example")

# write fastg.gz files to folder "examplefastq" in working directory
for (i in seq_len(length(examplefastq))) {
  ShortRead::writeFastq(examplefastq[[i]],
             paste0("examplefastq/", names(examplefastq)[i]),
             mode = "w")
}

# Build Rsubread alignment index
fasta <- "GRCm38_MT.fa"
index_prefix <- "GRCh38_MT"

write.table(GRCm38_MT_fa,
            fasta,
            quote = FALSE,
            sep = "\t",
            row.names = FALSE,
            col.names = TRUE)

# NOTE: Rsubread package does not support Windows environment.
library(Rsubread)
# Create index files for GRCh38_MT. For details, please refer to Rsubread user manual.
buildindex(
  basename = index_prefix,
  reference = fasta,
  indexSplit = FALSE
  )

# Export example gene annotation file
gtf <- "GRCm38_MT.gtf"
write.table(GRCm38_MT_gtf,
            gtf,
            quote = FALSE,
            sep = "\t",
            row.names = FALSE,
            col.names = FALSE)

# Run scruff
# Returns a list of demultiplex information, alignment information,
# and expression tables
result <- scruff(fastq.annot = exampleannot,
                bc = examplebc,
                index = index_prefix,
                features = gtf,
                bc.start = 1,
                bc.stop = 8,
                umi.start = 9,
                umi.stop = 12,
                keep = 75,
                cores = 2)
```

Stepwise Tutorial

Load Example Dataset

The `scruff` package contains a example single cell RNA-seq dataset `examplefastq` which is stored as a list of `ShortReadQ` objects. Each object has 20,000 sequenced reads. They can be accessed via variable `examplefastq` and exported to current working directory as fastq.gz files. `scruff` only accepts fastq/fastq.gz files for demultiplexing so these objects have to be stored in fastq.gz format first.

```
library(scruff)
dir.create("./scruff_example/examplefastq",
           showWarnings = FALSE,
           recursive = TRUE)
setwd("./scruff_example")

# write fastg.gz files to folder "examplefastq" in working directory
for (i in seq_len(length(examplefastq))) {
  ShortRead::writeFastq(examplefastq[[i]],
             paste0("examplefastq/", names(examplefastq)[i]),
             mode = "w")
}
```

Demultiplex and Assign Cell Specific Reads

Now the fastq files are saved in current working directory and are ready to be demultiplexed. `scruff` package provides built-in predetermined sample annotating table `exampleannot` and barcodes `examplebc` for demultiplexing the example dataset. In the example fastq files, the barcode sequence of each read starts at base 6 and ends at base 11. The UMI sequence starts at base 1 and ends at base 5. They can be set via `bc.start`, `bc.stop`, and `umi.start`, `umi.stop` parameters. By default, reads with any nucleotide in the barcode and UMI sequences with sequencing quality lower than 10 (Phred score) will be excluded. The following command demultiplexes the example fastq reads and trims reads longer than 50 nucleotides. The command returns a summary `data.table` containing the sample barcodes, filenames, number of reads, sample name, file paths, etc. By default, the cell specific demultiplexed fastq.gz files are stored in `../Demultiplex` folder.

```
de <- demultiplex(exampleannot,
                  examplebc,
                  bc.start = 1,
                  bc.stop = 8,
                  bc.edit = 1,
                  umi.start = 9,
                  umi.stop = 12,
                  keep = 75,
                  min.qual = 10,
                  yield.reads = 1e+06,
                  cores = 2,
                  verbose = TRUE,
                  overwrite = TRUE)
```

Alignment

`scruff` has a wrapper function `align.rsubread` which is a wrapper function to `align` in `Rsubread` package. It aligns the reads to reference sequence index and outputs sequence alignment map files. For demonstration purpose, the built-in mitochondrial DNA sequence from GRCh38 reference assembly `GRCh38_MT` will be used to map the reads. First, a `Rsubread` index for the reference sequence needs to be generated.

```
fasta <- "GRCm38_MT.fa"
index_prefix <- "GRCh38_MT"

# write reference sequence to "./GRCm38_MT.fa".
write.table(GRCm38_MT_fa,
            fasta,
            quote = FALSE,
            sep = "\t",
            row.names = FALSE,
            col.names = TRUE)

# NOTE: Rsubread package does not support Windows environment.
library(Rsubread)
# Create index files for GRCh38_MT. For details, please refer to Rsubread user manual.
buildindex(
  basename = index_prefix,
  reference = fasta,
  indexSplit = FALSE
  )
```

The following command maps the fastq files to GRCh38 mitochondrial reference sequence `GRCh38_MT` and returns the file paths of the generated sequence alignment map files. By default, the files are stored in BAM format in `../Demultiplex` folder.

```r
# get the paths to demultiplexed fastq.gz files
fastq.paths <- de[!(is.na(cell_num)), fastq_path]

# map the reads
al <- align.rsubread(
  fastq.paths,
  index_prefix,
  format = "BAM",
  cores = 2,
  overwrite = TRUE,
  verbose = TRUE
  )
```

UMI correction and Generation of Expression Table

Example GTF file `GRCh38_MT_gtf` will be used for feature counting. Currently, `scruff` applies the union counting mode of the HTSeq Python package. The following command generates the UMI corrected expression table for the example dataset.

```r
# first write example GTF file to working directory
gtf <- "GRCm38_MT.gtf"
write.table(GRCm38_MT_gtf,
            gtf,
            quote = FALSE,
            sep = "\t",
            row.names = FALSE,
            col.names = FALSE)

co = count.umi(
  al$Samples,
  gtf,
  format = "BAM",
  cores = 2,
  verbose = TRUE
  )
```

Collect gene annotation information from Biomart

Gene annotation information is needed before the visualization of data quality. In this tutorial, Ensembl Biomart database is used for the query and collection of gene annotation information. The following codes retrieve mouse gene names, biotypes, and chromosome names from latest version of biomart database.

```r
# get gene names, biotypes, and chromosome names from biomart
host = "www.ensembl.org"
biomart = "ENSEMBL_MART_ENSEMBL"
#dataset = "hsapiens_gene_ensembl"
dataset = "mmusculus_gene_ensembl"

# get gene IDs
# exclude ERCC spike-ins and last two rows in expression table
geneID <- co[!(gene.id %in% c("reads_mapped_to_genome",
                                "reads_mapped_to_genes")) &
                !grepl("ERCC", co[,gene.id]), gene.id]

# Get feature annotation data
ensembl <- biomaRt::useEnsembl(biomart = biomart,
                               dataset = dataset,
                               host = host)

biomart.result <- data.table::data.table(biomaRt::getBM(
  attributes = c("ensembl_gene_id", "external_gene_name",
                 "gene_biotype", "chromosome_name"),
  filters = "ensembl_gene_id",
  values = geneID,
  mart = ensembl))

# remove rows containing duplicate gene IDs
biomart.result <- biomart.result[!base::duplicated(biomart.result,
                                                   by = "ensembl_gene_id"), ]

# Reorder/insert rows so that rows of Biomart query result match
# rows of the expression matrix
biomart.result <- data.table::data.table(biomart.result[
  match(co[!(gene.id %in% c("reads_mapped_to_genome",
                            "reads_mapped_to_genes")) &
             !grepl("ERCC", co[, gene.id]), gene.id],
        biomart.result$ensembl_gene_id),])
```

Visualization of QC metrics

In order to plots data quality, a QC metrics table needs to be generated. `collectqc` function collects and summarizes data quality information from results from previous steps and returns a QC metrics table for plotting purpose.

```r
qc <- collectqc(de, al, co, biomart.result)
```

QC metrics can be visualized by running `qcplot` function.

```r
qcplot(qc)
```