Dune workflow on the baron dataset

Hector Roux de Bézieux

30 April , 2021

Contents

In this workflow, we will demonstrate a full full scRNA-Seq workflow using **Dune** on an example dataset. We rely on the the data from (Baron et al. 2016), a human pancreas dataset of 8569 samples. We will demonstrate how to generate various input clustering results, how to merge clusters using **Dune** and how to select the best output for use in downstream analysis. We will also monitor run times to show the impact of running **Dune** versus a workflow without it.

1 Load data

We rely on a pre-processed dataset where the count matrix has already been computed, using the **scRNAseq** R package. The dataset also contains the id of the human donor for each cell, which are used as batch labels. It also contains the cell labels assignements from the original publication. Note that, in that publication, cells were clustered using hierarchical clustering with a final manual merging step.

```
set.seed(19)
suppressPackageStartupMessages({
  library(SingleCellExperiment)
  library(stringr)
  library(scRNAseq)
})
# Load pre-processed dataset
sce <- BaronPancreasData()
# Filter very lowly expressed genes for computational practices.
filt <- rowSums(counts(sce) >= 2) >= 10
sce <- sce[filt, ]
print(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):
## altExpNames(0):
```

2 Pre-processing

Before running clustering algorithms, we will rely on two normalization pipelines.

- The default pipeline of **Seurat** (Stuart et al. 2019).

```
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
```

- The **scvi** method (Lopez et al. 2018).

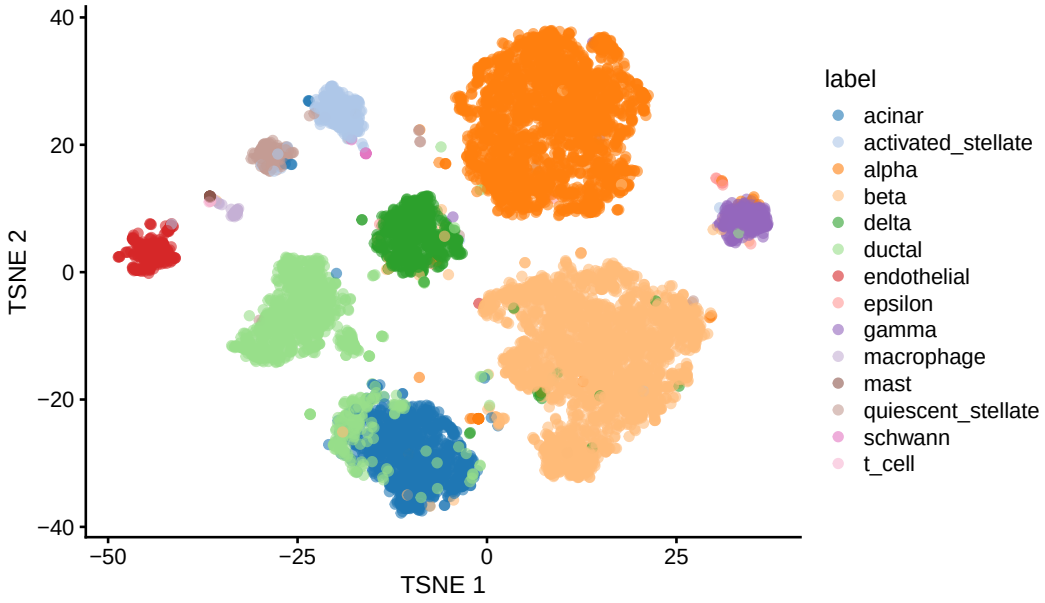
```
suppressPackageStartupMessages(library(reticulate))
scvi <- import('scvi', convert = FALSE)
anndata <- import("anndata")
np <- import("numpy")
sc <- import("scanpy")
scvi_time <- system.time({
  scvi$settings$seed = 0L
  adata <- anndata$AnnData(X = as.sparse(t(counts(sce))),
                           obs = data.frame(cells = colnames(sce),
                                            batch = sce$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 latent space produced by **scvi** using the labels from the original publication, and reducing the 10 dimensions of the latent space to 2 using t-SNE (van der Maaten and Hinton 2008, @tsne2, @tsne3).

```
suppressPackageStartupMessages(library(scater))
```

```
## Warning: package 'scater' was built under R version 4.0.4
```

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



As we can see, **scvi** mostly produces a latent space that is consistent with 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 will generate a set of such results using a combination of clustering methods and normalization techniques:

- **SC3** (Kiselev et al. 2017) using as input the denoised count matrix from **scvi**.
- **Seurat** using as input the latent space from **scvi**.
- **Seurat** using as input the top pcs from the count matrix normalized using the **Seurat** pre-processing pipeline.

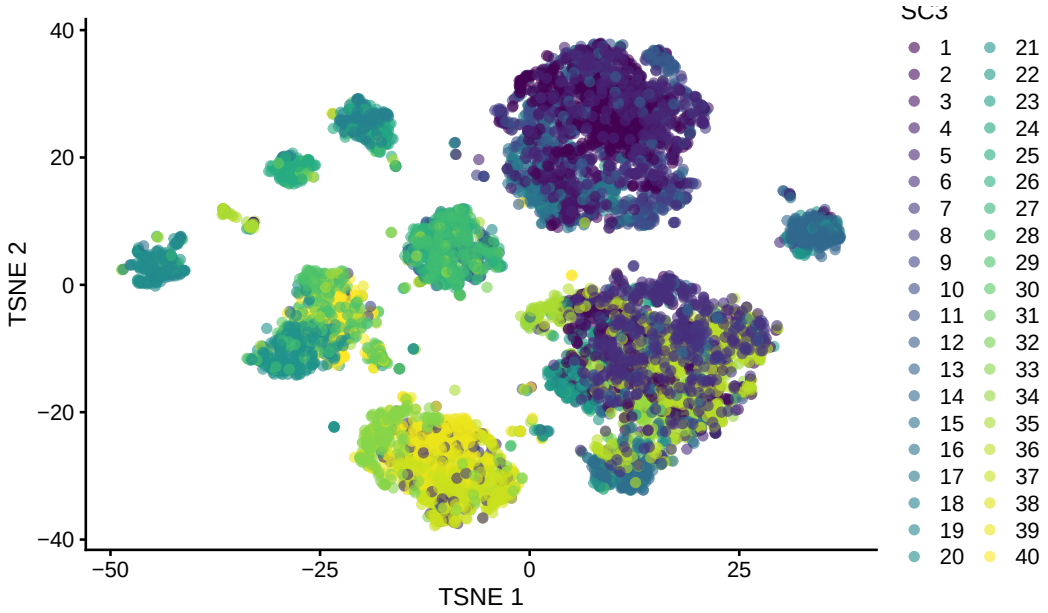
3.1 SC3

SC3 is a consensus method that takes as input a normalized count matrix and outputs a set of cluster labels. The **SC3** package provides a function to estimate the value of K, the exact number of clusters, which we will use.

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

```
suppressPackageStartupMessages(library(SC3))
default <- FALSE
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")
```



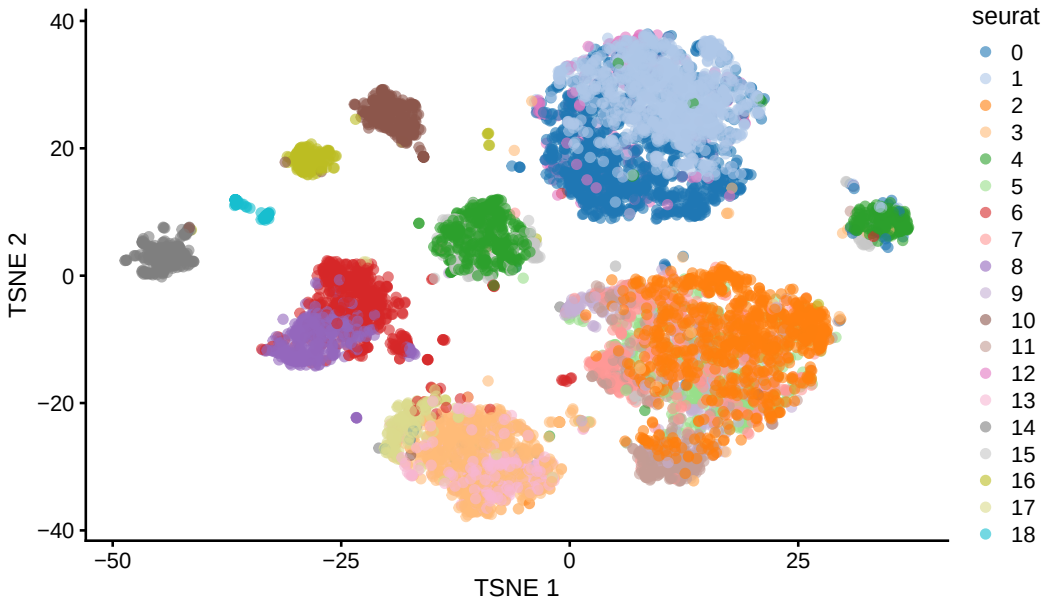
As we can see, **SC3** seems to overcluster the data, when compared either to the labels from the original publication, or to the reduced dimension representation. However, this is not a problem since **Dune** will work better on overclustered results.

3.2 Seurat

The second method we use is the clustering algorithm from the **Seurat** R package, which first construct a Shared Nearest Neighbor (SNN) Graph and then runs the Louvain algorithm on the graph to identify clusters. The SNN graph is build using a reduced dimension representation of the dataset. We first use the default, which is to use the top pcs from the normalized count matrix.

```
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")
```



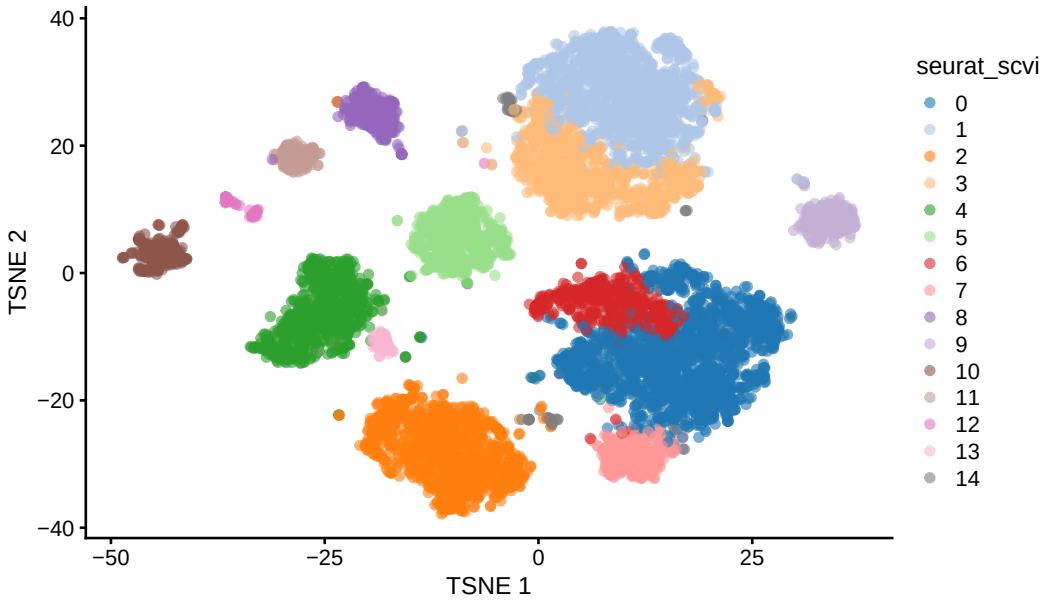
Seurat seems to perform better than **SC3** here but still seems to overpartition the data when run with the default parameters. Once again, it will not be a problem if used as input to **Dune**.

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**, which is not suprising. It also better matches the labels from the original publication 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 identify the common underlying structure and polish all inputs, using the Normalized Mutual Information (NMI) as a merging criterion.

```
library(Dune)
```

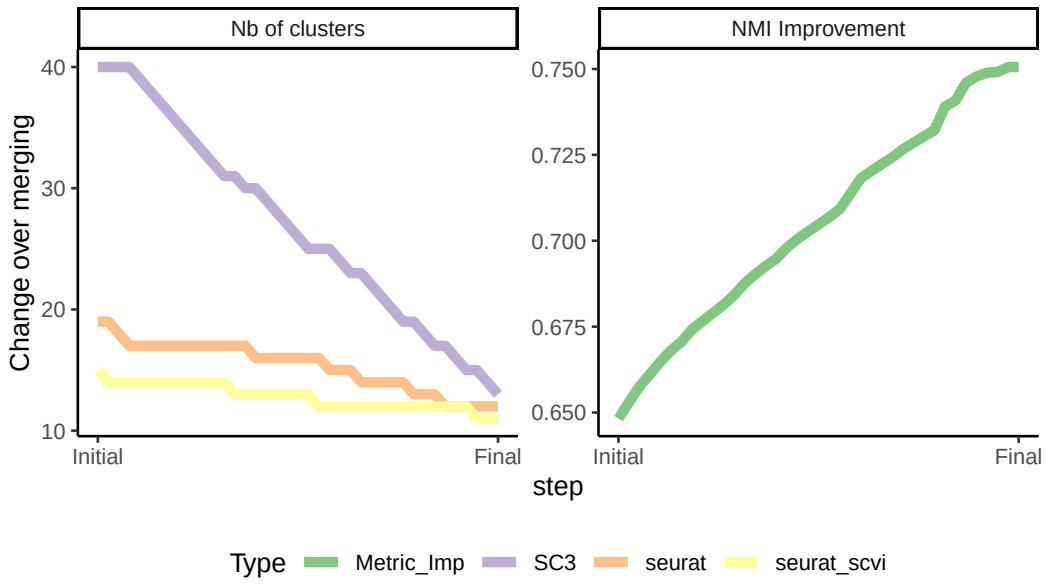
```
## Dune now uses the Normalized Mutual Informationinstead of the adjusted Rand Index. You can res
```

```
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) + theme(legend.position = "bottom")
```



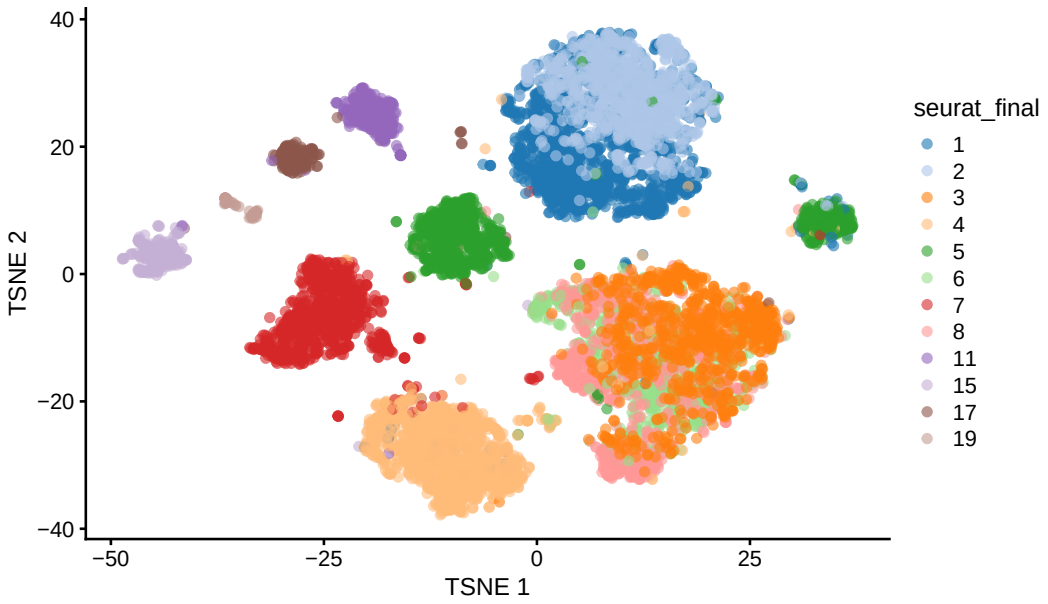
5 Picking the final clustering result to use

While **Dune** increases the concordance between the three sets of clusters, it does not pick one at the end. That choice remains up 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.

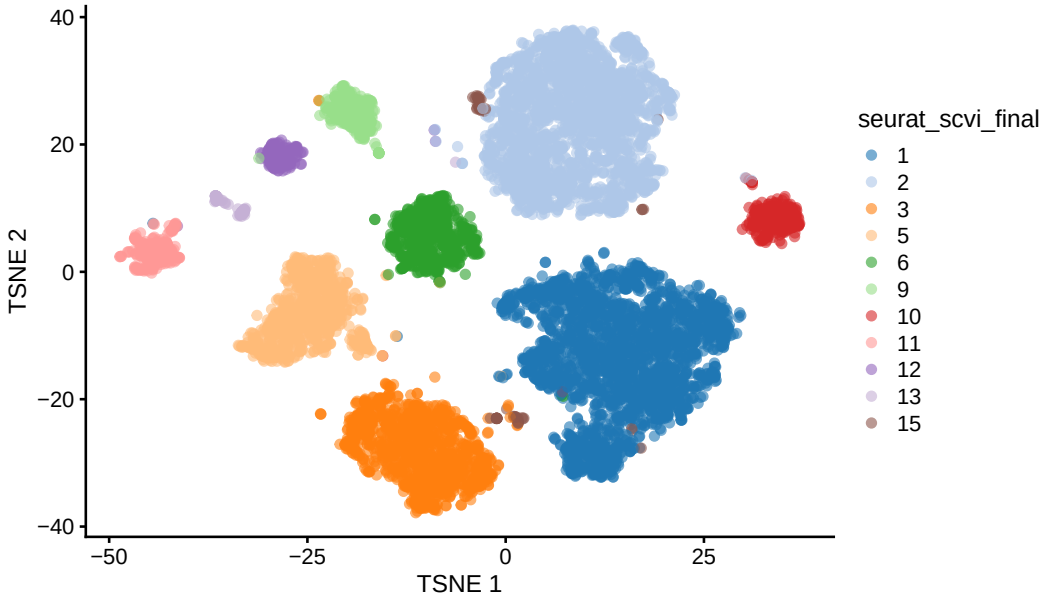
5.1 Manual selection

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

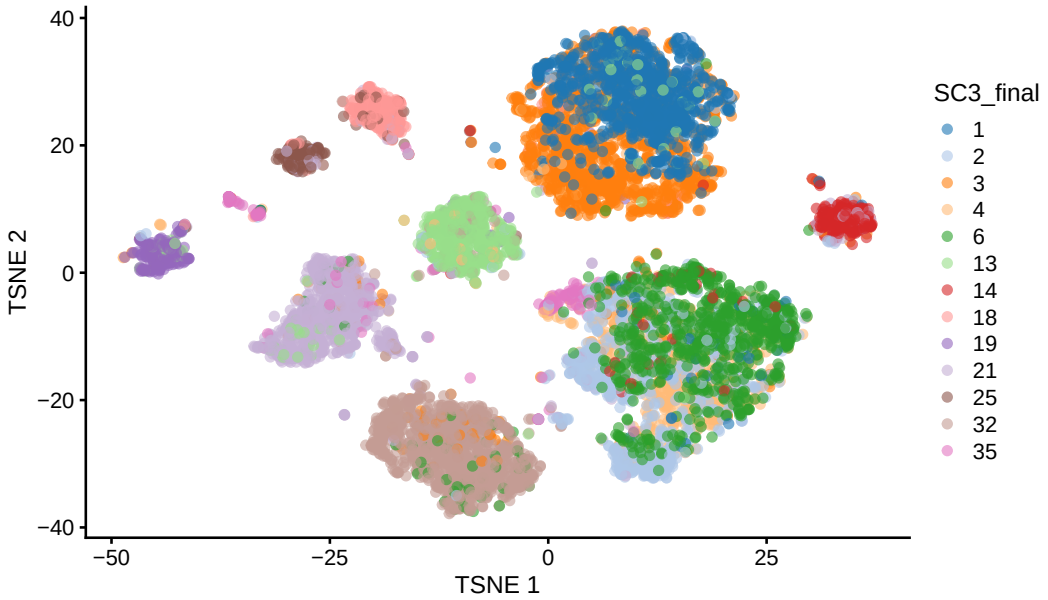
```
plotTSNE(sce, colour_by = "seurat_final")
```



```
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

To provide a more quantitative selection criterion, we can rely on the average silhouette width. This is a number between -1 and 1 that quantify the quality of clustering using the distance matrix between all cells. We compute the distance on the **scvi** latent space.

```
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
## -0.02385097  0.05998455  0.21037407
```

This confirm the visual impression: the cluster labels from **Seurat_scvi** are clearly better on this dataset than the others before merging with **Dune**.

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

```
##         SC3      seurat seurat_scvi
##  0.09578247  0.15157494  0.25300568
```

For all methods, the average silhouette information increased after merging with **Dune**. Even the best method, **Seurat_scvi**, is improved by the merging. However, the ranking of methods is unchanged: consistent with the visual representation, **Seurat** using the latent space from **scvi** clearly outperforms the other two. That is the one that should be used for downstream analysis such as trajectory inference, differential expression or cell type annotation.

5.3 Comparing with the original labels

This last step is not possible on a normal analysis of a new dataset. However, here, we can see how, running all methods using default, we recover cluster labels that match closely clusters from the original publication that had require manual merging using outside biological knowledge.

```
suppressPackageStartupMessages({
  library(aricode)
  library(mclust)
})
NMI(sce$label, sce$seurat_scvi_final) %>% round(2)
```

```
## [1] 0.91
```

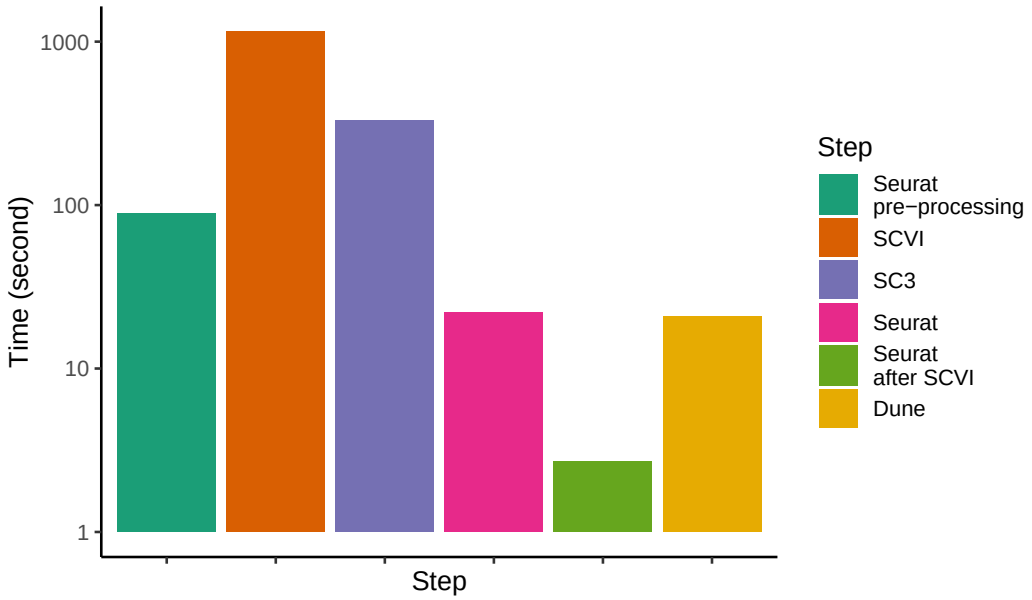
```
adjustedRandIndex(sce$label, sce$seurat_scvi_final) %>% round(2)
```

```
## [1] 0.93
```

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. Using **Dune** in a workflow increased total runtime but not by orders of magnitudes.

```
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("Seurat\npre-processing",
                  "SCVI",
                  "SC3",
                  "Seurat",
                  "Seurat\nafter 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)", fill = "Step") +
  scale_fill_brewer(palette = "Dark2") +
  theme(axis.text.x = element_blank()) +
  scale_y_log10() +
  guides(col = FALSE)
```




```
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Big Sur 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] mclust_5.4.7                aricode_1.0.0
##  [3] cluster_2.1.1               Dune_1.3.01
##  [5] SC3_1.18.0                  scater_1.18.6
##  [7] ggplot2_3.3.3               reticulate_1.18
##  [9] SeuratObject_4.0.0          Seurat_4.0.1
## [11] scRNAseq_2.4.0              stringr_1.4.0
## [13] SingleCellExperiment_1.12.0 SummarizedExperiment_1.20.0
## [15] Biobase_2.50.0              GenomicRanges_1.42.0
## [17] GenomeInfoDb_1.26.5         IRanges_2.24.1
```

```
## [19] S4Vectors_0.28.1            BiocGenerics_0.36.0
## [21] MatrixGenerics_1.2.1        matrixStats_0.58.0
## [23] knitr_1.31
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.1                    tidyselect_1.1.0
##   [3] RSQLite_2.2.5                 AnnotationDbi_1.52.0
##   [5] htmlwidgets_1.5.3             grid_4.0.3
##   [7] BiocParallel_1.24.1           Rtsne_0.15
##   [9] munsell_0.5.0                 codetools_0.2-18
##  [11] ica_1.0-2                     future_1.21.0
##  [13] miniUI_0.1.1.1                withr_2.4.1
##  [15] colorspace_2.0-0              ROCR_1.0-11
##  [17] robustbase_0.93-7             tensor_1.5
##  [19] listenv_0.8.0                 labeling_0.4.2
##  [21] GenomeInfoDbData_1.2.4        polyclip_1.10-0
##  [23] farver_2.1.0                  pheatmap_1.0.12
##  [25] bit64_4.0.5                   parallelly_1.24.0
##  [27] vctrs_0.3.7                   generics_0.1.0
##  [29] xfun_0.22                     BiocFileCache_1.14.0
##  [31] doParallel_1.0.16             R6_2.5.0
##  [33] ggbeeswarm_0.6.0              rsvd_1.0.3
##  [35] AnnotationFilter_1.14.0       bitops_1.0-6
##  [37] spatstat.utils_2.1-0          cachem_1.0.4
##  [39] DelayedArray_0.16.3           assertthat_0.2.1
##  [41] promises_1.2.0.1              scales_1.1.1
##  [43] beeswarm_0.3.1                gtable_0.3.0
##  [45] beachmat_2.6.4                globals_0.14.0
##  [47] goftest_1.2-2                 ensembldb_2.14.0
##  [49] rlang_0.4.10                  splines_4.0.3
##  [51] rtracklayer_1.50.0            lazyeval_0.2.2
##  [53] spatstat.geom_2.0-1           BiocManager_1.30.12
##  [55] yaml_2.2.1                    reshape2_1.4.4
##  [57] abind_1.4-5                   GenomicFeatures_1.42.3
##  [59] httpuv_1.5.5                  tools_4.0.3
##  [61] ellipsis_0.3.1                spatstat.core_2.0-0
##  [63] RColorBrewer_1.1-2            proxy_0.4-25
##  [65] ggridges_0.5.3                Rcpp_1.0.6
##  [67] plyr_1.8.6                    sparseMatrixStats_1.2.1
##  [69] progress_1.2.2                zlibbioc_1.36.0
##  [71] purrr_0.3.4                   RCurl_1.98-1.3
##  [73] prettyunits_1.1.1             rpart_4.1-15
##  [75] openssl_1.4.3                 deldir_0.2-10
##  [77] viridis_0.5.1                 pbapply_1.4-3
##  [79] cowplot_1.1.1                 zoo_1.8-9
##  [81] ggrepel_0.9.1                 magrittr_2.0.1
##  [83] magick_2.7.1                  data.table_1.14.0
##  [85] scattermore_0.7               lmtest_0.9-38
```

```
##  [87] RANN_2.6.1                    mvtnorm_1.1-1
##  [89] ProtGenerics_1.22.0           fitdistrplus_1.1-3
##  [91] hms_1.0.0                     patchwork_1.1.1
##  [93] mime_0.10                     evaluate_0.14
##  [95] xtable_1.8-4                  XML_3.99-0.6
##  [97] gridExtra_2.3                 compiler_4.0.3
##  [99] biomaRt_2.46.3                tibble_3.1.0
## [101] KernSmooth_2.23-18            crayon_1.4.1
## [103] htmltools_0.5.1.1             pcaPP_1.9-73
## [105] mgcv_1.8-34                   later_1.1.0.1
## [107] rrcov_1.5-5                   tidyr_1.1.3
## [109] DBI_1.1.1                     tweenr_1.0.2
## [111] ExperimentHub_1.16.0          WriteXLS_6.3.0
## [113] dbplyr_2.1.1                  MASS_7.3-53.1
## [115] rappdirs_0.3.3                Matrix_1.3-2
## [117] igraph_1.2.6                  pkgconfig_2.0.3
## [119] GenomicAlignments_1.26.0      plotly_4.9.3
## [121] scuttle_1.0.4                 spatstat.sparse_2.0-0
## [123] foreach_1.5.1                 xml2_1.3.2
## [125] vipor_0.4.5                   rngtools_1.5
## [127] XVector_0.30.0                doRNG_1.8.2
## [129] digest_0.6.27                 sctransform_0.3.2
## [131] RcppAnnoy_0.0.18              spatstat.data_2.1-0
## [133] Biostrings_2.58.0             rmarkdown_2.7
## [135] leiden_0.3.7                  uwot_0.1.10
## [137] DelayedMatrixStats_1.12.3     curl_4.3
## [139] shiny_1.6.0                   Rsamtools_2.6.0
## [141] lifecycle_1.0.0               nlme_3.1-152
## [143] jsonlite_1.7.2                BiocNeighbors_1.8.2
## [145] viridisLite_0.3.0             askpass_1.1
## [147] fansi_0.4.2                   pillar_1.5.1
## [149] lattice_0.20-41               DEoptimR_1.0-8
## [151] fastmap_1.1.0                 httr_1.4.2
## [153] survival_3.2-10               gganimate_1.0.7
## [155] interactiveDisplayBase_1.28.0 glue_1.4.2
## [157] iterators_1.0.13              png_0.1-7
## [159] BiocVersion_3.12.0            bit_4.0.4
## [161] class_7.3-18                  stringi_1.5.3
## [163] blob_1.2.1                    BiocSingular_1.6.0
## [165] AnnotationHub_2.22.0          memoise_2.0.0
## [167] dplyr_1.0.5                   e1071_1.7-6
## [169] irlba_2.3.3                   future.apply_1.7.0
```

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): 346–360.e4. https://doi.org/10.1016/j.cels.2016.08.011.

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): 483–86. https://doi.org/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): 1053–8. https://doi.org/10.1038/s41592-018-0229-2.

Stuart, Tim, Andrew Butler, Paul Hoffman, Christoph Hafemeister, Efthymia Papalexi, William M Mauck, Yuhan Hao, Marlon Stoeckius, Peter Smibert, and Rahul Satija. 2019. "Comprehensive Integration of Single-Cell Data." *Cell* 177 (7): 1888–1902.e21. https://doi.org/10.1016/j.cell.2019.05.031.

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.