alevinQC

Charlotte Soneson

2024-04-30

Abstract

alevinQC (https://bioconductor.org/packages/3.19/alevinQC) reads output files from alevin and generates summary reports.

Package

alevinQC 1.20.0

Contents

1 Introduction

2 Installation

3 Assumed output directory structure

4 Check that all required alevin files are available

5 Generate QC report

6 Create shiny app

7 Generate individual plots

8 Session info

References

1 Introduction

The purpose of the *alevinQC (https://bioconductor.org/packages/3.19/alevinQC)* package is to generate a summary QC report based on the output of an alevin (https://salmon.readthedocs.io/en/latest/alevin.html) (Srivastava et al. 2019) run. The QC report can be generated as a html or pdf file, or launched as a shiny application.

2 Installation

`alevinQC` can be installed using the `BiocManager` CRAN package.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("alevinQC")
```

After installation, load the package into the R session.

```
library(alevinQC)
```

Note that in order to process output from Salmon v0.14 or later, you need Alevin v1.1 or later.

3 Assumed output directory structure

For more information about running alevin, we refer to the documentation (https://salmon.readthedocs.io/en/latest/alevin.html). When invoked, alevin generates several output files in the specified output directory. *alevinQC (https://bioconductor.org/packages/3.19/alevinQC)* assumes that this structure is retained, and will return an error if it isn't - thus, it is not recommended to move or rename the output files from alevin. *alevinQC (https://bioconductor.org/packages/3.19/alevinQC)* assumes that the following files (in the indicated structure) are available in the provided `baseDir` (note that currently, in order to generate the full set of files, alevin must be invoked with the `--dumpFeatures` flag).

For alevin versions before 0.14:

```
baseDir
  |--alevin
  |    |--featureDump.txt
  |    |--filtered_cb_frequency.txt
  |    |--MappedUmi.txt
  |    |--quants_mat_cols.txt
  |    |--quants_mat_rows.txt
  |    |--quants_mat.gz
  |    |--raw_cb_frequency.txt
  |    |--whitelist.txt
  |--aux_info
  |    |--meta_info.json
  |--cmd_info.json
```

For alevin version 0.14 and later:

```
baseDir
  |--alevin
  |    |--featureDump.txt
  |    |--raw_cb_frequency.txt
  |    |--whitelist.txt  (depending on how alevin was run)
  |--aux_info
  |    |--meta_info.json
  |    |--alevin_meta_info.json
  |--cmd_info.json
```

4 Check that all required alevin files are available

The report generation functions (see below) will check that all the required files are available in the provided base directory. However, you can also call the function `checkAlevinInputFiles()` to run the check manually. If one or more files are missing, the function will raise an error indicating the missing file(s).

```
baseDir <- system.file("extdata/alevin_example_v0.14", package = '
checkAlevinInputFiles(baseDir = baseDir)
#> [1] "v0.14"
```

5 Generate QC report

The `alevinQCReport()` function generates the QC report from the alevin output. Depending on the file extension of the `outputFile` argument, and the value of `outputFormat`, the function can generate either an html report or a pdf report.

```
outputDir <- tempdir()
alevinQCReport(baseDir = baseDir, sampleId = "testSample",
               outputFile = "alevinReport.html",
               outputFormat = "html_document",
               outputDir = outputDir, forceOverwrite = TRUE)
```

6 Create shiny app

In addition to static reports, *alevinQC (https://bioconductor.org/packages/3.19/alevinQC)* can also generate a shiny application, containing the same summary figures as the pdf and html reports.

```
app <- alevinQCShiny(baseDir = baseDir, sampleId = "testSample")
```

Once created, the app can be launched using the `runApp()` function from the *shiny (https://CRAN.R-project.org/package=shiny)* package.

```
shiny::runApp(app)
```

It is possible to export the data used internally by the interactive application (in effect, the output from the internal call to `readAlevinQC()` or `readAlevinFryQC()`). To enable such export, first generate the `app` object as in the example above, and then assign the call to `shiny::runApp()` to a variable to capture the output. For example:

```
if (interactive()) {
    out <- shiny::runApp(app)
}
```

To activate the export, make sure to click the button ‘Close app’ in the top right corner in order to close the application (don’t just close the window). This will take you back to your R session, where the variable `out` will be populated with the data used in the app.

7 Generate individual plots

The individual plots included in the QC reports can also be independently generated. To do so, we must first read the alevin output into an R object.

```
alevin <- readAlevinQC(baseDir = baseDir)
```

The resulting list contains four entries:

- `cbTable` : a `data.frame` with various inferred characteristics of the individual cell barcodes.
- `summaryTables` : a list of `data.frame` s with summary information about the full data set, the initial set of whitelisted cells and the final set of whitelisted cells, respectively.
- `versionTable` : a `matrix` with information about the invokation of alevin.
- `type` : a `character` scalar indicating how alevinQC interpreted the alevin output directory.

```
head(alevin$cbTable)
#>                 CB originalFreq ranking collapsedFreq nbrMappec
#> 1 GACTGCGAGGGCATGT       121577       1        123419       104
#> 2 GGTGCGTAGGCTACGA       110467       2        111987        93
#> 3 ATGAGGGAGTAGTGCG       106446       3        108173        88
#> 4 ACTGTCCTCATGCTCC       104794       4        106085        81
#> 5 CGAACATTCTGATACG       104616       5        106072        84
#> 6 ACTGTCCCATATGGTC        99208       6        100776        81
#>   totalUMICount mappingRate dedupRate  MeanByMax nbrGenesAboveZ
#> 1         73312    0.843695  0.295943 0.00735194              7
#> 2         66002    0.835883  0.294911 0.00783094              7
#> 3         62196    0.817958  0.297069 0.00832595              7
#> 4         57082    0.771824  0.302849 0.00619664              6
#> 5         58547    0.795639  0.306274 0.00743685              7
#> 6         56534    0.804418  0.302618 0.00947029              6
#>   nbrGenesAboveMean ArborescenceCount inFinalWhiteList inFirstW
#> 1              1237           1.42034             TRUE
#> 2              1238           1.41826             TRUE
#> 3              1151           1.42262             TRUE
#> 4               957           1.43441             TRUE
#> 5              1238           1.44149             TRUE
#> 6              1068           1.43393             TRUE
```

```
knitr::kable(alevin$summaryTables$fullDataset)
```

Total number of processed reads	7197662
Number of reads with Ns	35362
Number of reads with valid cell barcode (no Ns)	7162300
Number of mapped reads	4869156
Percent mapped (of all reads)	67.65%
Number of noisy CB reads	1003624
Number of noisy UMI reads	266
Total number of observed cell barcodes	188613

```
knitr::kable(alevin$summaryTables$initialWhitelist)
```

Number of barcodes (initial whitelist)	100
Number of barcodes with quantification (initial whitelist)	100
Fraction reads in barcodes (initial whitelist)	84.64%
Mean number of reads per cell (initial whitelist)	60620
Median number of reads per cell (initial whitelist)	58132
Mean number of detected genes per cell (initial whitelist)	5163
Median number of detected genes per cell (initial whitelist)	5268
Mean UMI count per cell (initial whitelist)	33274
Median UMI count per cell (initial whitelist)	31353

```
knitr::kable(alevin$summaryTables$finalWhitelist)
```

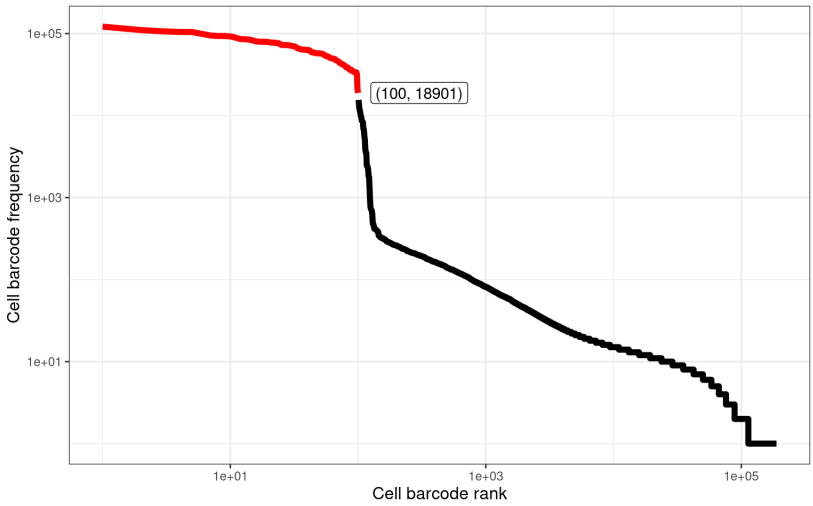
Number of barcodes (final whitelist)	95
Number of barcodes with quantification (final whitelist)	95
Fraction reads in barcodes (final whitelist)	82.39%
Mean number of reads per cell (final whitelist)	62118
Median number of reads per cell (final whitelist)	58725
Mean number of detected genes per cell (final whitelist)	5260
Median number of detected genes per cell (final whitelist)	5343
Mean UMI count per cell (final whitelist)	34091
Median UMI count per cell (final whitelist)	32028

```
knitr::kable(alevin$versionTable)
```

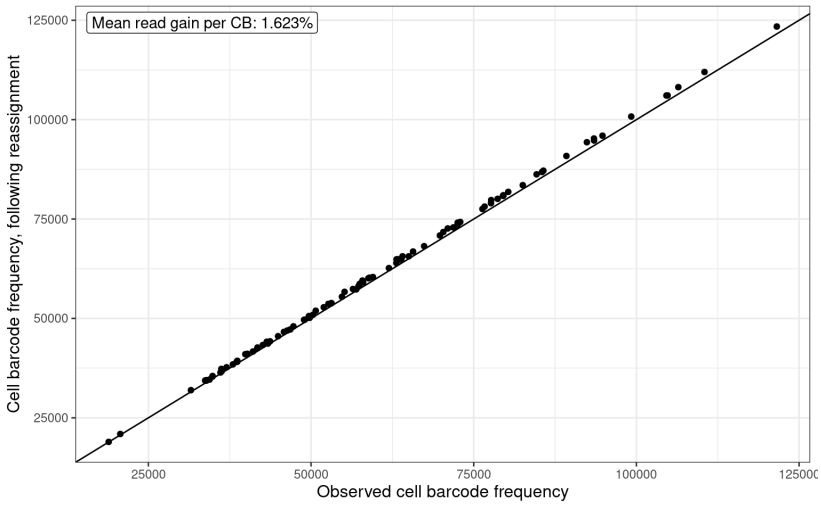
Start time	Thu May 30 13:06:55 2019
Salmon version	0.14.0
Index	/mnt/scratch5/avi/alevin/data/mohu/salmon_index
R1file	/mnt/scratch5/avi/alevin/data/10x/v2/mohu/100/all_bcs.fq
R2file	/mnt/scratch5/avi/alevin/data/10x/v2/mohu/100/all_reads.fq
tgMap	/mnt/scratch5/avi/alevin/data/mohu/gtf/txp2gene.tsv
Library type	ISR

The plots can now be generated using the dedicated plotting functions provided with *alevinQC (https://bioconductor.org/packages/3.19/alevinQC)* (see the help file for the respective function for more information).

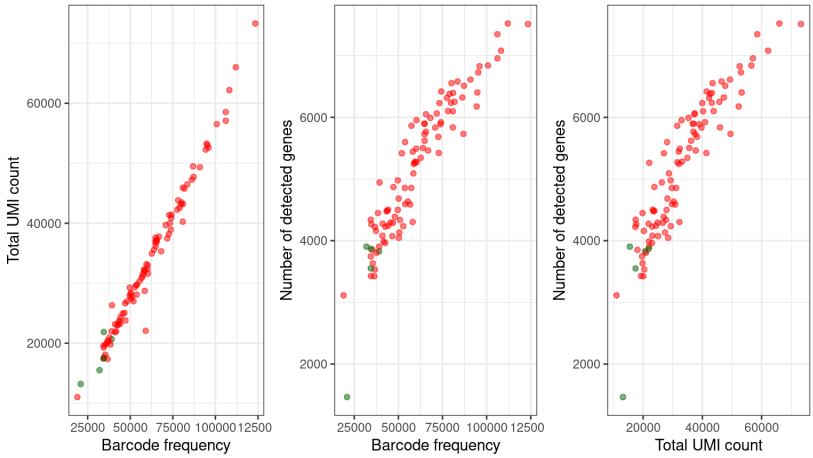
```
plotAlevinKneeRaw(alevin$cbTable)
```



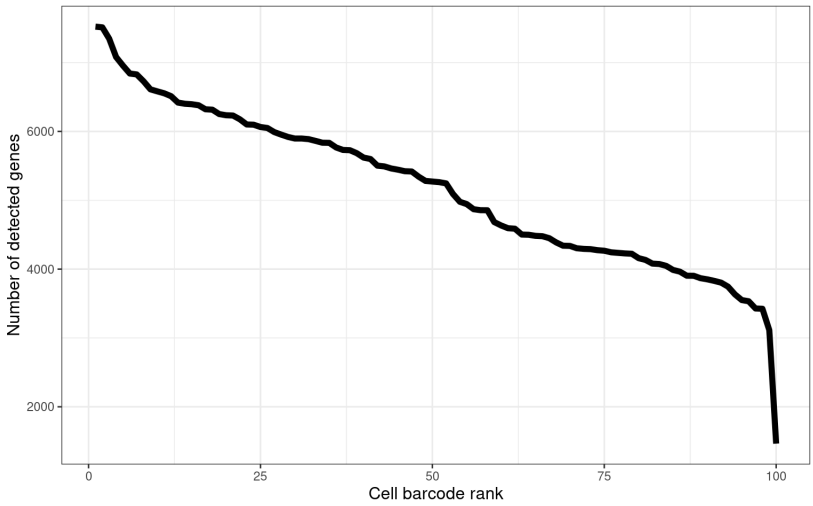

```
plotAlevinBarcodeCollapse(alevin$cbTable)
```




```
plotAlevinQuant(alevin$cbTable)
```




```
plotAlevinKneeNbrGenes(alevin$cbTable)
```



8 Session info

```
sessionInfo()
#> R version 4.4.0 beta (2024-04-15 r86425)
#> Platform: x86_64-pc-linux-gnu
#> Running under: Ubuntu 22.04.4 LTS
#>
#> Matrix products: default
#> BLAS:   /home/biocbuild/bbs-3.19-bioc/R/lib/libRblas.so
#> LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
#>  [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
#>
#> time zone: America/New_York
#> tzcode source: system (glibc)
#>
#> attached base packages:
#> [1] stats     graphics  grDevices utils     datasets  methods
#>
#> other attached packages:
#> [1] alevinQC_1.20.0  BiocStyle_2.32.0
#>
#> loaded via a namespace (and not attached):
#>  [1] tximport_1.32.0      sass_0.4.9           utf8_1.2.4
#>  [4] generics_0.1.3       tidyr_1.3.1          digest_0.6.35
#>  [7] magrittr_2.0.3       evaluate_0.23        grid_4.4.0
#> [10] RColorBrewer_1.1-3   bookdown_0.39        fastmap_1.1.1
#> [13] plyr_1.8.9           jsonlite_1.8.8       promises_1.3.0
#> [16] BiocManager_1.30.22  GGally_2.2.1         purrr_1.0.2
#> [19] fansi_1.0.6          crosstalk_1.2.1      scales_1.3.0
#> [22] shinydashboard_0.7.2 jquerylib_0.1.4      cli_3.6.2
#> [25] shiny_1.8.1.1        rlang_1.1.3          cowplot_1.1.3
#> [28] munsell_0.5.1        withr_3.0.0          cachem_1.0.8
#> [31] yaml_2.3.8           tools_4.4.0          dplyr_1.1.4
#> [34] colorspace_2.1-0     ggplot2_3.5.1        httpuv_1.6.15
#> [37] DT_0.33              ggstats_0.6.0        vctrs_0.6.5
#> [40] R6_2.5.1             mime_0.12            lifecycle_1.0.4
#> [43] htmlwidgets_1.6.4    fontawesome_0.5.2    pkgconfig_2.0.3
#> [46] pillar_1.9.0         bslib_0.7.0          later_1.3.2
#> [49] gtable_0.3.5         glue_1.7.0           Rcpp_1.0.12
#> [52] highr_0.10           xfun_0.43            tibble_3.2.1
#> [55] tidyselect_1.2.1     knitr_1.46           farver_2.1.1
#> [58] xtable_1.8-4         rjson_0.2.21         htmltools_0.5.8.
#> [61] labeling_0.4.3       rmarkdown_2.26       compiler_4.4.0
```

References

Srivastava, Avi, Laraib Malik, Tom Sean Smith, Ian Sudbery, and Rob Patro. 2019. “Alevin Efficiently Estimates Accurate Gene Abundances from dscRNA-seq Data.” *Genome Biology* 20: 65.