CellBench: RNA mixture

Kim-Anh Lê Cao, Luyi Tian

September 11, 2018

Contents

1 Important notes

- fix description to match the pure mix vignette
- clean up the intro about the number of cells

2 Packages

Install the relevant bioconductor packages to extract the data

Load the relevant libraries

3 Data

3.1 Experimental design

Cellbench is a project conducted at Walter and Eliza Hall Institute of Medical Research (WEHI) and University of Melbourne (Mr Luyi Tian, Dr Shalin Naik and Dr Matt Ritchie from WEHI, and Dr Kim-Anh Lê Cao from UoM) to generate gold standard single-cell RNA sequencing (scRNA-seq) datasets by mixing three human lung adenocarcinoma cell lines (**H2228, H1975 and HCC827**) and using different cell isolation and library preparation methods (CEL-seq2 and SORT-seq).

The Cellbench experiment has been carefully designed to have controlled variations in mRNA amount (ranging from 3.75pg to 30pg) to simulate different cell sizes. In addition, the cluster identity of each 'pseudo cell' determined by the mixture that was sampled from is known in advance. Such data sets will help computational biologists to propose and develop novel and useful tools for the analysis of single-cell gene expression data.

CellBench enables us to benchmark different types of statistical scRNA-seq analyses, ranging from normalization to clustering and trajectory analysis. Here we will particularly focus on exploring two data sets generated from two different isolation protocols, and the classification of different groups of mixtures of cell types.

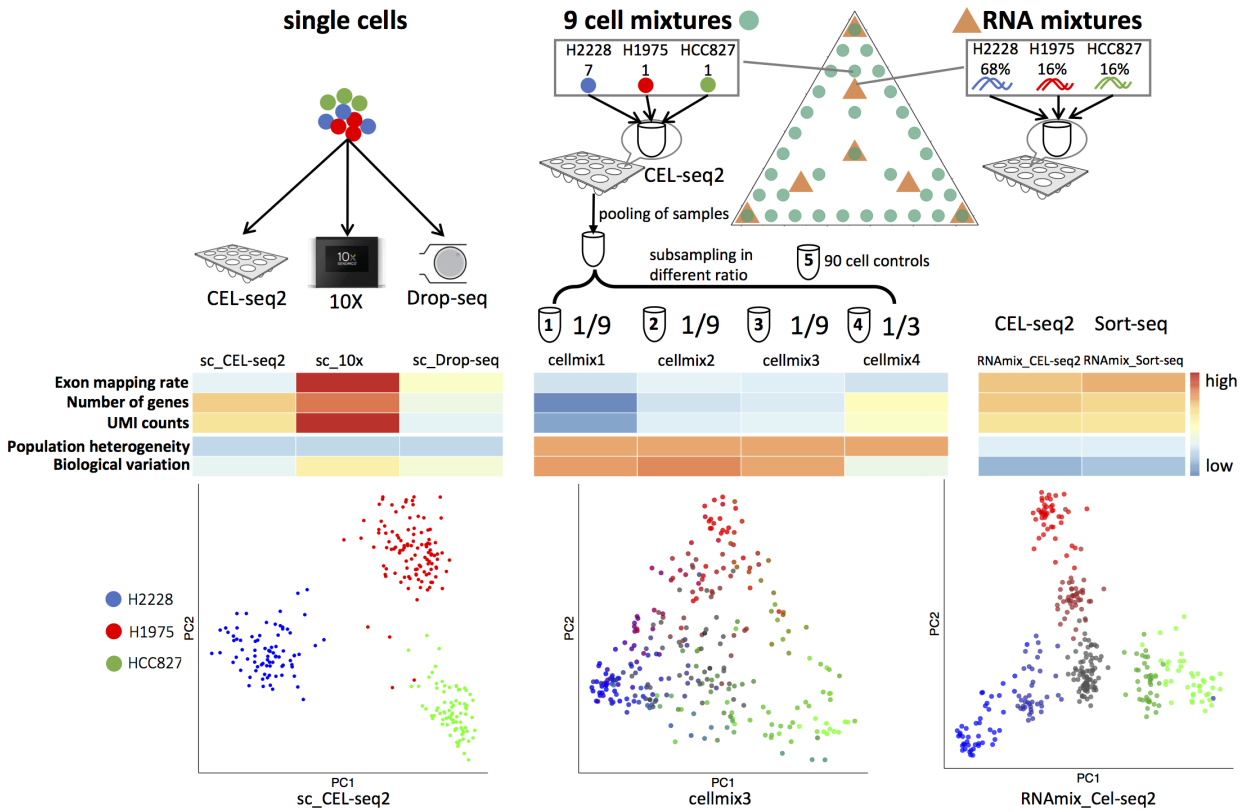


Figure 1: CellBench Experimental Design. For ths vignette we focus on the RNA mixtures (right hand side of this diagram). For this experiment, samples from the 7 pools were sub-sampled at varying levels to create 'pseudo single cells' of different sizes. Gene expression for these samples was then profiled using two different single cell protocols (CEL-seq2 and SORT-seq) and the resulting libraries were sequenced on an Illumina NextSeq 500.

The data were downloaded from Luyi Tian from his Github link https://github.com/LuyiTian/CellBench_data. The data were processed by Luyi Tian using scPipe (https://bioconductor.org/packages/devel/bioc/vignettes/scPipe/inst/doc/scPipe_tutorial.html).

3.2 Load data

We first load and look at the dimension of the data. Number of transcripts are in rows and number of cells are in columns. The objects **sce2_qc** and **sce8_qc** actually contain some detailed information about the characteristics of the pseudo cells (the *meta data*). Type **colData(sce2_qc)** to look at the information available to you (you can then access it specifically by typing **sce2_qc$** then 'tab' to choose the column of interest).

Check the dimension of the data

```
## [1] 14804   340
```

```
## [1] 15571   296
```

Table 1: Number of cells per RNA mixture type and per protocol

	1	2	3	4	5	6	7	8
CELseq2	39	43	45	45	44	40	45	39
DROPseq	32	42	37	20	41	40	45	39

3.3 Information on each cell and cell types mixtures

We then have a look at the proportion of each cell type in each pseudo cell:

This is also summarised into the 'mix' output, which we save:

```
##  1  2  3  4  5  6  7  8
## 39 43 45 45 44 40 45 39
```

```
##  1  2  3  4  5  6  7  8
## 32 42 37 20 41 40 45 39
```

Break down of the number of pseudo cells per RNA mixture:

3.4 Normalise data

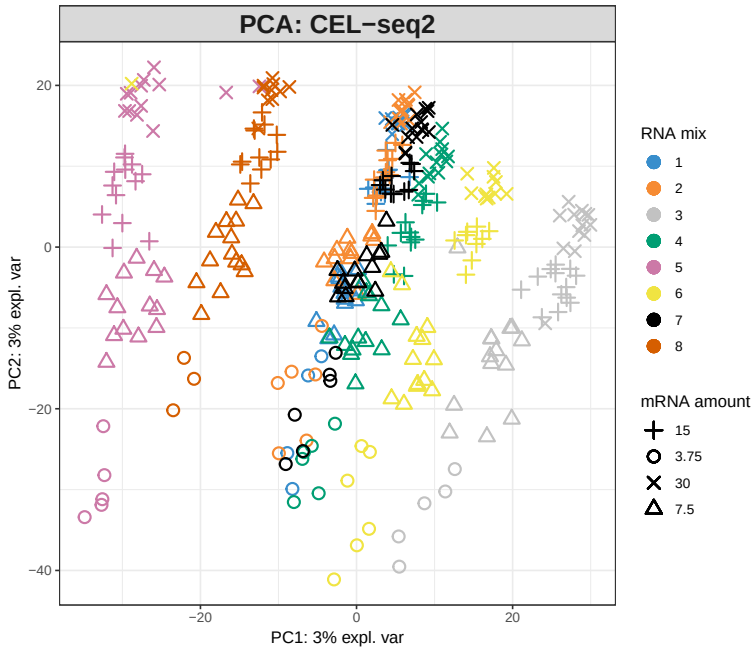
There might be a warning about the ERCC spike-in in the following code.

4 PCA

4.1 Each protocol independently

We first run a PCA on each data set individually on the log normalised counts.

Colors indicate the RNA mixture type. PCA is unsupervised but we can assign to each cell a group and color. On component 1, the patterns we observe are due to the specific proportion of cell types. On component 2, the patterns we see is due to the different mRNA amount. On component 3 (which you can plot by changing the arguments) we see the different proportions of original cell types. We see similar patterns in each data set.


PCA: CEL−seq2
PC2: 3% expl. var
PC1: 3% expl. var
20
0
−20
−40
−20
0
20
RNA mix
1
2
3
4
5
6
7
8
mRNA amount
15
3.75
30
7.5

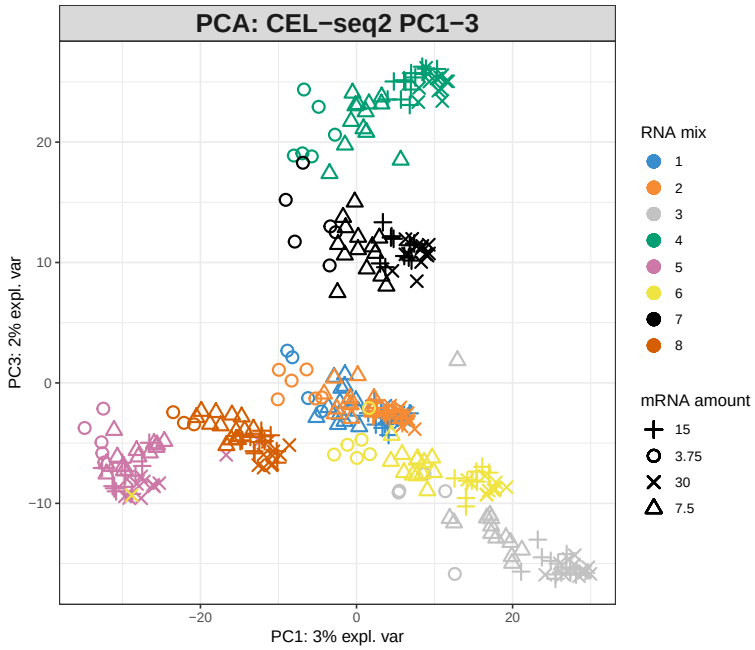
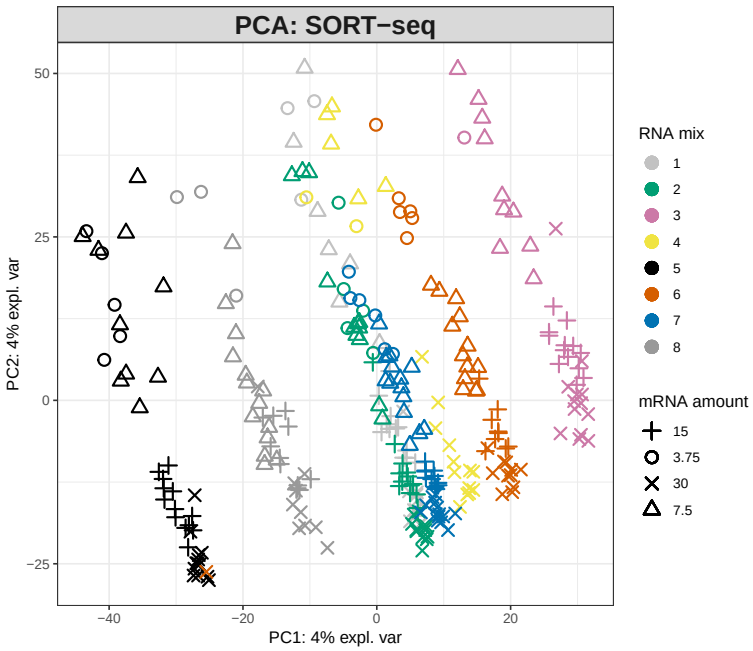

PCA: CEL−seq2 PC1−3
PC3: 2% expl. var
PC1: 3% expl. var
20
10
0
−10
−20
0
20
RNA mix
1
2
3
4
5
6
7
8
mRNA amount
15
3.75
30
7.5

Table 2: Number of cells per protocol

	x
CEL-seq2	340
SORT-seq	296



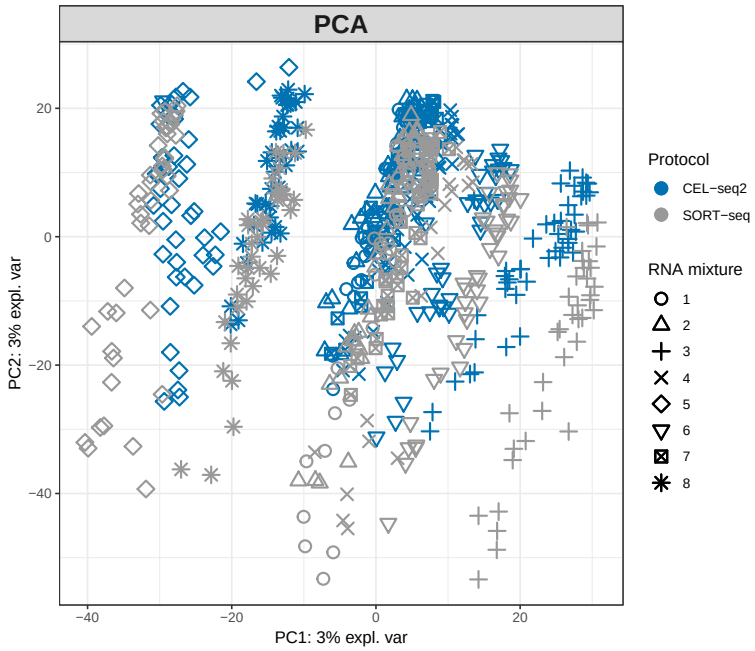
4.2 Data naively combined

We combine the data naively to identify the major sources of variation and visually assess if there is a strong protocol batch effect. See later section for a quantitative assessment.

We first need to extract the common UMI across platform.

We extract the RNA mixture type information and the batch (protocol) information:

PCA on the combined data, using mixOmics. Here the color indicates the protocol:



5 Data integration methods on most variable genes

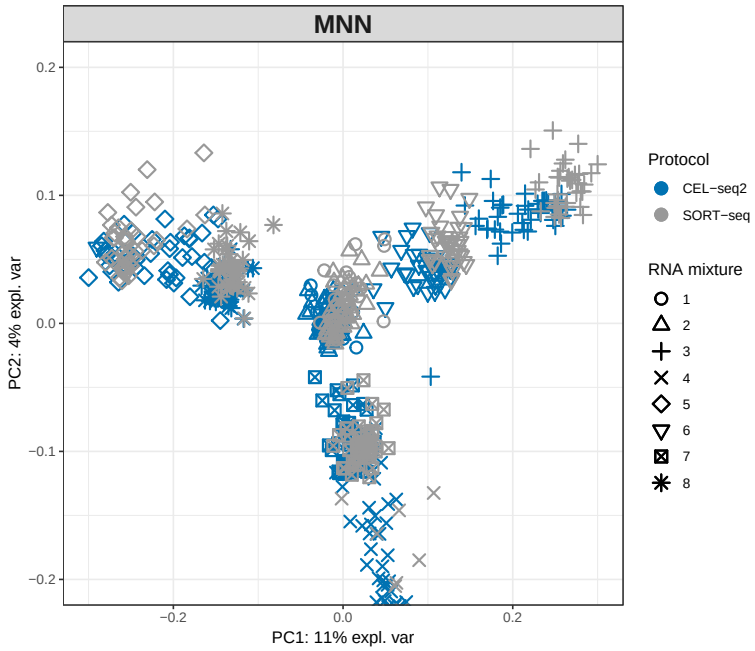
5.0.1 Custom function using decomposeVar (scran)

The decomposeVar function calculates the gene-specific biological and technical variance for an sce object, we then order the top variable genes with a high variance.

Here we obtain 1136 variable genes. We carry on with this list. Note that is it still rather small.

6 MNN correct on HVG

Starting from the highly variable genes identified above



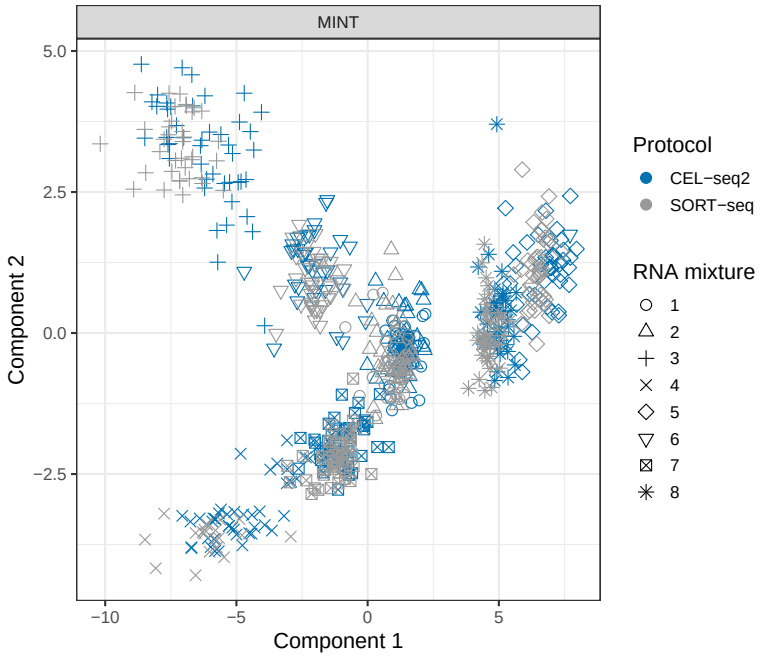
7 sparse MINT with variable selection

With MINT there is not need to select the most variable genes beforehand. We start from the all combined data. The method will select internally the best genes that are agnostic of protocol effect and best discriminate the RNA mixture types.

7.1 Parameters

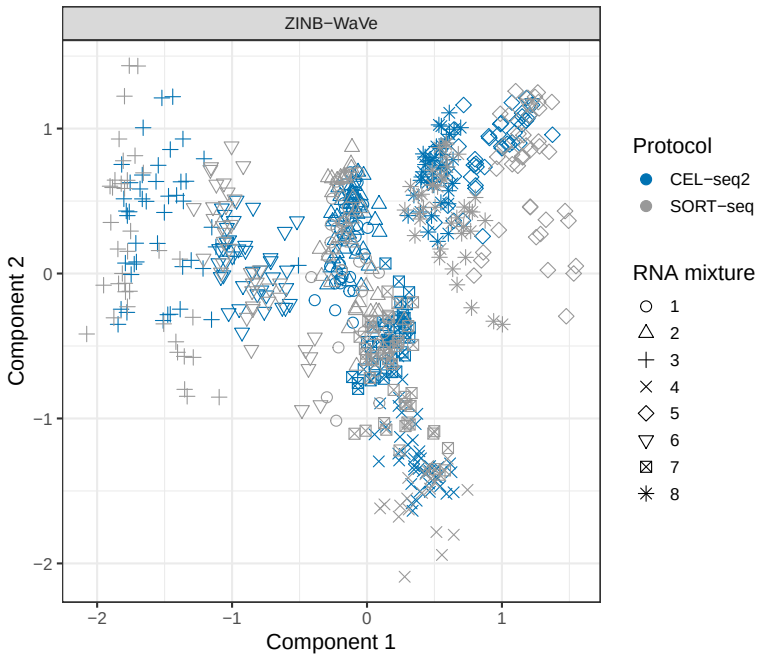
We need to specify the number of variables to select per MINT component. Here we make an arbitrary choice, but we could use the tuning function for an optimal choice, see ??tune.mint.splsda in mixOmics. We provide a full analysis where we explain the tuning step in : https://github.com/AJABADI/MINT_sPLSDA (MINT_Data_Integration). See also another example on microarray data here: http://mixomics.org/mixmint/stemcells-example/.

7.2 MINT variable selection and analysis



8 ZINB-WaVe

Starting from the highly variable genes identified above. ZINB-WaVe gives us a low-dimensional representation of the data. We can extract the weights from the data to carry on with a differential expression analysis (Van den Berge et al. 2018 and example in https://www.bioconductor.org/packages/devel/bioc/vignettes/zinbwave/inst/doc/intro.html). 'The zinbwave package can be used to compute observational weights to "unlock" bulk RNA-seq tools for single-cell applications, as illustrated in (Van den Berge et al. 2018). Since version 1.1.5, zinbwave computes the observational weights by default. See the man page of zinbwave. The weights are stored in an assay named weights and can be accessed with the following call' (from the ZINB-WaVe vignette).

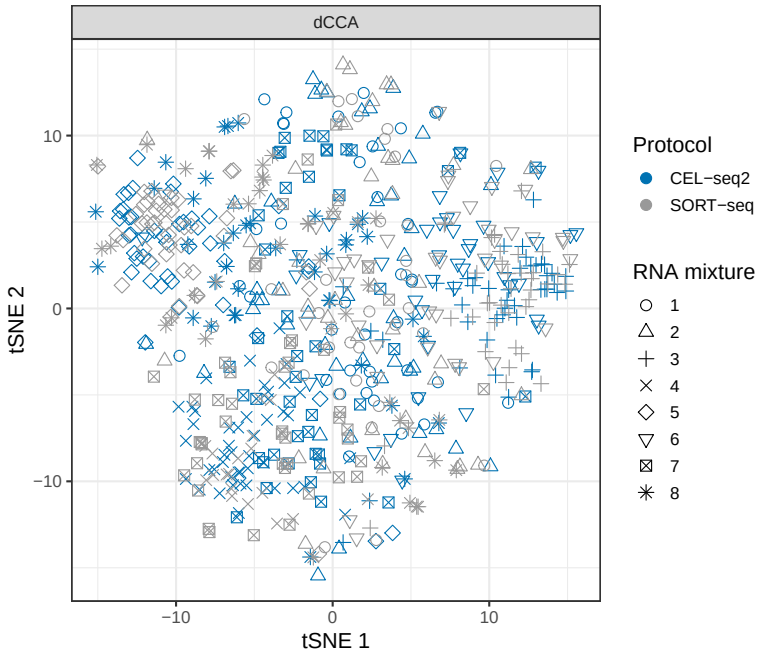


9 Seurat

We run diagonal CCA from Seurat with 15 components. Visualisation of the reduced dimension is through t-SNE.

We first need to normalise the data Seurat-style:

Note that according to scanorama, the order in which the datasets are provided in RunCCA matter. We have not seen different results when setting the reference data set to one study or the other.



10 scMerge

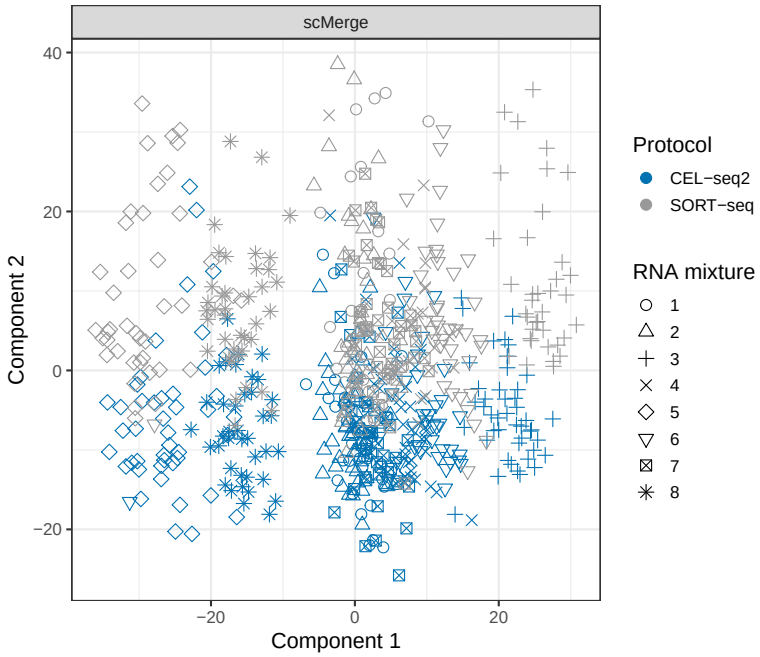
10.1 Identify Stably Expressed Genes

scMerge takes logcounts as input, but we also need to provide the counts to estimate some of the parameters. We identify the SEGs by choosing the most 2000 lowly variable genes per platform and then take the intersection.

We end up with 395 stably expressed genes. We test both unsupervised and supervised version.

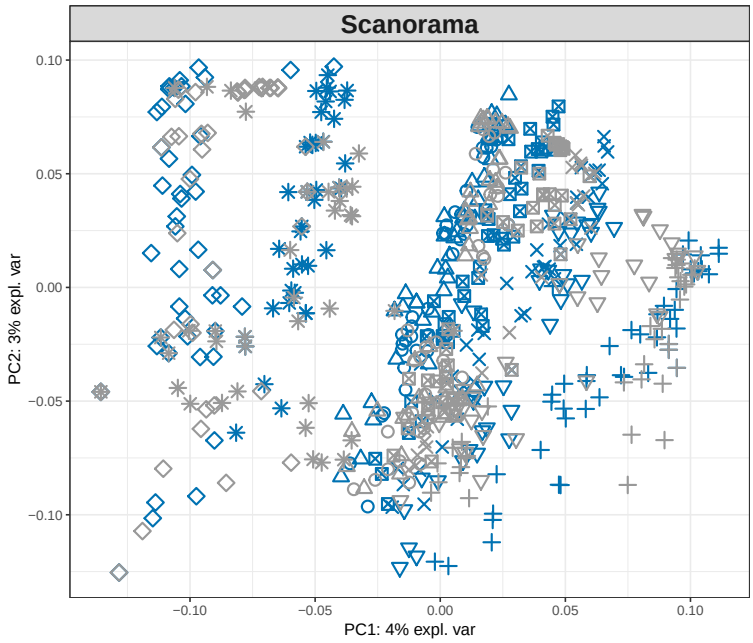
10.2 Unsupervised

We then run an unsupervised scMerge and run PCA on the resulting data matrix.



11 Scanorama

Scanorama is coded in python, we load the results here of the data matrix and run a PCA. By default the scanorama is run on the most 10,000 HVG



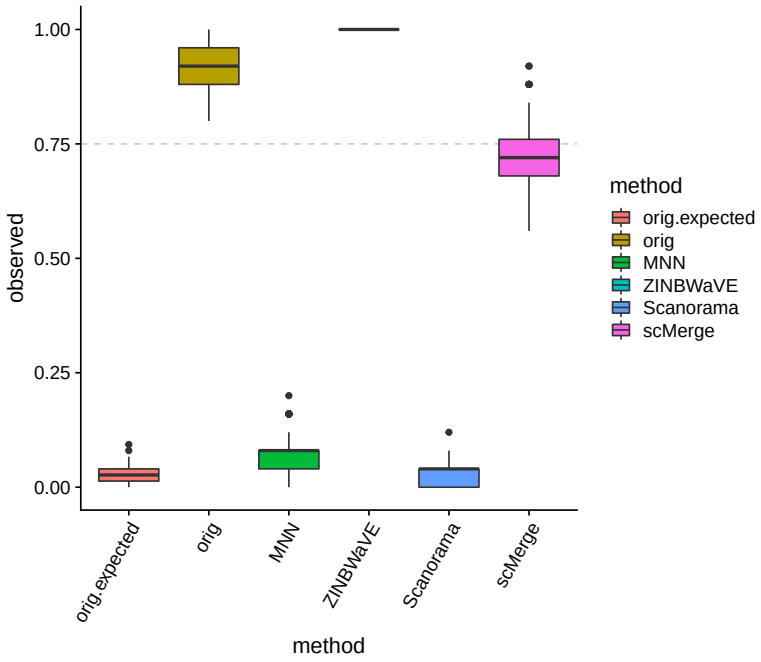
12 Assessment

12.1 kBET evaluation on batch-free matrices

We run kBET on 25% of the sample size, as advised in the help file. kBET is run on the data matrix resulting from the methods run previously, as well as the original data. We abstrain from running it on the methods that output a reduce dimension (Seurat CCA, MINT).

12.1.1 Summary kBet

We highlight as a horizontal line an acceptance rate of 0.75 (see kBET publication https://www.biorxiv.org/content/biorxiv/early/2017/10/27/200345.full.pdf). For each dataset, kBET returns an overall rejection rate. In our case, a high rejecton rate means that cells are surrounded by samples from the same batch.



12.2 Silhouette width for batch and cell line

To assess the clustering of the data, we use silhouette width, an internal validation metric which is an aggregated measure of how similar an observation is to its own cluster compared its closest neighboring cluster. Here our clusters are already defined, based on either the batch information or the cell type information. The metric ranges from -1 to 1, where higher values indicate a strong cluster.

We calculate the silhouette based on the PCs from PCA for each method that yielded either in a data matrix, or a reduced dimension (in the latter case we calculate the silhouette on those components directly). In our case, a high value for each batch indicate a strong batch effect.

Since we use an Euclidean distance we do not run the Silhouette on Seurat CCA t-SNE components.

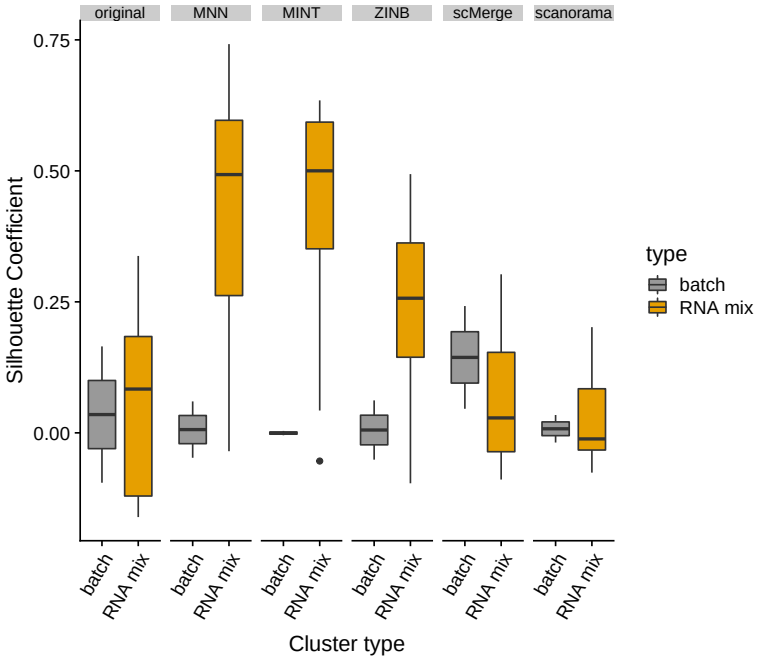
12.2.1 Custom function:

12.2.2 Summary silhouette results

We calculate Silhouette for each method:

Table 3: Summary ARI for components-based methods

	method	ARI
batch	orig	0.51
RNA mix	orig	0.83
batch	MNN	0.5
RNA mix	MNN	0.93
batch	MINT	0.5
RNA mix	MINT	0.94
batch	ZINB	0.5
RNA mix	ZINB	0.9
batch	scMerge	0.5
RNA mix	scMerge	0.82
batch	scanorama	0.5
RNA mix	scanorama	0.82



12.3 ARI

We create a function to calculate the ARI based a distance (euclidean here) and a PAM clustering. The adjusted rand index is then calculate based on the clusters from PAM and the real cluster information (here batch of cell line). A high ARI index with respect to cell line and low with respect to batch ndicates that the method was successful at removing the batch effect whilst retaining the biological information. Here ARI is calculated on component-based methods (from PCA, or directly from ZINB-WaVe or MINT). Seurat CCA was omitted as the reduced representation uses t-SNE.

12.3.1 Custom function

12.3.2 Summary ARI

13 Session information

```
## R version 3.5.0 (2018-04-23)
```

```
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.3
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] clues_0.5.9                 cluster_2.0.7-1
##  [3] kBET_0.99.5                 scMerge_0.1.8
##  [5] scRNAseq_1.6.0              zinbwave_1.2.0
##  [7] Seurat_2.3.4                Matrix_1.2-14
##  [9] cowplot_0.9.2               mixOmics_6.3.2
## [11] lattice_0.20-35             MASS_7.3-50
## [13] scater_1.8.0                ggplot2_3.0.0
## [15] scran_1.8.2                 SingleCellExperiment_1.2.0
## [17] SummarizedExperiment_1.10.1 DelayedArray_0.6.0
## [19] BiocParallel_1.14.1         matrixStats_0.53.1
## [21] Biobase_2.40.0              GenomicRanges_1.32.3
## [23] GenomeInfoDb_1.16.0         IRanges_2.14.10
## [25] S4Vectors_0.18.3            BiocGenerics_0.26.0
## [27] kableExtra_0.9.0            tictoc_1.0
## [29] knitr_1.20
##
## loaded via a namespace (and not attached):
##   [1] shinydashboard_0.7.0     reticulate_1.9
##   [3] R.utils_2.6.0            tidyselect_0.2.4
##   [5] RSQLite_2.1.1            AnnotationDbi_1.42.1
##   [7] htmlwidgets_1.2          grid_3.5.0
##   [9] trimcluster_0.1-2.1      Rtsne_0.13
##  [11] munsell_0.4.3            codetools_0.2-15
##  [13] ica_1.0-2                statmod_1.4.30
##  [15] DT_0.4                   miniUI_0.1.1.1
##  [17] withr_2.1.2              colorspace_1.3-2
##  [19] pspline_1.0-18           rstudioapi_0.7
##  [21] ROCR_1.0-7               robustbase_0.93-0
##  [23] dtw_1.20-1               gbRd_0.4-11
##  [25] labeling_0.3             Rdpack_0.9-0
##  [27] lars_1.2                 tximport_1.8.0
##  [29] GenomeInfoDbData_1.1.0   bit64_0.9-7
##  [31] rhdf5_2.24.0             rprojroot_1.3-2
##  [33] diptest_0.75-7           R6_2.2.2
##  [35] ggbeeswarm_0.6.0         locfit_1.5-9.1
##  [37] hdf5r_1.0.0              manipulateWidget_0.9.0
##  [39] flexmix_2.3-14           bitops_1.0-6
##  [41] assertthat_0.2.0         promises_1.0.1
##  [43] SDMTools_1.1-221         scales_0.5.0
```

```
##  [45] nnet_7.3-12              beeswarm_0.2.3
##  [47] gtable_0.2.0             rlang_0.2.1
##  [49] genefilter_1.62.0        splines_3.5.0
##  [51] lazyeval_0.2.1           acepack_1.4.1
##  [53] checkmate_1.8.5          rgl_0.99.16
##  [55] yaml_2.1.19              reshape2_1.4.3
##  [57] crosstalk_1.0.0          backports_1.1.2
##  [59] httpuv_1.4.3             Hmisc_4.1-1
##  [61] tools_3.5.0              gplots_3.0.1
##  [63] RColorBrewer_1.1-2       proxy_0.4-22
##  [65] stabledist_0.7-1         dynamicTreeCut_1.63-1
##  [67] ggridges_0.5.0           Rcpp_0.12.17
##  [69] plyr_1.8.4               base64enc_0.1-3
##  [71] zlibbioc_1.26.0          purrr_0.2.5
##  [73] RCurl_1.95-4.10          rpart_4.1-13
##  [75] pbapply_1.3-4            viridis_0.5.1
##  [77] zoo_1.8-2                magrittr_1.5
##  [79] data.table_1.11.4        RSpectra_0.13-1
##  [81] lmtest_0.9-36            RANN_2.6
##  [83] mvtnorm_1.0-8            fitdistrplus_1.0-9
##  [85] gsl_1.9-10.3             hms_0.4.2
##  [87] mime_0.5                 evaluate_0.10.1
##  [89] xtable_1.8-2             XML_3.98-1.11
##  [91] mclust_5.4               gridExtra_2.3
##  [93] compiler_3.5.0           ellipse_0.4.1
##  [95] tibble_1.4.2             KernSmooth_2.23-15
##  [97] R.oo_1.22.0              htmltools_0.3.6
##  [99] pcaPP_1.9-73             segmented_0.5-3.0
## [101] corpcor_1.6.9            later_0.7.3
## [103] Formula_1.2-3            snow_0.4-2
## [105] tidyr_0.8.1              DBI_1.0.0
## [107] fpc_2.1-11.1             readr_1.1.1
## [109] R.methodsS3_1.7.1        gdata_2.18.0
## [111] metap_1.0                bindr_0.1.1
## [113] igraph_1.2.2             pkgconfig_2.0.1
## [115] numDeriv_2016.8-1        foreign_0.8-70
## [117] xml2_1.2.0               foreach_1.4.4
## [119] annotate_1.58.0          rARPACK_0.11-0
## [121] vipor_0.4.5              XVector_0.20.0
## [123] bibtex_0.4.2             rvest_0.3.2
## [125] stringr_1.3.1            digest_0.6.15
## [127] copula_0.999-18          tsne_0.1-3
## [129] ADGofTest_0.3            softImpute_1.4
## [131] rmarkdown_1.10           htmlTable_1.12
## [133] edgeR_3.22.2             DelayedMatrixStats_1.2.0
## [135] kernlab_0.9-27           shiny_1.1.0
## [137] gtools_3.5.0             modeltools_0.2-22
## [139] rjson_0.2.20             nlme_3.1-137
## [141] jsonlite_1.5             bindrcpp_0.2.2
## [143] Rhdf5lib_1.2.1           viridisLite_0.3.0
## [145] limma_3.36.1             pillar_1.2.3
## [147] httr_1.3.1               DEoptimR_1.0-8
## [149] survival_2.42-3          glue_1.2.0
## [151] FNN_1.1.2.1              png_0.1-7
```

```
## [153] prabclus_2.2-6           iterators_1.0.9
## [155] glmnet_2.0-16            bit_1.1-14
## [157] mixtools_1.1.0           class_7.3-14
## [159] stringi_1.2.2            blob_1.1.1
## [161] memoise_1.1.0            doSNOW_1.0.16
## [163] latticeExtra_0.6-28      caTools_1.17.1
## [165] dplyr_0.7.6              irlba_2.3.2
## [167] ape_5.1
```