Utilities for handling droplet-based single-cell RNA-seq data

Aaron Lun

4 January 2019

Package

DropletUtils 1.2.2

Contents

1 Introduction

2 Reading in 10X Genomics data
- 2.1 From the UMI count matrix
- 2.2 From the molecule information file

3 Downsampling across batches
- 3.1 From the count matrix
- 3.2 From the reads

4 Computing barcode ranks

5 Detecting empty droplets

6 Removing the effect of barcode swapping

7 Session information

1 Introduction

Droplet-based single-cell RNA sequencing (scRNA-seq) technologies allow researchers to obtain transcriptome-wide expression profiles for thousands of cells at once. Briefly, each cell is encapsulated in a droplet in a oil-water emulsion, along with a bead containing reverse transcription primers with a unique barcode sequence. After reverse transcription inside the droplet, each cell's cDNA is labelled with that barcode (referred to a "cell barcode"). Bursting of the droplets yields a pool of cDNA for library preparation and sequencing. Debarcoding of the sequences can then be performed to obtain the expression profile for each cell.

This package implements some general utilities for handling these data after quantification of expression. In particular, we focus on the 10X Genomics platform, providing functions to load in the matrix of unique molecule identifier (UMI) counts as well as the raw molecule information. Functions are also available for downsampling the UMI count matrix or the raw reads; for distinguishing cells from empty droplets, based on the UMI counts; and to eliminate the effects of barcode swapping on Illumina 4000 sequencing machine.

2 Reading in 10X Genomics data

2.1 From the UMI count matrix

The *CellRanger* pipeline from 10X Genomics will process the raw sequencing data and produce a matrix of UMI counts. Each row of this matrix corresponds to a gene, while each column corresponds to a cell barcode. This is saved in a single directory for each sample, usually named like `<OUTPUT>/outs/filtered_gene_bc_matrices/<GENOME>` [1]. We mock up an example directory below using some simulated data:

[1] If you use the "filtered" matrix, each column corresponds to a putative cell. If you use the "raw" matrix, all barcodes are loaded, and no distinction is made between cells and empty droplets.

```
# To generate the files.
example(write10xCounts, echo=FALSE)
dir.name <- tmpdir
list.files(dir.name)
```

```
## [1] "barcodes.tsv" "genes.tsv"    "matrix.mtx"
```

The `matrix.mtx` file contains the UMI counts, while the other two files contain the cell barcodes and the gene annotation. We can load this into memory using the `read10xCounts` function, which returns a `SingleCellExperiment` object containing all of the relevant information. This includes the barcode sequence for each cell (column), as well as the identifier and symbol for each gene (row).

```
sce <- read10xCounts(dir.name)
sce
```

```
## class: SingleCellExperiment
## dim: 100 10
## metadata(0):
## assays(1): counts
## rownames(100): ENSG00001 ENSG00002 ... ENSG000099 ENSG0000100
## rowData names(2): ID Symbol
## colnames: NULL
## colData names(2): Sample Barcode
## reducedDimNames(0):
## spikeNames(0):
```

The counts themselves are loaded as a sparse matrix, specifically a

`dgCMatrix` from the *Matrix (https://CRAN.R-project.org/package=Matrix)* package. This reduces memory usage by only storing the non-zero counts, which is useful for sparse scRNA-seq data with lots of dropouts.

```
class(counts(sce))
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

Users can also load multiple samples at once by supplying a character vector to `read10xCounts`. This will return a single `SingleCellExperiment` where all of the individual matrices are combined by column. Obviously, this only makes sense when the same set of genes is being used across samples.

2.2 From the molecule information file

CellRanger will also produce a molecule information file (`molecule_info.h5`) that contains... well, information about the transcript molecules. This includes the UMI sequence[2], the cell barcode sequence, the gene to which it was assigned, and the number of reads covering the molecule. For demonstration purposes, we create an example molecule information file below:

[2] For readers who are unfamiliar with UMIs, they allow reads from different PCR amplicons to be unambiguously assigned to the same original molecule.

```
set.seed(1000)
mol.info.file <- DropletUtils:::sim10xMolInfo(tempfile())
mol.info.file
```

```
## [1] "/tmp/RtmpmpYsS5/file734a628abdf7.1.h5"
```

We can subsequently load this information into our R session using the `read10xMolInfo` function:

```
mol.info <- read10xMolInfo(mol.info.file)
mol.info
```

```
## $data
## DataFrame with 9521 rows and 5 columns
##             cell       umi gem_group      gene     reads
##      <character> <integer> <numeric> <integer> <integer>
## 1           CCAT   1034772         1        18        11
## 2           TAAG    239507         1        10        11
## 3           ACTC    399037         1         8         8
## 4           GTAA     27021         1        14         7
## 5           GACA    985911         1         9        14
## ...          ...       ...       ...       ...       ...
## 9517        CACA    324723         1        13        10
## 9518        CCGA    713203         1         2         8
## 9519        AGCC    522272         1         2         6
## 9520        TAAC    710058         1        20         9
## 9521        GCTC    359348         1         3        10
##
## $genes
##  [1] "ENSG1"  "ENSG2"  "ENSG3"  "ENSG4"  "ENSG5"  "ENSG6"  "E
NSG7"  "ENSG8"
##  [9] "ENSG9"  "ENSG10" "ENSG11" "ENSG12" "ENSG13" "ENSG14" "E
NSG15" "ENSG16"
## [17] "ENSG17" "ENSG18" "ENSG19" "ENSG20"
```

This information can be useful for quality control purposes, especially when the underlying read counts are required, e.g., to investigate sequencing saturation. Note that the function will automatically guess the length of the barcode sequence, as this is not formally defined in the molecule information file. For most experiments, the guess is correct, but users can force the function to use a known barcode length with the `barcode.length` argument.

3 Downsampling across batches

3.1 From the count matrix

Given multiple batches of very different sequencing depths, it can be beneficial to downsample the deepest batches to match the coverage of the shallowest batches. This avoids differences in technical noise that can drive clustering by batch. We can achieve this using the `downsampleMatrix` function on the count matrix:

```
set.seed(100)
new.counts <- downsampleMatrix(counts(sce), prop=0.5)
library(Matrix)
colSums(counts(sce))
```

```
##  [1] 515 549 466 474 524 519 466 450 490 484
```

```
colSums(new.counts)
```

```
##  [1] 258 275 233 237 262 260 233 225 245 242
```

The above code will downsample the counts for each cell such that the total count is halved. Calculation of an appropriate `prop` is the responsibility of the user, depending on the number of batches in their experiment and which ones have the lowest coverage. You can also downsample with cell-specific proportions by supplying a vector to `prop`.

3.2 From the reads

Technically, downsampling on the reads is more appropriate as it recapitulates the effect of differences in sequencing depth per cell. This can be achieved by applying the `downsampleReads` function to the molecule information file containing the read counts:

```
set.seed(100)
no.sampling <- downsampleReads(mol.info.file, prop=1)
sum(no.sampling)
```

```
## [1] 9521
```

```
with.sampling <- downsampleReads(mol.info.file, prop=0.5)
sum(with.sampling)
```

```
## [1] 9491
```

The above code will downsample the **reads** to 50% of the original coverage across the experiment. However, the function will return a matrix of **UMI counts**, so the final total count may not actually decrease if sequencing saturation is high! Users should use `downsampleMatrix` instead if they want to guarantee similar total counts after downsampling.

4 Computing barcode ranks

A useful diagnostic for droplet-based data is the barcode rank plot, which shows the (log-)total UMI count for each barcode on the x-axis and the (log-)rank on the y-axis. This is effectively a transposed empirical cumulative density plot with log-transformed axes. It is useful as it allows users to examine the distribution of total counts across barcodes, focusing on those with the largest counts. To demonstrate, let us mock up a count matrix:

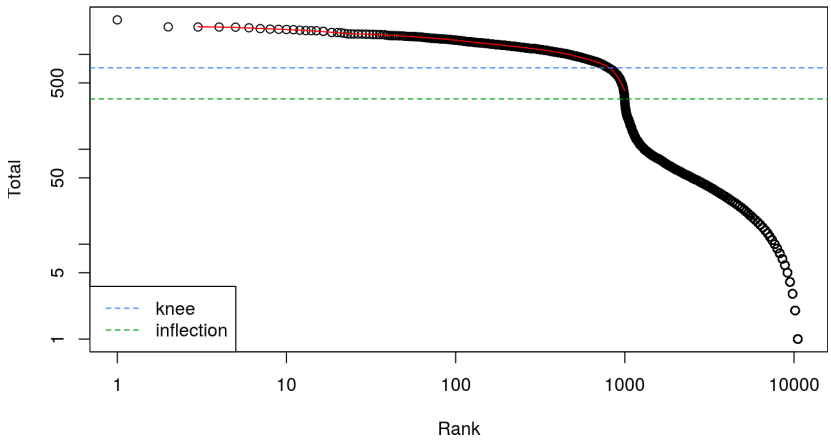
```
set.seed(0)
my.counts <- DropletUtils:::simCounts()
```

We compute the statistics using the `barcodeRanks` function, and then

create the plot as shown below.

```
br.out <- barcodeRanks(my.counts)

# Making a plot.
plot(br.out$rank, br.out$total, log="xy", xlab="Rank", ylab="Tot
al")
o <- order(br.out$rank)
lines(br.out$rank[o], br.out$fitted[o], col="red")
abline(h=br.out$knee, col="dodgerblue", lty=2)
abline(h=br.out$inflection, col="forestgreen", lty=2)
legend("bottomleft", lty=2, col=c("dodgerblue", "forestgreen"),
    legend=c("knee", "inflection"))
```



The knee and inflection points on the curve mark the transition between two components of the total count distribution. This is assumed to represent the difference between empty droplets with little RNA and cell-containing droplets with much more RNA, though a more rigorous method for distinguishing between these two possibilities is discussed below.

5 Detecting empty droplets

Empty droplets often contain RNA from the ambient solution, resulting in non-zero counts after debarcoding. The `emptyDrops` function is designed to distinguish between empty droplets and cells. It does so by testing each barcode's expression profile for significant deviation from the ambient profile. Given a matrix `my.counts` containing UMI counts for **all** barcodes, we call:

```
set.seed(100)
e.out <- emptyDrops(my.counts)
e.out
```

```
## DataFrame with 11100 rows and 5 columns
##           Total           LogProb            PValue   Limited
##       <integer>         <numeric>         <numeric> <logical>
## 1             2                NA                NA        NA
## 2             9                NA                NA        NA
## 3            20                NA                NA        NA
## 4            20                NA                NA        NA
## 5             1                NA                NA        NA
## ...         ...               ...               ...       ...
## 11096       215 -246.428398419154 9.99900009999e-05      TRUE
## 11097       201 -250.233733996276 9.99900009999e-05      TRUE
## 11098       247 -275.904828737003 9.99900009999e-05      TRUE
## 11099       191 -228.762585591832 9.99900009999e-05      TRUE
## 11100       198 -233.042851615118 9.99900009999e-05      TRUE
##                       FDR
##                 <numeric>
## 1                      NA
## 2                      NA
## 3                      NA
## 4                      NA
## 5                      NA
## ...                   ...
## 11096 0.00014426970177566
## 11097 0.00014426970177566
## 11098 0.00014426970177566
## 11099 0.00014426970177566
## 11100 0.00014426970177566
```

Droplets with significant deviations from the ambient profile are detected at a specified FDR threshold, e.g., with `FDR` below 1%. These can be considered to be cell-containing droplets, with a frequency of false positives (i.e., empty droplets) at the specified FDR. Furthermore, droplets with very large counts are automatically retained by setting their *p*-values to zero. This avoids discarding droplets containing cells that are very similar to the ambient profile.

```
is.cell <- e.out$FDR <= 0.01
sum(is.cell, na.rm=TRUE)
```

```
## [1] 902
```

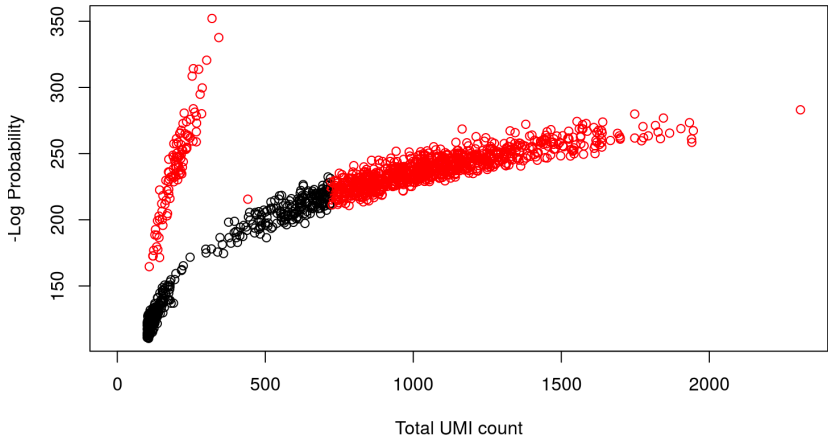
The *p*-values are calculated by permutation testing, hence the need to set a seed. The `Limited` field indicates whether a lower p-value could be obtained by increasing the number of permutations. If there are any entries with `FDR` above the desired threshold and `Limited==TRUE`, it indicates that `npts` should be increased in the `emptyDrops` call.

```
table(Limited=e.out$Limited, Significant=is.cell)
```

```
##        Significant
## Limited FALSE TRUE
##   FALSE   398  802
##   TRUE      0  100
```

We recommend making some diagnostic plots such as the total count against the negative log-probability. Droplets detected as cells should show up with large negative log-probabilities *or* very large total counts (based on the knee point reported by `barcodeRanks`). Note that the example below is based on simulated data and is quite exaggerated.

```
plot(e.out$Total, -e.out$LogProb, col=ifelse(is.cell, "red", "bl
ack"),
    xlab="Total UMI count", ylab="-Log Probability")
```



6 Removing the effect of barcode swapping

Barcode swapping is a phenomenon that occurs upon multiplexing samples on the Illumina 4000 sequencer. Molecules from one sample are incorrectly labelled with *sample* barcodes from another sample, resulting in their misassignment upon demultiplexing. Fortunately, droplet experiments provide a unique opportunity to eliminate this effect, by assuming that it is effectively impossible to generate multiple molecules with the same combination of cell barcode, assigned gene and UMI sequence. Thus, any molecules with the same combination across multiple samples are likely to arise from barcode swapping.

The `swappedDrops` function will identify overlapping combinations in the molecule information files of all multiplexed 10X samples sequenced on the same run. It will remove these combinations and return “cleaned” UMI count matrices for all samples to use in downstream analyses. To demonstrate, we mock up a set of molecule information files for three multiplexed 10X samples:

```
set.seed(1000)
mult.mol.info <- DropletUtils:::sim10xMolInfo(tempfile(), nsampl
es=3)
mult.mol.info
```

```
## [1] "/tmp/RtmpmpYsS5/file734a725d8eb8.1.h5"
## [2] "/tmp/RtmpmpYsS5/file734a725d8eb8.2.h5"
## [3] "/tmp/RtmpmpYsS5/file734a725d8eb8.3.h5"
```

We then apply `swappedDrops` to these files to remove the effect of swapping in our count matrices.

```
s.out <- swappedDrops(mult.mol.info, min.frac=0.9)
length(s.out$cleaned)
```

```
## [1] 3
```

```
class(s.out$cleaned[[1]])
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

For combinations where 90% of the reads belong to a single sample, the molecule is assigned to that sample rather than being removed. This assumes that swapping is relatively rare, so that the read count should be highest in the sample of origin. The exact percentage can be tuned by altering `min.frac` in the `swappedDrops` call.

7 Session information

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils
datasets
## [8] methods   base
##
## other attached packages:
##  [1] Matrix_1.2-15               DropletUtils_1.2.2
##  [3] SingleCellExperiment_1.4.1  SummarizedExperiment_1.12.0
##  [5] DelayedArray_0.8.0          matrixStats_0.54.0
##  [7] Biobase_2.42.0              GenomicRanges_1.34.0
##  [9] GenomeInfoDb_1.18.1         IRanges_2.16.0
## [11] S4Vectors_0.20.1            BiocGenerics_0.28.0
## [13] BiocParallel_1.16.5         knitr_1.21
## [15] BiocStyle_2.10.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.0             compiler_3.5.2         BiocManage
r_1.30.4
##  [4] XVector_0.22.0         bitops_1.0-6           tools_3.5.
2
##  [7] zlibbioc_1.28.0        digest_0.6.18          evaluate_
0.12
## [10] rhdf5_2.26.2           lattice_0.20-38        yaml_2.2.0
## [13] xfun_0.4               GenomeInfoDbData_1.2.0 stringr_1.
3.1
## [16] locfit_1.5-9.1         grid_3.5.2             HDF5Array_
1.10.1
## [19] rmarkdown_1.11         bookdown_0.9           limma_3.3
8.3
## [22] edgeR_3.24.3           Rhdf5lib_1.4.2         magrittr_
1.5
## [25] htmltools_0.3.6        stringi_1.2.4          RCurl_1.95
-4.11
```