Data Integration

Ahmed Mahfouz and Indu Khatri

11/16/2021

Overview

In this tutorial we will look at different ways of integrating multiple single cell RNA-seq datasets. We will explore two different methods to correct for batch effects across datasets. At the end of the session, we will also show how to transfer cell type labels from a reference dataset to a new dataset.

Datasets

For this tutorial, we will use the 3 different PBMC datasets we also started the normalization practical with.

Load required packages:

```
# Clear the workspace
rm(list=ls())
suppressMessages(require(Seurat))
suppressMessages(require(SeuratDisk))
```

Seurat (anchors and CCA)

First we will use the data integration method presented in Comprehensive Integration of Single Cell Data.

Data preprocessing

First, we load the three datasets and some celltype labels.

```
pbmc_v3.1k <- readRDS('pbmc3k_Clust.rds')
v2.1k <- Read10X_h5("pbmc_1k_v2_filtered_feature_bc_matrix.h5")

## Warning in sparseMatrix(i = indices[] + 1, p = indptr[], x = as.numeric(x
=
## counts[]), : 'giveCsparse' has been deprecated; setting 'repr = "T"' for
you

p3.1k <- Read10X_h5("pbmc_1k_protein_v3_filtered_feature_bc_matrix.h5")

## Warning in sparseMatrix(i = indices[] + 1, p = indptr[], x = as.numeric(x
=
## counts[]), : 'giveCsparse' has been deprecated; setting 'repr = "T"' for
you
```

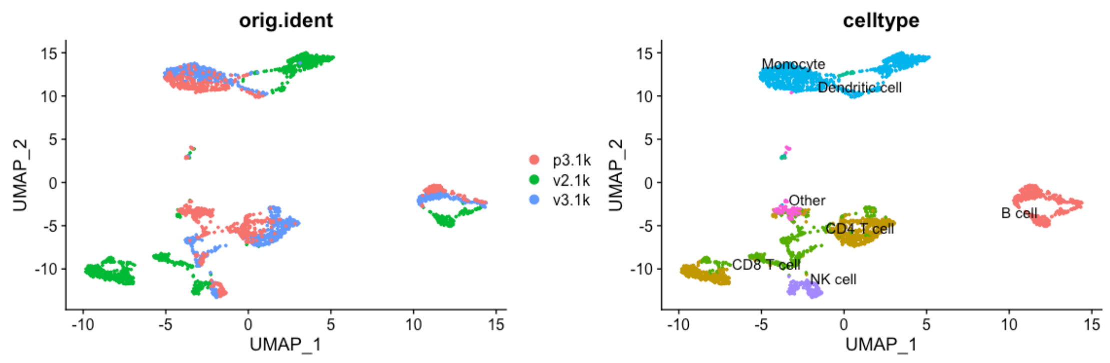
```
## Genome matrix has multiple modalities, returning a list of matrices for
this genome

p3.1k <- p3.1k$`Gene Expression`

pbmc_v2.1k <- CreateSeuratObject(v2.1k, project = "v2.1k")
pbmc_p3.1k <- CreateSeuratObject(p3.1k, project = "p3.1k")

labels_v2.1k = read.delim('celltypes_1k_v2.tsv', row.names = 1)
labels_p3.1k = read.delim('celltypes_1k_protein.tsv', row.names = 1)

pbmc_v2.1k <- AddMetaData(
    object = pbmc_v2.1k,
    metadata = labels_v2.1k)

pbmc_p3.1k <- AddMetaData(
    object = pbmc_p3.1k,
    metadata = labels_p3.1k)
```

Create a Seurat object with all datasets.

```
pbmc <- merge(pbmc_v2.1k, c(pbmc_v3.1k, pbmc_p3.1k),
add.cell.ids=c("v2.1k","v3.1k","p3.1k"))
pbmc$orig.ident[pbmc$orig.ident == 'PBMC'] <- 'v3.1k'
```

Let's first look at the datasets before applying any batch correction. We perform standard preprocessing (log-normalization), and identify variable features based on a variance stabilizing transformation ("vst"). Next, we scale the integrated data, run PCA, and visualize the results with UMAP. As you can see, the different batches do not overlap in the UMAP.

```
# Normalize and find variable features
pbmc <- NormalizeData(pbmc, verbose = FALSE)
pbmc <- FindVariableFeatures(pbmc, selection.method = "vst", nfeatures =
2000, verbose = FALSE)

# Run the standard workflow for visualization and clustering
pbmc <- ScaleData(pbmc, verbose = FALSE)
pbmc <- RunPCA(pbmc, npcs = 30, verbose = FALSE)
pbmc <- RunUMAP(pbmc, reduction = "pca", dims = 1:30, verbose = FALSE)

## Warning: The default method for RunUMAP has changed from calling Python
UMAP via reticulate to the R-native UWOT using the cosine metric
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and
metric to 'correlation'
## This message will be shown once per session

p1 <- DimPlot(pbmc, reduction = "umap", group.by = "orig.ident")
p2 <- DimPlot(pbmc, reduction = "umap", group.by = "celltype", label = TRUE,
repel = TRUE) +
    NoLegend()
p1 + p2
```



We split the combined object into a list, with each dataset as an element. We perform standard preprocessing (log-normalization), and identify variable features individually for each dataset based on a variance stabilizing transformation ("vst").

```
pbmc.list <- SplitObject(pbmc, split.by = "orig.ident")

for (i in 1:length(pbmc.list)) {
    pbmc.list[[i]] <- NormalizeData(pbmc.list[[i]], verbose = FALSE)
    pbmc.list[[i]] <- FindVariableFeatures(pbmc.list[[i]], selection.method =
"vst", nfeatures = 2000,
        verbose = FALSE)
}

### Select features that are repeatedly variable genes across the different
dataset

features <- SelectIntegrationFeatures(pbmc.list)
```

Integration of three PBMC datasets

We identify anchors using the `FindIntegrationAnchors` function, which takes a list of Seurat objects as input.

```
pbmc.anchors <- FindIntegrationAnchors(pbmc.list, dims = 1:30)

## Computing 2000 integration features

## Scaling features for provided objects

## Finding all pairwise anchors

## Running CCA

## Merging objects

## Finding neighborhoods
## Finding anchors
##  Found 3031 anchors
## Filtering anchors
##  Retained 2717 anchors
## Running CCA
## Merging objects
## Finding neighborhoods
## Finding anchors
##  Found 2540 anchors
## Filtering anchors
##  Retained 2143 anchors
## Running CCA
## Merging objects
## Finding neighborhoods
## Finding anchors
##  Found 2554 anchors
## Filtering anchors
##  Retained 2182 anchors
```

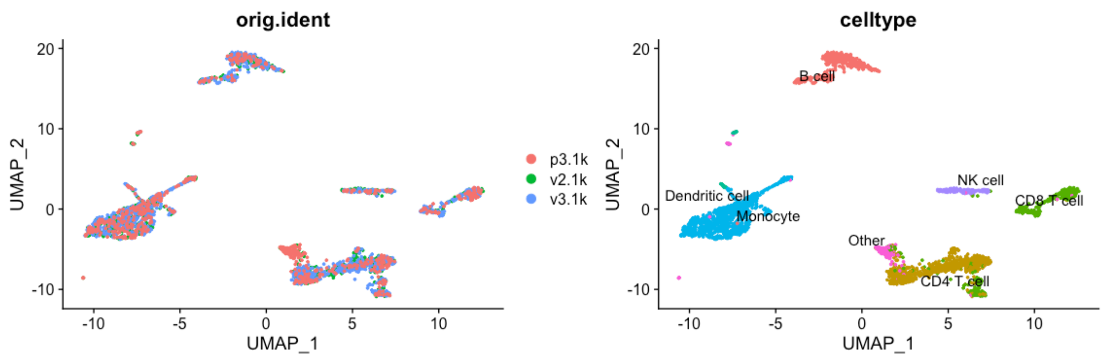
We then pass these anchors to the `IntegrateData` function, which returns a Seurat object.

```
pbmc.integrated <- IntegrateData(pbmc.anchors)
## Merging dataset 3 into 2
## Extracting anchors for merged samples
## Finding integration vectors
## Finding integration vector weights
## Integrating data
## Merging dataset 1 into 2 3
## Extracting anchors for merged samples
## Finding integration vectors
## Finding integration vector weights

## Integrating data
```

After running `IntegrateData`, the `Seurat` object will contain a new `Assay` with the integrated (or 'batch-corrected') expression matrix. Note that the original (uncorrected values) are still stored in the object in the "RNA" assay, so you can switch back and forth.

We can then use this new integrated matrix for downstream analysis and visualization. Here we scale the integrated data, run PCA, and visualize the results with UMAP. The integrated datasets cluster by cell type, instead of by technology.

```
# switch to integrated assay. The variable features of this assay are
automatically set during
# IntegrateData
DefaultAssay(pbmc.integrated) <- "integrated"

# Run the standard workflow for visualization and clustering
pbmc.integrated <- ScaleData(pbmc.integrated, verbose = FALSE)
pbmc.integrated <- RunPCA(pbmc.integrated, npcs = 30, verbose = FALSE)
pbmc.integrated <- RunUMAP(pbmc.integrated, reduction = "pca", dims = 1:30,
verbose = FALSE)
p1 <- DimPlot(pbmc.integrated, reduction = "umap", group.by = "orig.ident")
p2 <- DimPlot(pbmc.integrated, reduction = "umap", group.by = "celltype",
label = TRUE, repel = TRUE) +
    NoLegend()
p1+p2
```



Projecting labels from a reference atlas

For some well studied tissues, there exists already a reference atlas. Cell type labels from this reference atlas can then be easily propagated to your own new dataset. As discussed in the lecture, clustering can be quite subjective and time-consuming. With these automatic approaches, you can overcome these issues.

An example of an automatic method is Azimuth. When using Azimuth for small datasets, it is easiest to use the webportal. Here, you can choose which reference atlas you want to use, upload your own dataset, choose the normalization procedure, and transfer the labels. You can try this yourself with the three PBMC datasets. The best way to annotate them, is to upload the raw counts separately to the portal.

For larger datasets it is easier to download the reference atlas itself and annotate them. The code below shows how to do it. This code will take a long time or is impossible to run on a normal desktop or Rstudio cloud, but is very convenient for an HPC cluster.

First we download the reference dataset.

Next, we load the reference and align one of the pbmc datasets.

```
reference <- LoadH5Seurat("pbmc_multimodal.h5seurat")

pbmc_v3.1k <- SCTransform(pbmc_v3.1k, verbose=FALSE)

anchors <- FindTransferAnchors(reference=reference,
                               query = pbmc_v3.1k,
                               normalization.method='SCT',
                               reference.reduction='spca',
                               dims=1:50
                               )

pbmc_v3.1k <- MapQuery(
  anchorset = anchors,
  query = pbmc_v3.1k,
  reference = reference,
  refdata = list(
    celltype.l1 = "celltype.l1",
    celltype.l2 = "celltype.l2",
    predicted_ADT = "ADT"
  ),
  reference.reduction = "spca",
  reduction.model = "wnn.umap"
)

p1 = DimPlot(pbmc_v3.1k, reduction = "ref.umap", group.by =
"predicted.celltype.l1", label = TRUE, label.size = 3, repel = TRUE) +
NoLegend()
p2 = DimPlot(pbmc_v3.1k, reduction = "ref.umap", group.by =
"predicted.celltype.l2", label = TRUE, label.size = 3 ,repel = TRUE) +
NoLegend()
p1 + p2
```

Session info

```
sessionInfo()

## R version 4.1.0 (2021-05-18)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
```

```
##
## Matrix products: default
## BLAS:
/Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.dylib
## LAPACK:
/Library/Frameworks/R.framework/Versions/4.1/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] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] SeuratDisk_0.0.0.9019 SeuratObject_4.0.2    Seurat_4.0.4
##
## loaded via a namespace (and not attached):
##   [1] Rtsne_0.15            colorspace_2.0-2      deldir_0.2-10
##   [4] ellipsis_0.3.2        ggridges_0.5.3        rstudioapi_0.13
##   [7] spatstat.data_2.1-0   farver_2.1.0          leiden_0.3.9
##  [10] listenv_0.8.0         ggrepel_0.9.1         bit64_4.0.5
##  [13] RSpectra_0.16-0       fansi_0.5.0           codetools_0.2-18
##  [16] splines_4.1.0         knitr_1.34            polyclip_1.10-0
##  [19] jsonlite_1.7.2        ica_1.0-2             cluster_2.1.2
##  [22] png_0.1-7             uwot_0.1.10           shiny_1.6.0
##  [25] sctransform_0.3.2     spatstat.sparse_2.0-0 compiler_4.1.0
##  [28] httr_1.4.2            assertthat_0.2.1      Matrix_1.3-4
##  [31] fastmap_1.1.0         lazyeval_0.2.2        cli_3.0.1
##  [34] later_1.3.0           htmltools_0.5.2       tools_4.1.0
##  [37] igraph_1.2.6          gtable_0.3.0          glue_1.4.2
##  [40] RANN_2.6.1            reshape2_1.4.4        dplyr_1.0.7
##  [43] Rcpp_1.0.7            scattermore_0.7       vctrs_0.3.8
##  [46] nlme_3.1-153          lmtest_0.9-38         xfun_0.26
##  [49] stringr_1.4.0         globals_0.14.0        mime_0.11
##  [52] miniUI_0.1.1.1        lifecycle_1.0.0       irlba_2.3.3
##  [55] goftest_1.2-2         future_1.22.1         MASS_7.3-54
##  [58] zoo_1.8-9             scales_1.1.1          spatstat.core_2.3-0
##  [61] promises_1.2.0.1      spatstat.utils_2.2-0  parallel_4.1.0
##  [64] RColorBrewer_1.1-2    yaml_2.2.1            reticulate_1.22
##  [67] pbapply_1.5-0         gridExtra_2.3         ggplot2_3.3.5
##  [70] rpart_4.1-15          stringi_1.7.4         highr_0.9
##  [73] rlang_0.4.11          pkgconfig_2.0.3       matrixStats_0.60.1
##  [76] evaluate_0.14         lattice_0.20-44       ROCR_1.0-11
##  [79] purrr_0.3.4           tensor_1.5            labeling_0.4.2
##  [82] patchwork_1.1.1       htmlwidgets_1.5.4     cowplot_1.1.1
##  [85] bit_4.0.4             tidyselect_1.1.1      parallelly_1.28.1
##  [88] RcppAnnoy_0.0.19      plyr_1.8.6            magrittr_2.0.1
##  [91] R6_2.5.1              generics_0.1.0        DBI_1.1.1
##  [94] pillar_1.6.2          withr_2.4.2           mgcv_1.8-36
##  [97] fitdistrplus_1.1-5    survival_3.2-13       abind_1.4-5
## [100] tibble_3.1.4          future.apply_1.8.1    crayon_1.4.1
## [103] hdf5r_1.3.5           KernSmooth_2.23-20    utf8_1.2.2
## [106] spatstat.geom_2.2-2   plotly_4.9.4.1        rmarkdown_2.11
## [109] grid_4.1.0            data.table_1.14.0     digest_0.6.27
## [112] xtable_1.8-4          tidyr_1.1.3           httpuv_1.6.3
## [115] munsell_0.5.0         viridisLite_0.4.0
```