Dune workflow on the baron dataset

Hector Roux de Bézieux

31 October , 2020

Contents

We use the data from (Baron et al. 2016), which is a human pancreas dataset, to display a full scRNA-Seq workflow where **Dune** is run to improve the quality of the clustering. We will see how **Dune** improves the quality of each of the clustering methods used as input.

1 Load data

We rely on a pre-processed dataset where the count matrix has already been computed. We also have information on the batches, i.e the ids of the donors, as well as the cell labels assigned in the original publication. Note that, in that publication, cells were clustered using hierarchical clustering with a final manual merging step. We will show how, running each algorithm with its default, we can polish the results with Dune.

```
set.seed(19)
suppressPackageStartupMessages({
  library(SingleCellExperiment)
  library(stringr)
  library(scRNAseq)
```

```
})
# Load pre-processed dataset
sce <- BaronPancreasData()
```

```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: `filter_()` is deprecated as of dplyr 0.7.0.
## Please use `filter()` instead.
## See vignette('programming') for more help
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## snapshotDate(): 2019-10-22
```

```
## see ?scRNAseq and browseVignettes('scRNAseq') for documentation
```

```
## loading from cache
```

```
## see ?scRNAseq and browseVignettes('scRNAseq') for documentation
```

```
## loading from cache
```

```
# Filter very lowly expressed genes for computational practices.
filt <- rowSums(counts(sce) >= 2) >= 10
sce <- sce[filt, ]
sce
```

```
## class: SingleCellExperiment
## dim: 12336 8569
## metadata(0):
## assays(1): counts
## rownames(12336): A1CF A2M ... ZZZ3 pk
## rowData names(0):
## colnames(8569): human1_lib1.final_cell_0001 human1_lib1.final_cell_0002
##   ... human4_lib3.final_cell_0700 human4_lib3.final_cell_0701
## colData names(2): donor label
## reducedDimNames(0):
## spikeNames(0):
## altExpNames(0):
```

```
rm(filt)
```

2 Pre-processing

To normalize the dataset, we can rely on two methods. We first use the default pipeline from **Seurat** (Cao et al. 2019). Then, we use the **scvi** method from (Lopez et al. 2018).

```
suppressPackageStartupMessages({
  library(Seurat)
})
pre_process_time <- system.time({
  se <- CreateSeuratObject(counts = counts(sce),
                           min.cells = 0,
                           min.features = 0,
```

```
                            project = "de")
  se <- AddMetaData(se, as.data.frame(colData(sce)))
  se <- NormalizeData(se, verbose = FALSE)
  se <- FindVariableFeatures(se, selection.method = 'vst', nfeatures = 4000,
                             verbose = FALSE)
  se <- se[VariableFeatures(se), ]
  se <- ScaleData(object = se, vars.to.regress = c("nCount_RNA", "donor"))
  sce <- as.SingleCellExperiment(se)
})
```

```
## Regressing out nCount_RNA, donor
```

```
## Centering and scaling data matrix
```

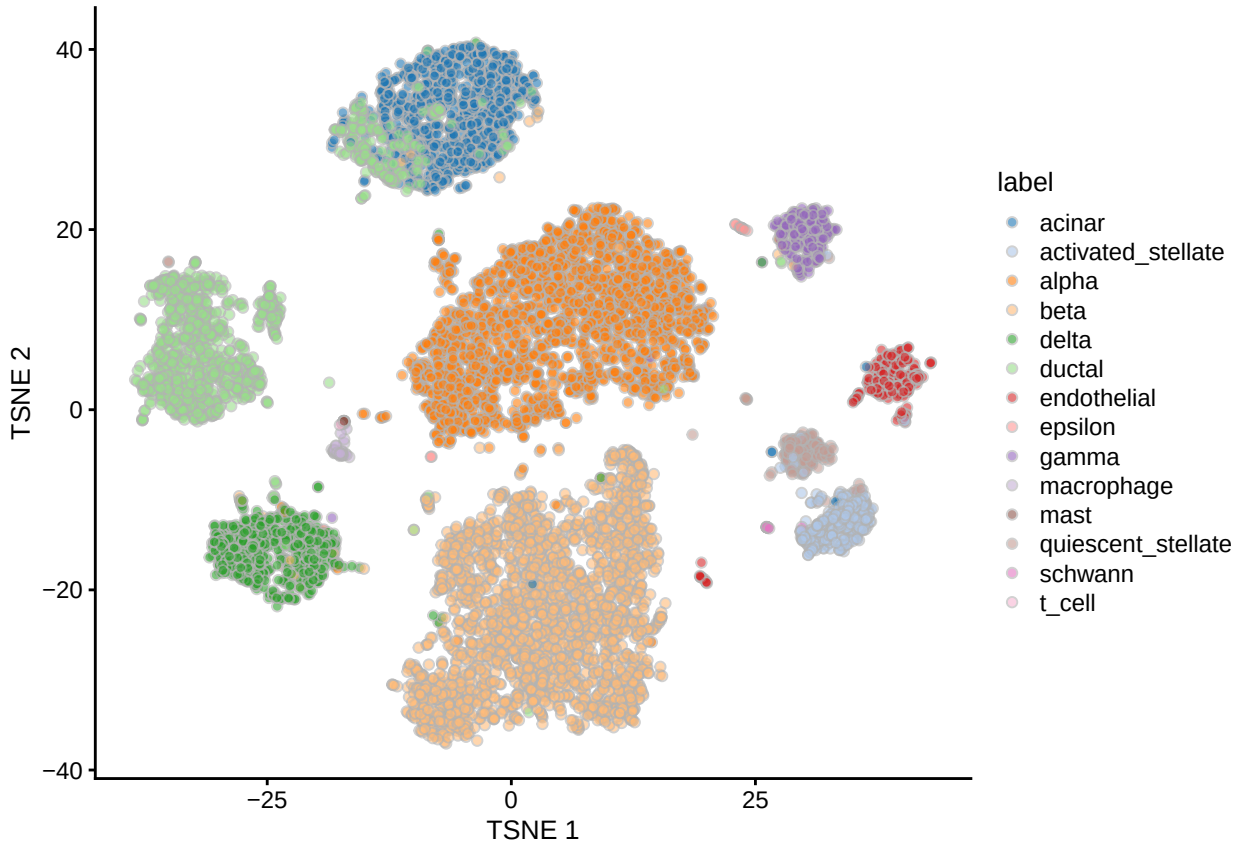
```
# Note that this chunk of code is actually python code run from R. To use the
# reticulate package, please follow https://rstudio.github.io/reticulate/index.html
suppressPackageStartupMessages({
  library(reticulate)
})
use_python("/usr/local/linux/anaconda3.7/bin/python3")
scvi <- import("scvi")
anndata <- import("anndata")
np <- import("numpy")
sc <- import("scanpy")
scvi_time <- system.time({
  scvi$settings$seed = 0L
  adata <- anndata$AnnData(X = as.sparse(t(counts(sce))),
                           obs = data.frame(cells = colnames(sce),
                                            batch = sce$donor))
  scvi$data$setup_anndata(adata, batch_key = "batch")
  model <- scvi$model$SCVI(adata)
  model$train(n_epochs = 100L, n_epochs_kl_warmup = 25L)
})
```

We can visualize the data using the labels from the original publication, and reducing the 10 dimensions of the latent space from **scvi** to 2 using t-SNE (van der Maaten and Hinton 2008, van der Maaten (2014), Krijthe (2015)).

```
library(scater)
```

```
## Loading required package: ggplot2
```

```
reducedDim(sce, "scvi") <- model$get_latent_representation()
denoised <- t(model$get_normalized_expression(adata, library_size = 10e4))
dimnames(denoised) <- dimnames(counts(sce))
assay(sce, "denoised") <- log1p(denoised)
sce <- runTSNE(sce, dimred = "scvi")
plotTSNE(sce, colour_by = "label")
```



As we can see, the **scvi** method clearly clusters correctly the original labels. Note however that this is information that would not available while analyzing a new dataset. One would instead need to rely on known-marker genes.

3 Creating inputs to Dune

Dune takes as input a set of clustering results. We therefore need to produce such a set. Here, we will demonstrate by producing three clustering results.

3.1 SC3

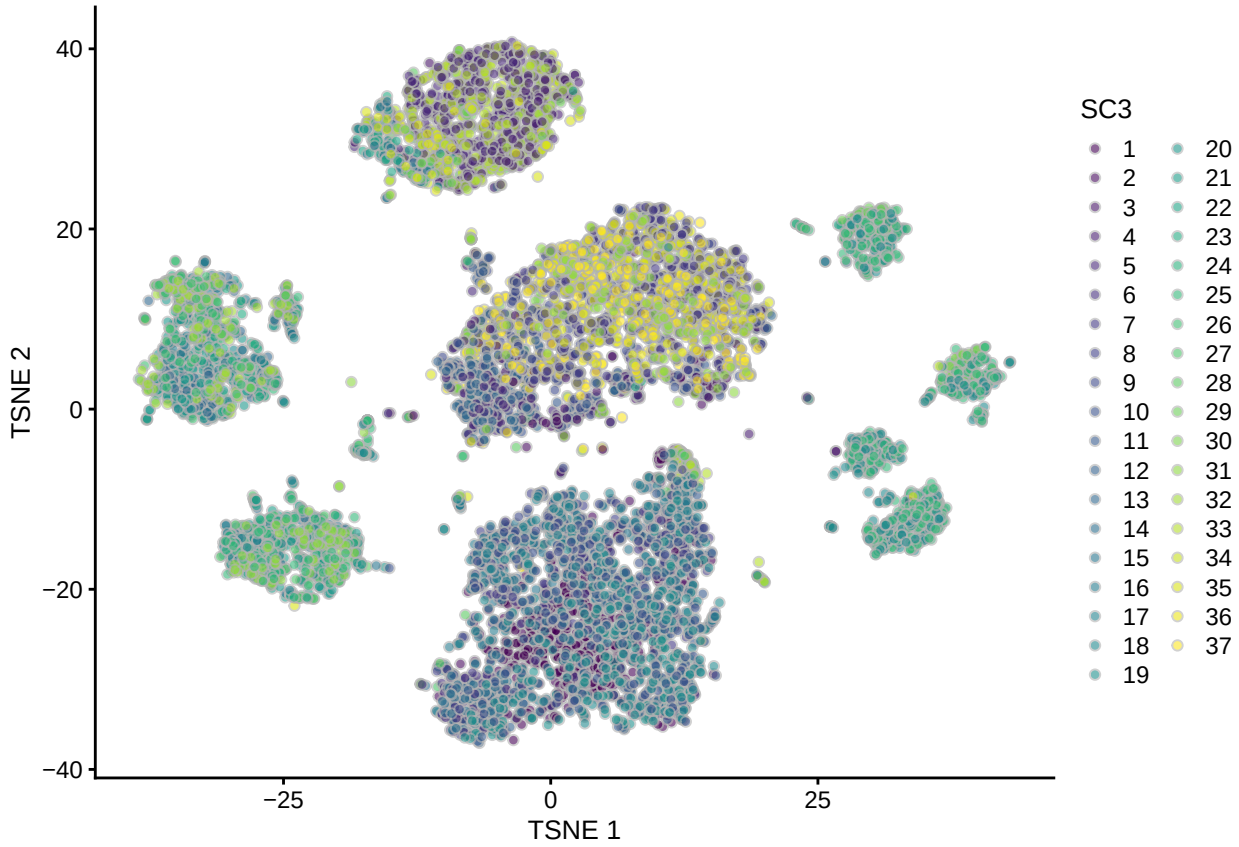
We first use **SC3** (Kiselev et al. 2017). We use as input the denoised count matrix from **scvi**. **SC3** has a function to estimate the value of K, the exact number of clusters.

Since the dataset has more than 5000 cells, **SC3** is automatically run in hybrid mode to lower runtime. However, the process is still quite slow. The code below is run in the default mode. If you want to run it faster, set `default=FALSE`

```
suppressPackageStartupMessages(library(SC3))
default <- TRUE
sc3_time <- system.time({
  sce_sc3 <- sce
  logcounts(sce_sc3) <- assay(sce, "denoised")
  rowData(sce_sc3)$feature_symbol <- rownames(sce_sc3)
  counts(sce_sc3) <- as.matrix(counts(sce_sc3))
```

```
  logcounts(sce_sc3) <- as.matrix(logcounts(sce_sc3))
  sce_sc3 <- sc3_estimate_k(sce_sc3)
  K <- metadata(sce_sc3)$sc3$k_estimation
  # Note: with R >= 4.0, RStudio and Mac OS, this can fails.
  # A workaround is running
  # parallel:::setDefaultClusterOptions(setup_strategy = "sequential")
  if (default) {
    sce_sc3 <- sc3(sce_sc3, ks = K, n_cores = NCORES, rand_seed = 786907)
  } else {
    sce_sc3 <- sc3(sce_sc3, ks = K, n_cores = NCORES, rand_seed = 786907,
                   svm_num_cells = round(.1 * ncol(sce)))
  }
  sce_sc3 <- sc3_run_svm(sce_sc3, ks = K)
  sce$SC3 <- colData(sce_sc3)[, paste0("sc3_", K, "_clusters")] %>% as.factor()
})
```

```
plotTSNE(sce, colour_by = "SC3")
```

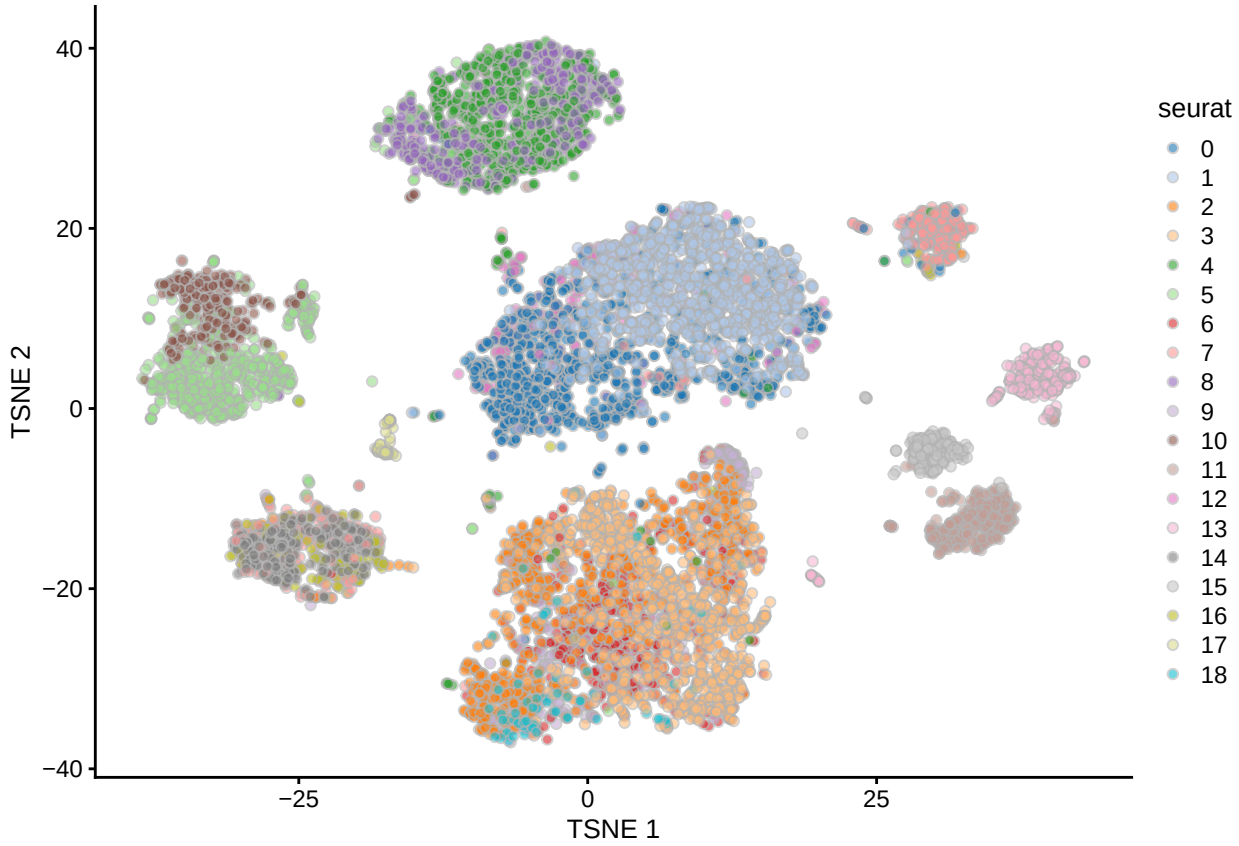


As we can see, **SC3** estimates too many clusters and, as a result, fails to properly cluster the data (according to the original clusters).

3.2 Seurat

The second method we use is **Seurat**, which first construct a Shared Nearest Neighbor (SNN) Graph and then runs the Louvain algorithm on the graph to identify clusters.

```
seurat_time <- system.time({
  se <- RunPCA(se, verbose = FALSE)
  se <- FindNeighbors(se, verbose = FALSE)
  se <- FindClusters(object = se, verbose = FALSE)
  sce$seurat <- Idents(se)
})
plotTSNE(sce, colour_by = "seurat")
```

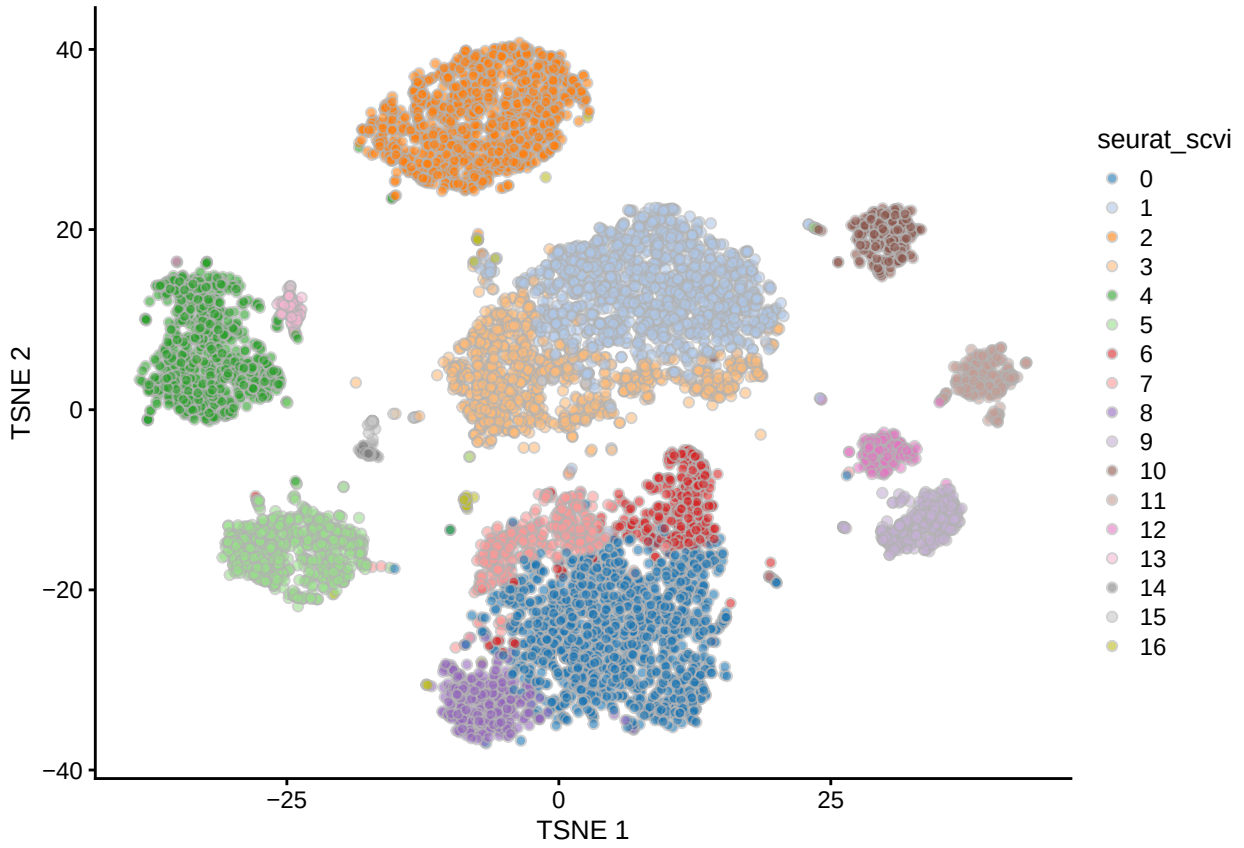


Seurat seems to perform better than **SC3** here but still seems to overpartition the data (with the default parameters).

3.3 Seurat with SCVI

Finally, we run the **Seurat** clustering workflow, but instead of building the SNN using the top 10 pcs, we build it using the latent space from **scvi**.

```
seurat_scvi_time <- system.time({
  seu <- as.Seurat(x = sce, counts = "counts", data = "counts")
  seu <- FindNeighbors(seu, reduction = "scvi", verbose = FALSE)
  seu <- FindClusters(object = seu, verbose = FALSE)
  sce$seurat_scvi <- Idents(seu)
})
plotTSNE(sce, colour_by = "seurat_scvi")
```



This seems to produce the best result, at least on the latent space of **scvi**, but still results in possible over-partition.

4 Dune

4.1 Running Dune

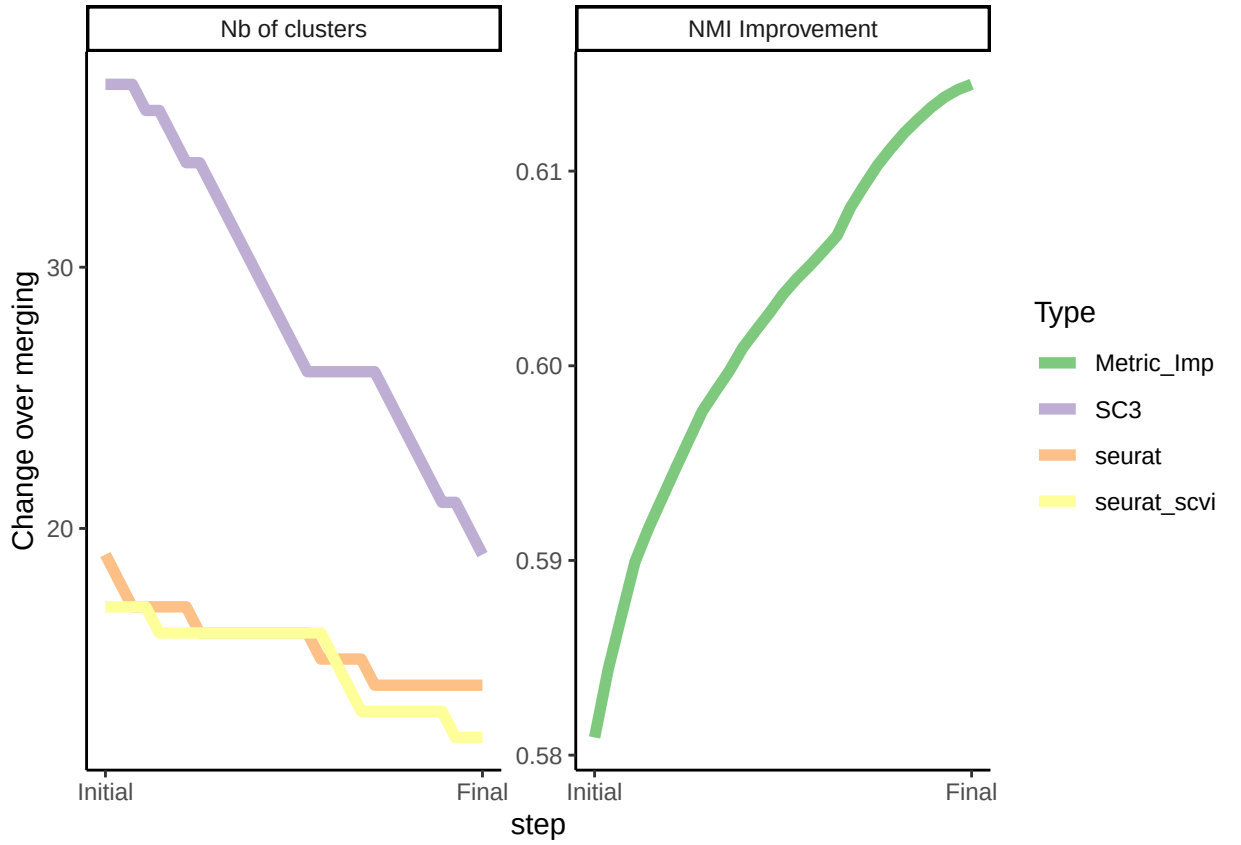
We can now run **Dune**, using the three clustering results as input. Since all clusterings seem to reflect over-partitioning of the data, **Dune** will aim to identify the common underlying structure and polish all inputs, using the Normalized Mutual Information (NMI) as a merging criterion.

```
library(Dune)
df <- colData(sce)[, c("SC3", "seurat", "seurat_scvi")] %>%
  as.matrix()
dune_time <- system.time(
  merger <- Dune(clusMat = df, metric = "NMI")
)
colData(sce)[, c("SC3_final", "seurat_final", "seurat_scvi_final")] <-
  lapply(merger$currentMat, as.factor) %>%
    as.data.frame()
```

4.2 Vizualing the merging

We can first see how the number of clusters in each clustering set decreased as merging occurred, and how the mean NMI increased when merging.

```
NMItrend(merger)
```



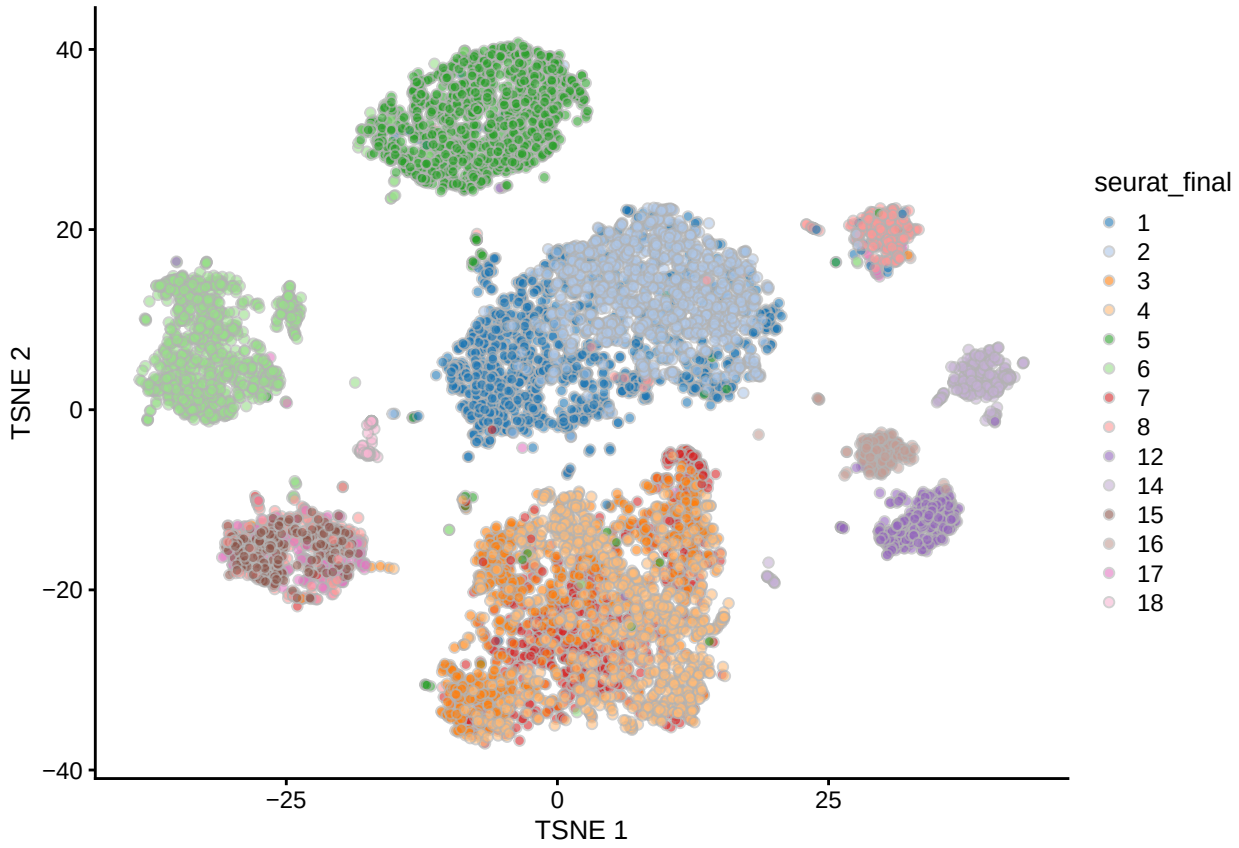
5 Picking the final clustering result to use

While **Dune** increases the concordance between the three sets of clusters, it does not pick one at the end. That choice remains to the user. **Dune** does not seek to replace biological knowledge or other metrics used to rank clustering methods. Instead, it aims to improve all its inputs, and to lessen the impact of the selection of one set of clusters.

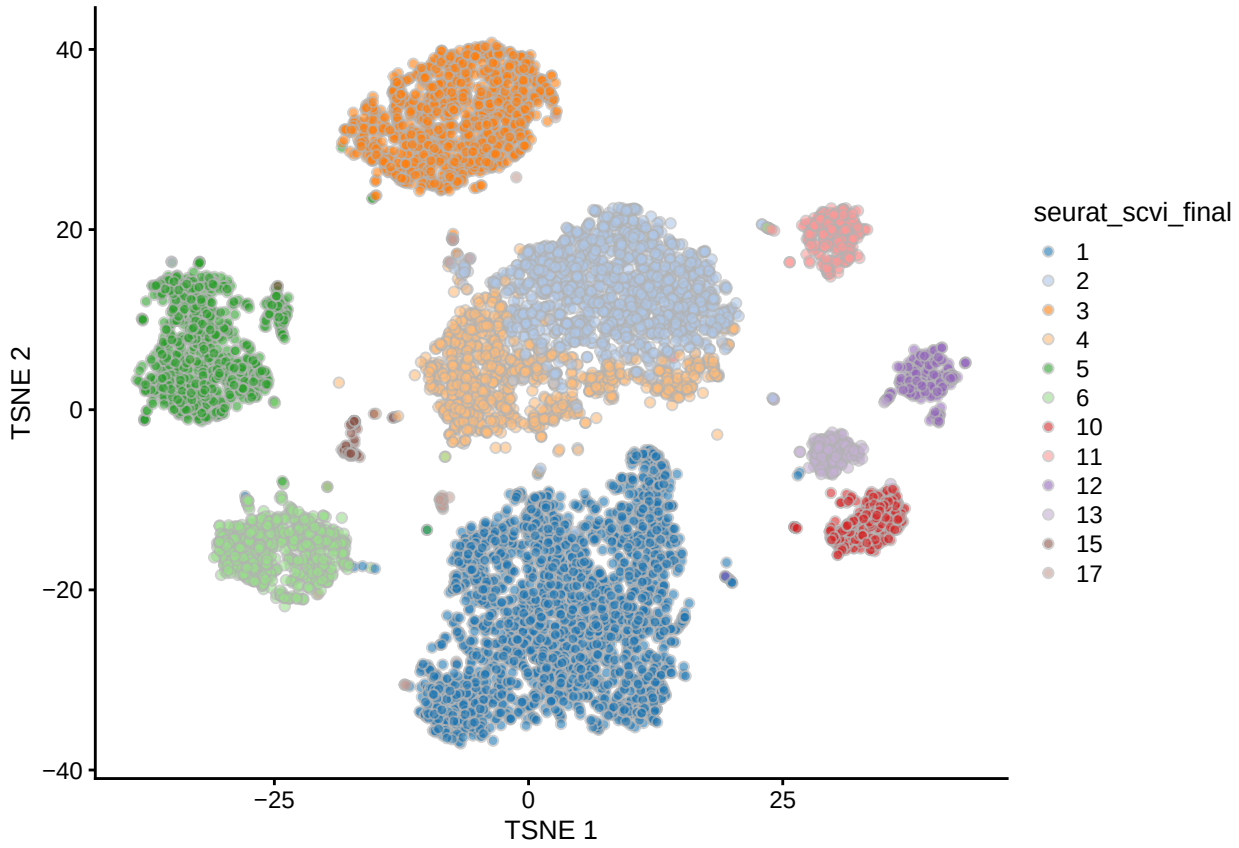
5.1 Manual selection

One common way to pick clustering results is still manual, using visualization.

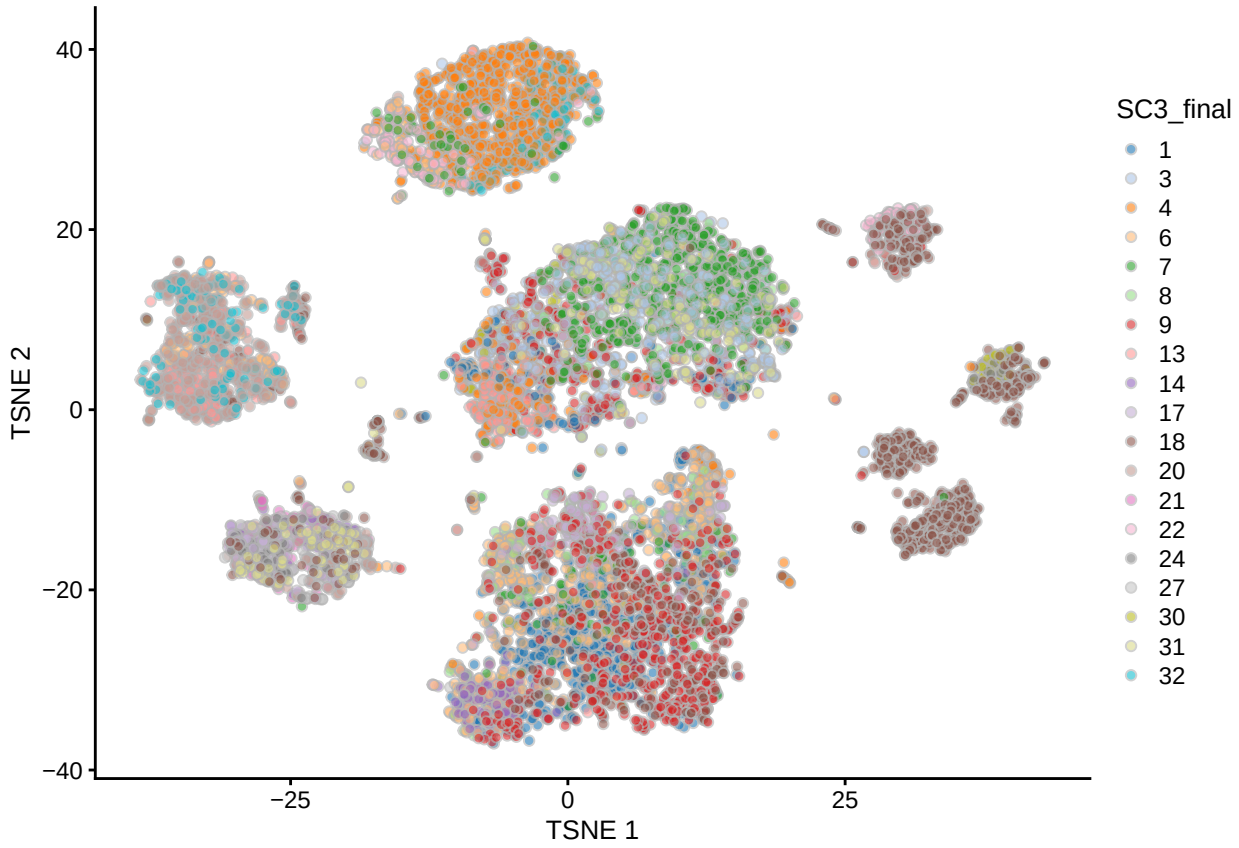
```
plotTSNE(sce, colour_by = "seurat_final")
```




```
plotTSNE(sce, colour_by = "seurat_scvi_final")
```




```
plotTSNE(sce, colour_by = "SC3_final")
```



Here, we can see that all clustering results look more consistent with the low dimensionality representation. Moreover, it clearly looks like **Seurat** using the latent space from **scvi** produces better results.

5.2 Selection based on sihouette

```
library(cluster)
dist_mat <- dist(as.matrix(reducedDim(sce, "scvi")))
sils_init <- lapply(merger$initialMat %>% as.data.frame, function(label){
  silhouette(label, dist = dist_mat)[,3] %>%  mean()
}) %>% unlist()
sils_init
```

```
##         SC3      seurat seurat_scvi
## -0.12061265  0.05173394  0.21896687
```

```
sils_final <- lapply(merger$currentMat %>% as.data.frame, function(label){
  silhouette(label, dist = dist_mat)[,3] %>%  mean()
}) %>% unlist()
sils_final
```

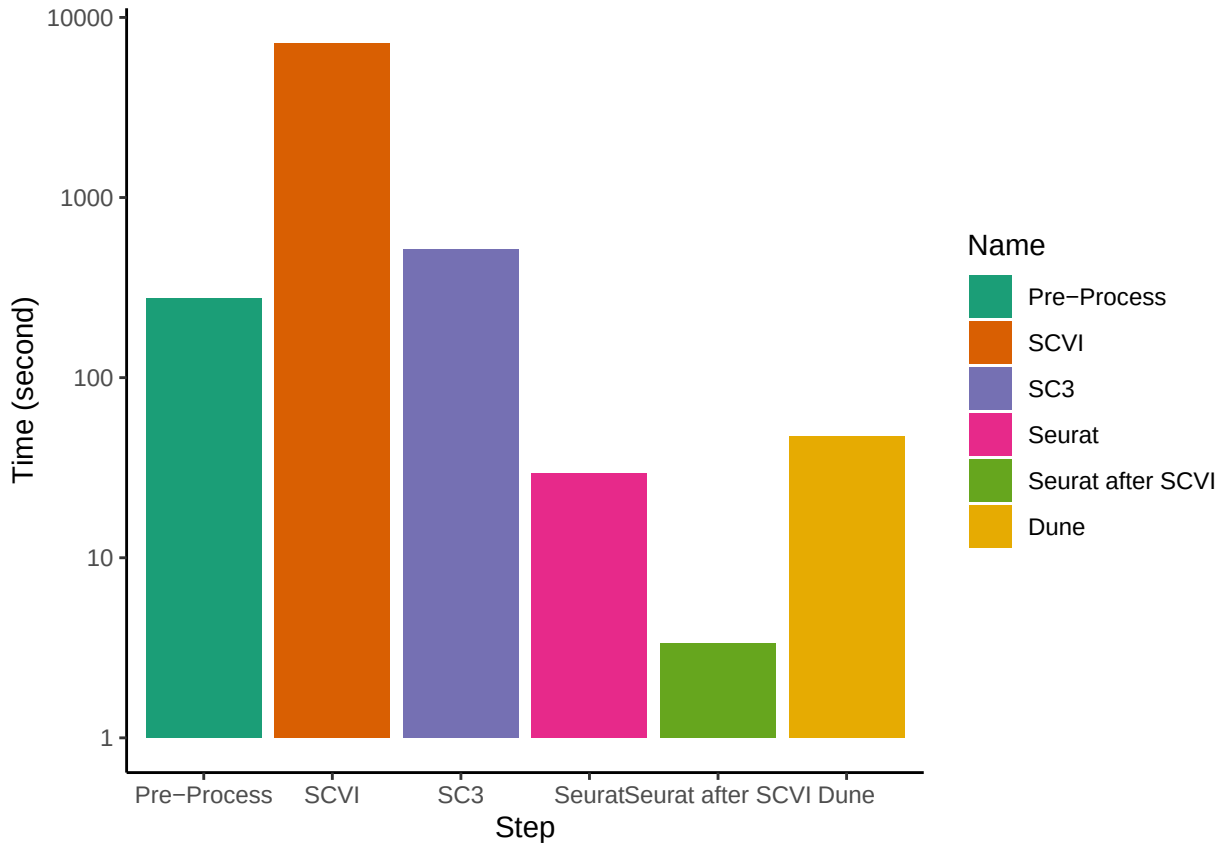
```
##         SC3      seurat seurat_scvi
## -0.05884367  0.13027202  0.23961572
```

For all methods, the average silhouette information increased after merging with **Dune**. On that metric, all methods are improved with the merging. Moreover, and consistent with the visual representation, **Seurat** using the latent space from **scvi** clearly outperforms the other two. That is the one that will be used.

6 Runtimes

We can also compare the runtimes of all parts of the workflow. Running **SC3** in default mode is quite slow, followed by **scvi**. Running **Dune** itself is quite quick compared to other steps.

```
times <- c(pre_process_time[1],
           scvi_time[1],
           sc3_time[1],
           seurat_time[1],
           seurat_scvi_time[1],
           dune_time[1])
names(times) <- c("Pre-Process",
                  "SCVI",
                  "SC3",
                  "Seurat",
                  "Seurat after SCVI",
                  "Dune")
df <- data.frame(times = times,
                 Name = factor(names(times), levels = names(times)))
ggplot(df, aes(x = Name, y = times, fill = Name)) +
  geom_col() +
  theme_classic() +
  labs(x = "Step", y = "Time (second)") +
  scale_fill_brewer(palette = "Dark2") +
  scale_y_log10()
```



7 Session

```
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so
##
## 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
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] cluster_2.1.0               Dune_1.1.0
##  [3] SC3_1.14.0                  scater_1.14.6
##  [5] ggplot2_3.3.1               Seurat_3.2.1
##  [7] scRNAseq_2.0.2              stringr_1.4.0
##  [9] SingleCellExperiment_1.8.0  SummarizedExperiment_1.16.1
## [11] DelayedArray_0.12.3         BiocParallel_1.20.1
## [13] matrixStats_0.56.0          Biobase_2.46.0
## [15] GenomicRanges_1.38.0        GenomeInfoDb_1.22.1
## [17] IRanges_2.20.2              S4Vectors_0.24.4
## [19] BiocGenerics_0.32.0         knitr_1.28
## [21] rmarkdown_2.2               reticulate_1.13
## [23] SCF_3.6.1
##
## loaded via a namespace (and not attached):
##   [1] tidyselect_1.1.0              RSQLite_2.1.2
##   [3] AnnotationDbi_1.48.0          htmlwidgets_1.5.1
##   [5] grid_3.6.1                    Rtsne_0.15
##   [7] munsell_0.5.0                 codetools_0.2-16
##   [9] ica_1.0-2                     future_1.17.0
##  [11] miniUI_0.1.1.1                withr_2.1.2
##  [13] colorspace_1.4-1              ROCR_1.0-11
##  [15] robustbase_0.93-5             tensor_1.5
##  [17] listenv_0.8.0                 labeling_0.3
##  [19] GenomeInfoDbData_1.2.2        polyclip_1.10-0
##  [21] farver_2.0.1                  bit64_0.9-7
##  [23] pheatmap_1.0.12               vctrs_0.3.0
##  [25] generics_0.0.2                xfun_0.14
##  [27] BiocFileCache_1.10.2          R6_2.4.0
```

```
##  [29] doParallel_1.0.15             ggbeeswarm_0.6.0
##  [31] rsvd_1.0.2                    bitops_1.0-6
##  [33] spatstat.utils_1.17-0         assertthat_0.2.1
##  [35] promises_1.1.0                scales_1.0.0
##  [37] beeswarm_0.2.3                gtable_0.3.0
##  [39] npsurv_0.4-0                  globals_0.12.5
##  [41] goftest_1.2-2                 rlang_0.4.6
##  [43] splines_3.6.1                 lazyeval_0.2.2
##  [45] BiocManager_1.30.10           yaml_2.2.0
##  [47] reshape2_1.4.3                abind_1.4-5
##  [49] httpuv_1.5.3.1                tools_3.6.1
##  [51] ellipsis_0.3.1                RColorBrewer_1.1-2
##  [53] ggridges_0.5.1                Rcpp_1.0.5
##  [55] plyr_1.8.4                    progress_1.2.2
##  [57] zlibbioc_1.32.0               purrr_0.3.4
##  [59] RCurl_1.95-4.12               prettyunits_1.0.2
##  [61] rpart_4.1-15                  deldir_0.1-23
##  [63] pbapply_1.4-2                 viridis_0.5.1
##  [65] cowplot_1.0.0                 zoo_1.8-6
##  [67] ggrepel_0.8.1                 magrittr_1.5
##  [69] data.table_1.12.8             lmtest_0.9-37
##  [71] RANN_2.6.1                    mvtnorm_1.0-11
##  [73] fitdistrplus_1.0-14           hms_0.5.3
##  [75] patchwork_1.0.0               lsei_1.2-0
##  [77] mime_0.7                      evaluate_0.14
##  [79] xtable_1.8-4                  gridExtra_2.3
##  [81] compiler_3.6.1                tibble_2.1.3
##  [83] KernSmooth_2.23-15            crayon_1.3.4
##  [85] htmltools_0.4.0               mgcv_1.8-28
##  [87] pcaPP_1.9-73                  later_1.0.0
##  [89] tidyr_1.1.0                   rrcov_1.4-7
##  [91] DBI_1.0.0                     tweenr_1.0.1
##  [93] ExperimentHub_1.12.0          WriteXLS_5.0.0
##  [95] dbplyr_1.4.2                  MASS_7.3-51.4
##  [97] rappdirs_0.3.1                Matrix_1.2-17
##  [99] igraph_1.2.4.1                pkgconfig_2.0.3
## [101] registry_0.5-1                plotly_4.9.0
## [103] foreach_1.4.7                 vipor_0.4.5
## [105] rngtools_1.4                  pkgmaker_0.27
## [107] XVector_0.26.0                bibtex_0.4.2
## [109] doRNG_1.7.1                   digest_0.6.25
## [111] sctransform_0.2.0             RcppAnnoy_0.0.16
## [113] spatstat.data_1.4-3           leiden_0.3.3
## [115] uwot_0.1.8                    DelayedMatrixStats_1.8.0
## [117] curl_4.3                      shiny_1.4.0.2
## [119] lifecycle_0.2.0               nlme_3.1-140
## [121] aricode_1.0.0                 jsonlite_1.6.1
## [123] BiocNeighbors_1.4.2           viridisLite_0.3.0
## [125] pillar_1.4.2                  lattice_0.20-38
## [127] fastmap_1.0.1                 httr_1.4.1
## [129] DEoptimR_1.0-8                survival_2.44-1.1
## [131] interactiveDisplayBase_1.24.0 glue_1.4.1
## [133] gganimate_1.0.6               spatstat_1.64-1
## [135] png_0.1-7                     iterators_1.0.12
```

```
## [137] BiocVersion_3.10.1           bit_1.1-14
## [139] class_7.3-15                 stringi_1.4.3
## [141] blob_1.2.0                   BiocSingular_1.2.2
## [143] AnnotationHub_2.18.0         memoise_1.1.0
## [145] dplyr_1.0.0                  irlba_2.3.3
## [147] e1071_1.7-2                  future.apply_1.3.0
```

References

Baron, Maayan, Adrian Veres, Samuel L. Wolock, Aubrey L. Faust, Renaud Gaujoux, Amedeo Vetere, Jennifer Hyoje Ryu, et al. 2016. "A Single-Cell Transcriptomic Map of the Human and Mouse Pancreas Reveals Inter- and Intra-cell Population Structure." *Cell Systems* 3 (4). Cell Press: 346–360.e4. doi:10.1016/j.cels.2016.08.011.

Cao, Junyue, Malte Spielmann, Xiaojie Qiu, Xingfan Huang, Daniel M. Ibrahim, Andrew J. Hill, Fan Zhang, et al. 2019. "The single-cell transcriptional landscape of mammalian organogenesis." *Nature* 566 (7745). Nature Publishing Group: 496–502. doi:10.1038/s41586-019-0969-x.

Kiselev, Vladimir Yu, Kristina Kirschner, Michael T Schaub, Tallulah Andrews, Andrew Yiu, Tamir Chandra, Kedar N Natarajan, et al. 2017. "SC3: Consensus clustering of single-cell RNA-seq data." *Nature Methods* 14 (5). Nature Publishing Group: 483–86. doi:10.1038/nmeth.4236.

Krijthe, Jesse H. 2015. *Rtsne: T-Distributed Stochastic Neighbor Embedding Using Barnes-Hut Implementation*. https://github.com/jkrijthe/Rtsne.

Lopez, Romain, Jeffrey Regier, Michael B. Cole, Michael I. Jordan, and Nir Yosef. 2018. "Deep generative modeling for single-cell transcriptomics." *Nature Methods* 15 (12). Nature Publishing Group: 1053–8. doi:10.1038/s41592-018-0229-2.

van der Maaten, L.J.P. 2014. "Accelerating T-Sne Using Tree-Based Algorithms." *Journal of Machine Learning Research* 15: 3221–45.

van der Maaten, L.J.P., and G.E. Hinton. 2008. "Visualizing High-Dimensional Data Using T-Sne." *Journal of Machine Learning Research* 9: 2579–2605.