Single Cell RNA-sequencing Practical - Part B

Ahmed Mahfouz

10/14/2019

Overview

In this tutorial we will look at different approaches to clustering scRNA-seq datasets in order to characterize the different subgroups of cells. Using unsupervised clustering, we will try to identify groups of cells based on the similarities of the transcriptomes without any prior knowledge of the labels.

Load required packages:

```r
suppressMessages(require(tidyverse))
suppressMessages(require(Seurat))
suppressMessages(require(cowplot))
suppressMessages(require(scater))
suppressMessages(require(scran))
suppressMessages(require(igraph))
```

Datasets

In this tutorial, we will use a small dataset of cells from developing mouse embryo Deng et al. 2014. We have preprocessed the dataset and created a `SingleCellExperiment` object in advance. We have also annotated the cells with the cell types identified in the original publication (it is the `cell_type2` column in the `colData` slot).

```r
# load expression matrix
deng <- readRDS("deng-reads.rds")
deng
```

```
## class: SingleCellExperiment
## dim: 22431 268
## metadata(0):
## assays(2): counts logcounts
## rownames(22431): Hvcn1 Gbp7 ... Sox5 Alg11
## rowData names(10): feature_symbol is_feature_control ...
##   total_counts log10_total_counts
## colnames(268): 16cell 16cell.1 ... zy.2 zy.3
## colData names(30): cell_type2 cell_type1 ... pct_counts_ERCC
##   is_cell_control
## reducedDimNames(0):
## spikeNames(1): ERCC
```

```r
# look at the cell type annotation
table(colData(deng)$cell_type2)
```

```
##
##     16cell      4cell      8cell early2cell earlyblast  late2cell
##         50         14         37          8         43         10
##  lateblast   mid2cell   midblast         zy
##         30         12         60          4
```

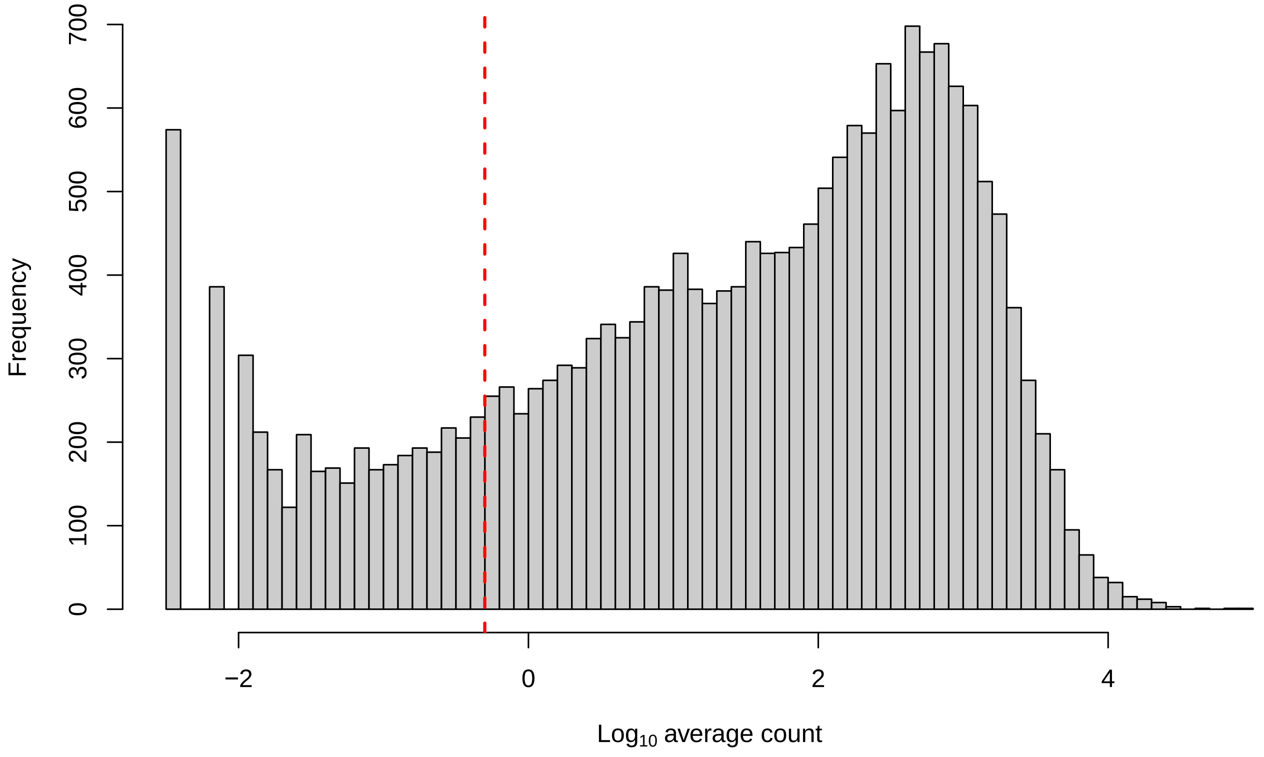
Feature selection

The first step is to decide which genes to use in clustering the cells. Single cell RNASeq can profile a huge number of genes in a lot of cells. But most of the genes are not expressed enough to provide a meaningful signal and are often driven by technical noise. Including them could potentially add some unwanted signal that would blur the biological variation. Moreover gene filtering can also speed up the computational time for downstream analysis.

Filtering out low abundance genes

Low-abundance genes are mostly non informative and are not representative of the biological variance of the data. They are often driven by technical noise such as dropout event. However, their presence in downstream analysis leads often to a lower accuracy since they can interfere with some statistical model that will be used and also will pointlessly increase the computational time which can be critical when working with very large data.

```
# Calculate gene mean across cell
gene_mean <- rowMeans(counts(deng))
# Calculate gene variance across cell
gene_var <- rowVars(counts(deng))
abundant_genes <- gene_mean >= 0.5  #Remove Low abundance genes
# plot low abundance gene filtering
hist(log10(gene_mean), breaks = 100, main = "", col = "grey80", xlab = expression(Log[10] ~
    "average count"))
abline(v = log10(0.5), col = "red", lwd = 2, lty = 2)
```



```
# remove low abundance gene in SingleCellExperiment Object
deng <- deng[abundant_genes, ]
dim(deng)
```

```
## [1] 17093   268
```

Filtering genes that are expressed in very few cells

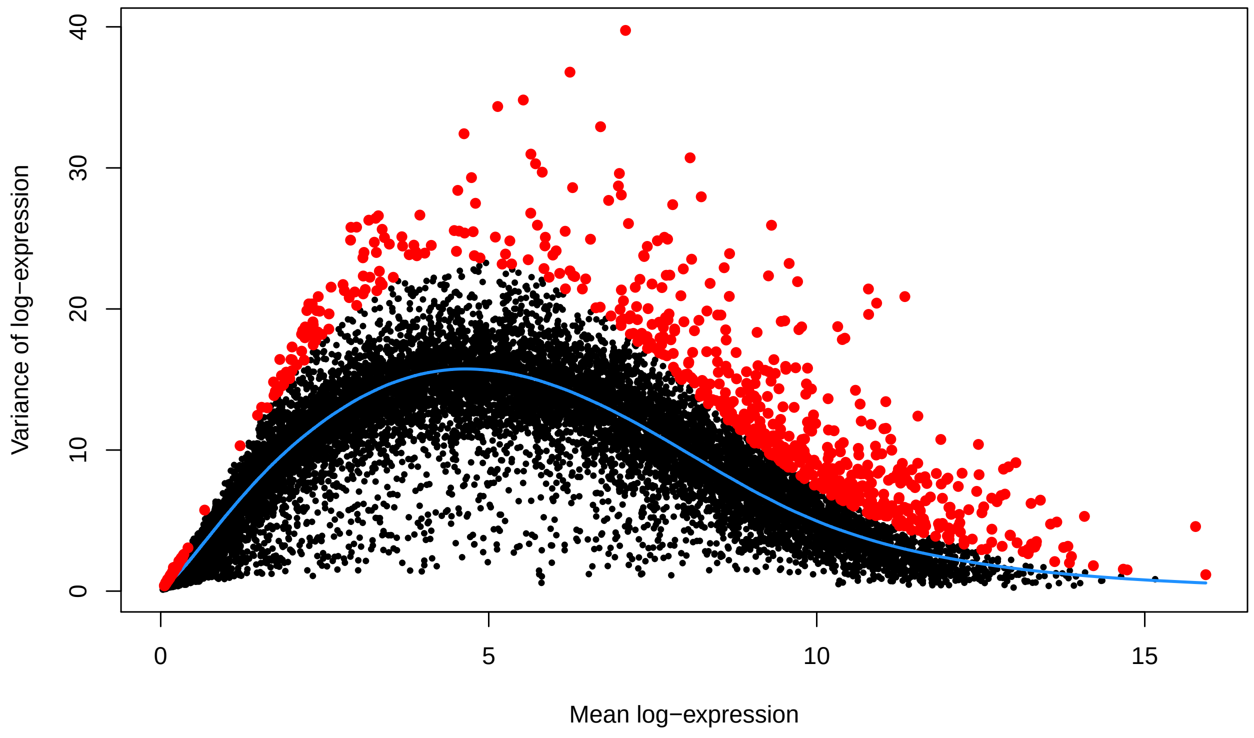
We can also filter some genes that are in a small number of cells. This procedure would remove some outlier genes that is highly expressed in one or two cells. These genes are unwanted for further analysis since they mostly comes from an iregular amplification of artifacts. It is important to note that we might not want to filter with this procedure when the aim of the analysis is to detect a very rare subpopulation in the data.

```r
# Calculate the number of non zero expression for each genes
numcells <- nexprs(deng, byrow = TRUE)

# Filter genes detected in less than 5 cells
numcells2 <- numcells >= 5
deng <- deng[numcells2, ]
dim(deng)
```

```
## [1] 16946   268
```

Detecting Highly Variable Genes

```r
fit <- trendVar(deng, parametric = TRUE, use.spikes = FALSE)
dec <- decomposeVar(deng, fit)
dec$HVG <- (dec$FDR < 1e-05)
hvg_genes <- rownames(dec[dec$FDR < 1e-05, ])

# plot highly variable genes
plot(dec$mean, dec$total, pch = 16, cex = 0.6, xlab = "Mean log-expression",
    ylab = "Variance of log-expression")
o <- order(dec$mean)
lines(dec$mean[o], dec$tech[o], col = "dodgerblue", lwd = 2)
points(dec$mean[dec$HVG], dec$total[dec$HVG], col = "red", pch = 16)
```

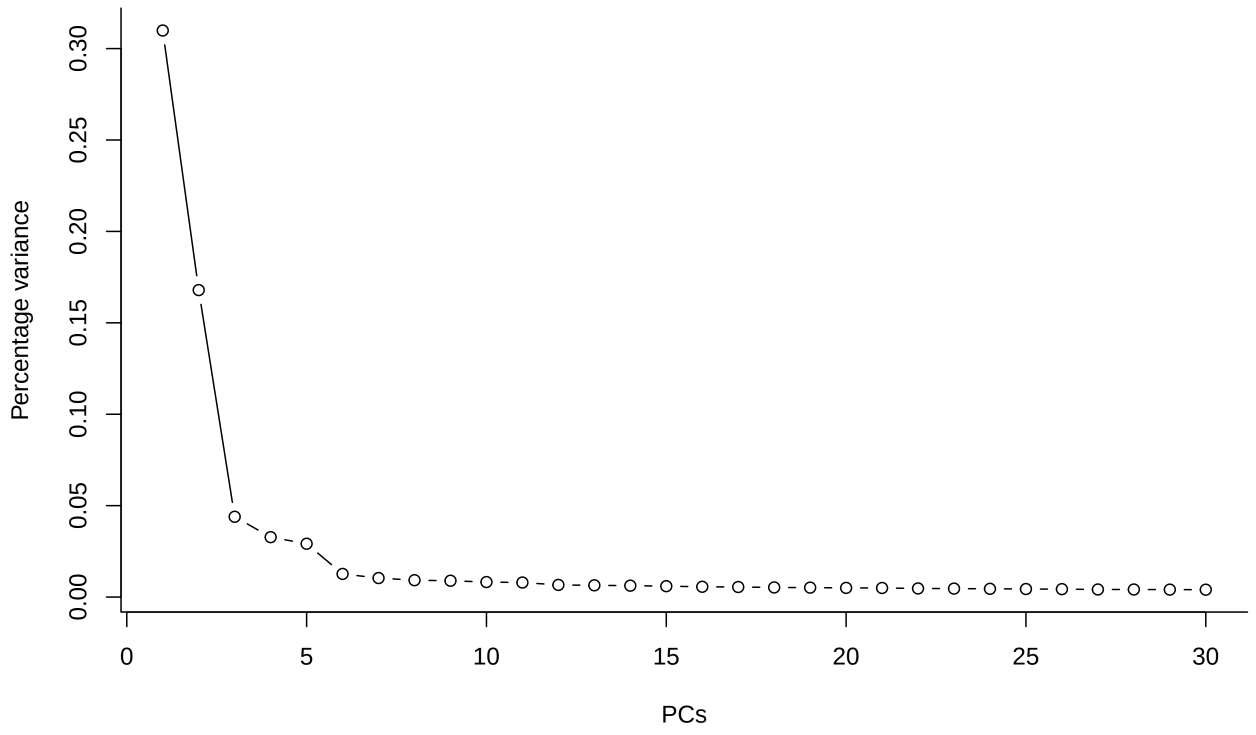


```
## save the decomposed variance table and hvg_genes into metadata for
## safekeeping
metadata(deng)$hvg_genes <- hvg_genes
metadata(deng)$dec_var <- dec
```

Dimensionality reduction

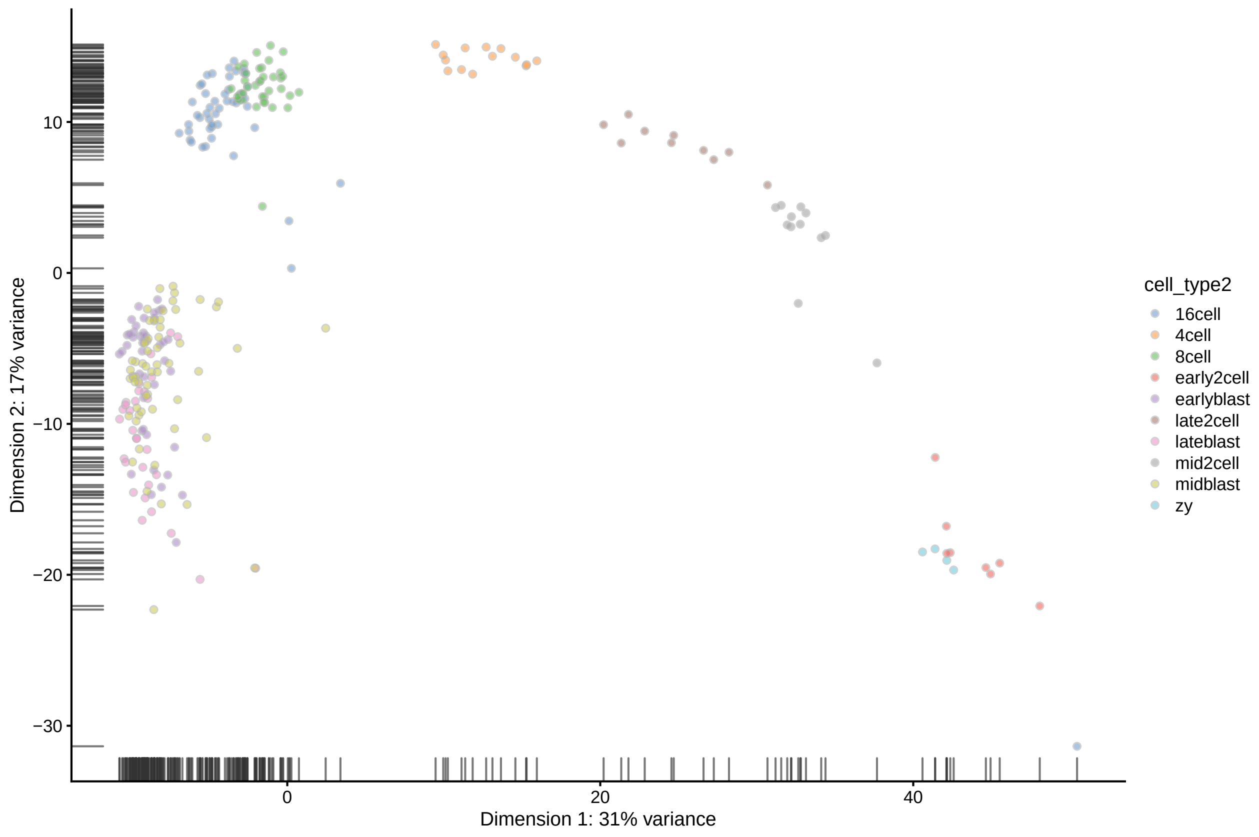
The clustering problem is computationally difficult due to the high level of noise (both technical and biological) and the large number of dimensions (i.e. genes). We can solve these problems by applying dimensionality reduction methods (e.g. PCA, tSNE, and UMAP)

```
# PCA (select the number of components to calculate)
deng <- runPCA(deng, method = "irlba", ncomponents = 30, feature_set = metadata(deng)$hvg_genes)

# Make a scree plot (percentage variance explained per PC) to determine the
# number of relevant components
X <- attributes(deng@reducedDims$PCA)
plot(X$percentVar ~ c(1:30), type = "b", lwd = 1, ylab = "Percentage variance",
    xlab = "PCs", bty = "l", pch = 1)
```



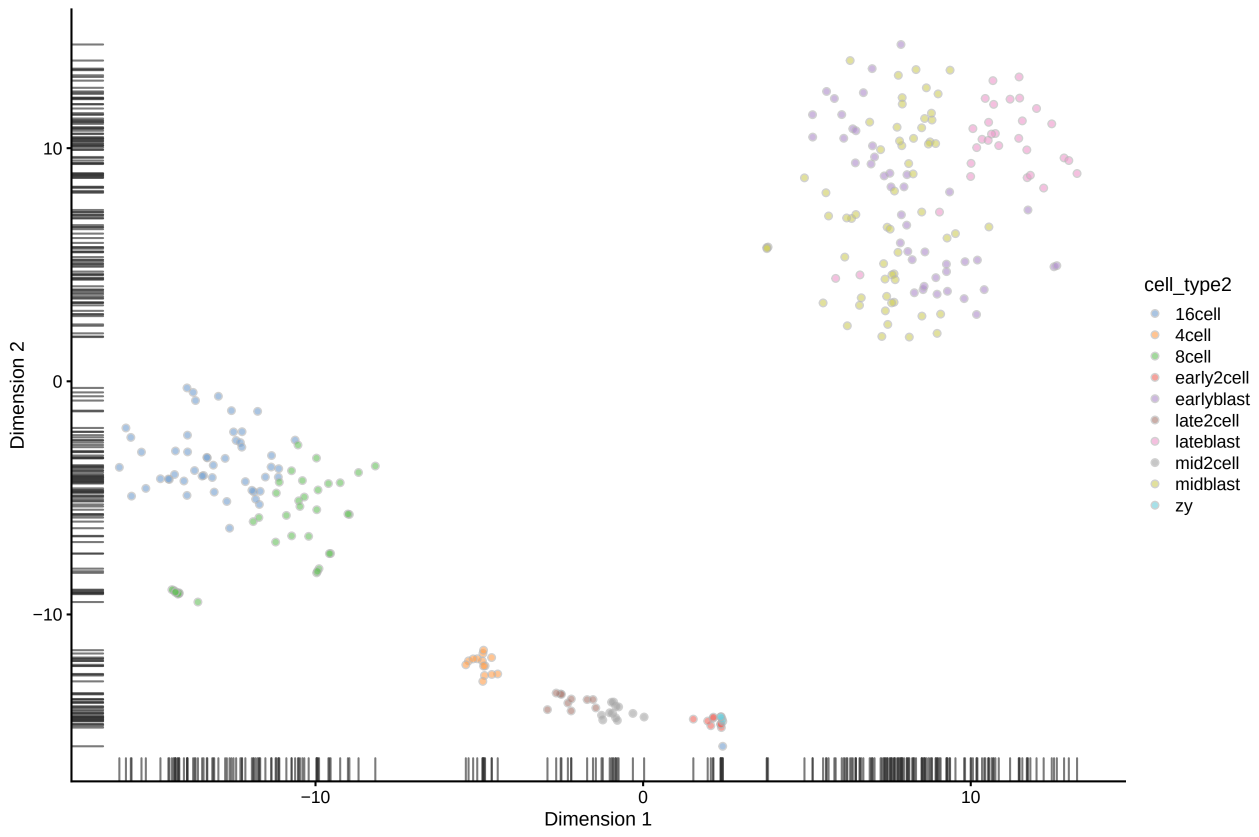
Make a PCA plot (PC1 vs. PC2)

```
plotReducedDim(deng, "PCA", colour_by = "cell_type2")
```



Make a tSNE plot *Note:* tSNE is a stochastic method. Everytime you run it you will get slightly different results. For convinience we can get the same results if we seet the seed.

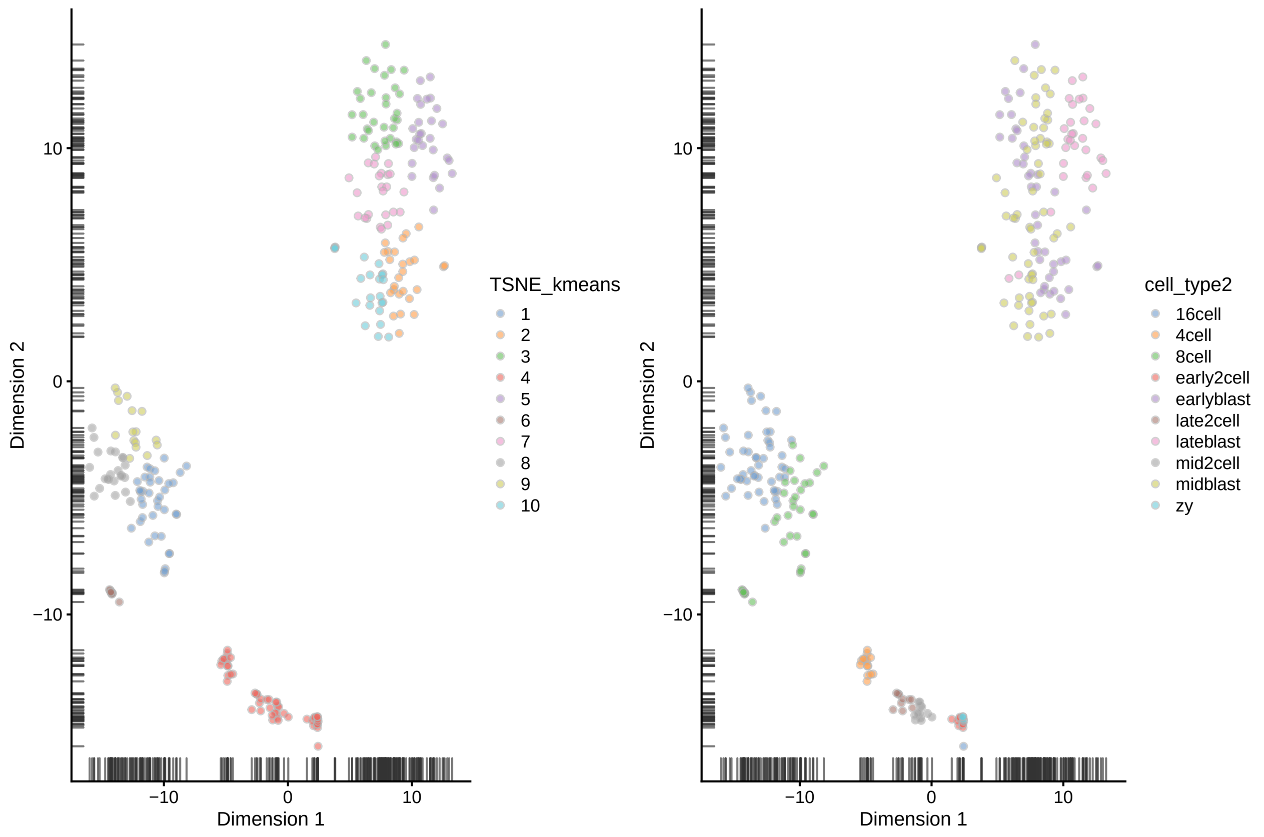
```
# tSNE
deng <- runTSNE(deng, perplexity = 30, feature_set = metadata(deng)$hvg_genes,
    set.seed = 1)


plotReducedDim(deng, "TSNE", colour_by = "cell_type2")
```



Clustering

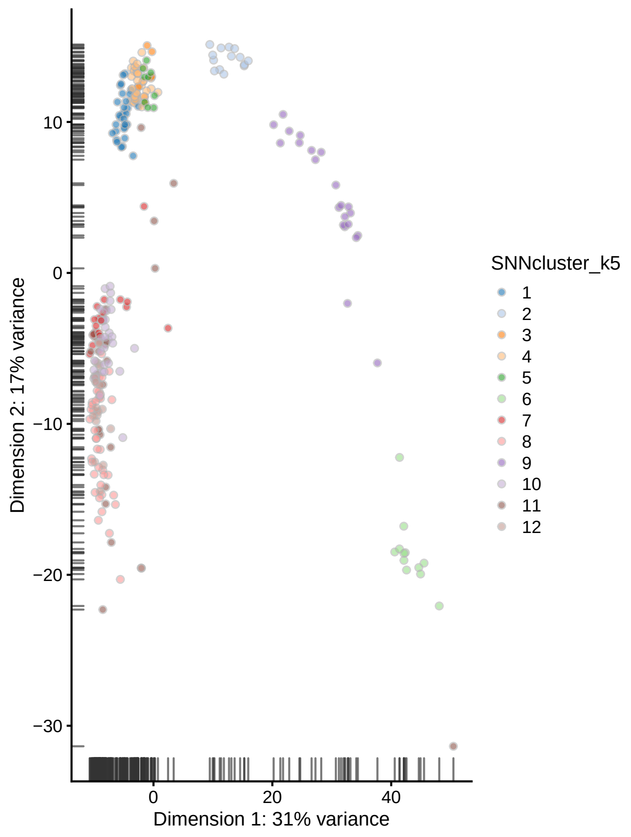
TSNE + Kmeans

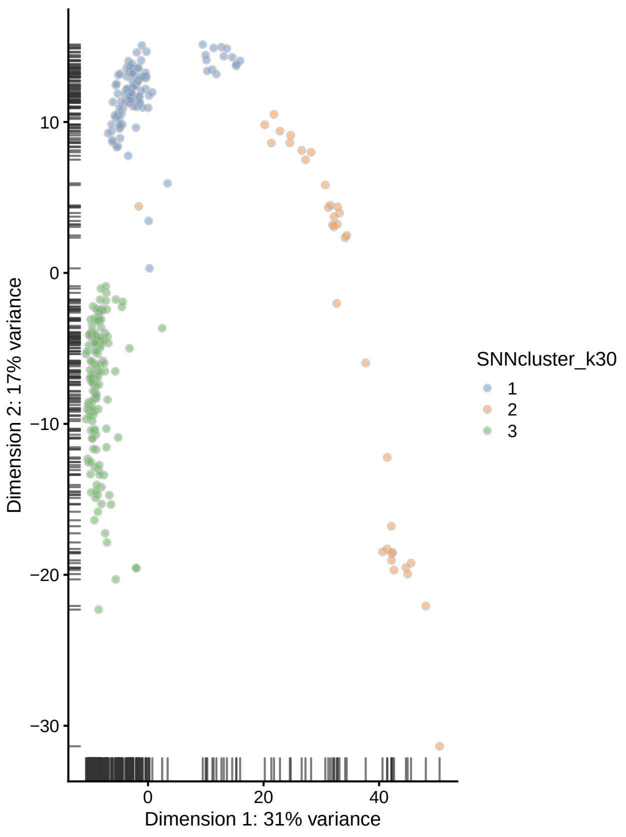
```
# Do kmeans algorithm on TSNE coordinates
deng_kmeans <- kmeans(x = deng@reducedDims$TSNE, centers = 10)
TSNE_kmeans <- factor(deng_kmeans$cluster)
colData(deng)$TSNE_kmeans <- TSNE_kmeans
# Compare with ground truth
plot_grid(plotTSNE(deng, colour_by = "TSNE_kmeans"), plotTSNE(deng, colour_by = "cell_type2"))
```



Graph Based Clustering

```
# k=5 The k parameter denfines the number of closest cells to look for each
# cells
SNNgraph_k5 <- buildSNNGraph(x = deng, k = 5)
SNNcluster_k5 <- cluster_louvain(SNNgraph_k5)
colData(deng)$SNNcluster_k5 <- factor(SNNcluster_k5$membership)
p5 <- plotPCA(deng, colour_by = "SNNcluster_k5") + guides(fill = guide_legend(ncol = 2))

# k30
SNNgraph_k30 <- buildSNNGraph(x = deng, k = 30)
SNNcluster_k30 <- cluster_louvain(SNNgraph_k30)
colData(deng)$SNNcluster_k30 <- factor(SNNcluster_k30$membership)
p30 <- plotPCA(deng, colour_by = "SNNcluster_k30")

# plot the different clustering.
plot_grid(p5 + guides(fill = guide_legend(ncol = 1)), p30)
```





Session info

```r
sessionInfo()
```

```
## R version 3.5.0 (2018-04-23)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS  10.14.2
##
## 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_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] igraph_1.2.4.1              scran_1.10.2
##  [3] scater_1.10.1               SingleCellExperiment_1.4.1
##  [5] SummarizedExperiment_1.12.0 DelayedArray_0.8.0
##  [7] BiocParallel_1.16.6         matrixStats_0.55.0
##  [9] Biobase_2.42.0              GenomicRanges_1.34.0
## [11] GenomeInfoDb_1.18.2         IRanges_2.16.0
## [13] S4Vectors_0.20.1            BiocGenerics_0.28.0
## [15] cowplot_1.0.0               Seurat_3.1.0
## [17] forcats_0.4.0               stringr_1.4.0
## [19] dplyr_0.8.3                 purrr_0.3.2
## [21] readr_1.3.1                 tidyr_1.0.0
## [23] tibble_2.1.3                ggplot2_3.2.1
## [25] tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##   [1] readxl_1.3.1             backports_1.1.4
##   [3] plyr_1.8.4               lazyeval_0.2.2
##   [5] splines_3.5.0            listenv_0.7.0
##   [7] digest_0.6.21            htmltools_0.3.6
##   [9] viridis_0.5.1            gdata_2.18.0
##  [11] magrittr_1.5             cluster_2.1.0
##  [13] ROCR_1.0-7               limma_3.38.3
##  [15] globals_0.12.4           modelr_0.1.5
##  [17] RcppParallel_4.4.4       R.utils_2.9.0
##  [19] colorspace_1.4-1         rvest_0.3.4
##  [21] ggrepel_0.8.1            haven_2.1.1
##  [23] xfun_0.9                 crayon_1.3.4
##  [25] RCurl_1.95-4.12          jsonlite_1.6
##  [27] zeallot_0.1.0            survival_2.44-1.1
##  [29] zoo_1.8-6                ape_5.3
##  [31] glue_1.3.1               gtable_0.3.0
##  [33] zlibbioc_1.28.0          XVector_0.22.0
##  [35] leiden_0.3.1             Rhdf5lib_1.4.3
##  [37] future.apply_1.3.0       HDF5Array_1.10.1
##  [39] scales_1.0.0             edgeR_3.24.3
##  [41] bibtex_0.4.2             Rcpp_1.0.2
##  [43] metap_1.1                viridisLite_0.3.0
##  [45] reticulate_1.13          rsvd_1.0.2
##  [47] SDMTools_1.1-221.1       tsne_0.1-3
##  [49] htmlwidgets_1.3          httr_1.4.1
##  [51] gplots_3.0.1.1           RColorBrewer_1.1-2
##  [53] ica_1.0-2                pkgconfig_2.0.3
##  [55] R.methodsS3_1.7.1        uwot_0.1.4
##  [57] locfit_1.5-9.1           dynamicTreeCut_1.63-1
##  [59] labeling_0.3             tidyselect_0.2.5
##  [61] rlang_0.4.0              reshape2_1.4.3
##  [63] munsell_0.5.0            cellranger_1.1.0
##  [65] tools_3.5.0              cli_1.1.0
##  [67] generics_0.0.2           broom_0.5.2
##  [69] ggridges_0.5.1           evaluate_0.14
##  [71] yaml_2.2.0               npsurv_0.4-0
##  [73] knitr_1.25               fitdistrplus_1.0-14
##  [75] caTools_1.17.1.2         RANN_2.6.1
##  [77] pbapply_1.4-2            future_1.14.0
##  [79] nlme_3.1-141             formatR_1.7
##  [81] R.oo_1.22.0              xml2_1.2.2
##  [83] compiler_3.5.0           rstudioapi_0.10
##  [85] beeswarm_0.2.3           plotly_4.9.0
##  [87] png_0.1-7                lsei_1.2-0
##  [89] statmod_1.4.32           stringi_1.4.3
##  [91] lattice_0.20-38          Matrix_1.2-17
##  [93] vctrs_0.2.0              pillar_1.4.2
##  [95] lifecycle_0.1.0          Rdpack_0.11-0
##  [97] lmtest_0.9-37            BiocNeighbors_1.0.0
##  [99] RcppAnnoy_0.0.13         data.table_1.12.2
## [101] bitops_1.0-6             irlba_2.3.3
## [103] gbRd_0.4-11              R6_2.4.0
## [105] KernSmooth_2.23-15       gridExtra_2.3
## [107] vipor_0.4.5              codetools_0.2-16
## [109] MASS_7.3-51.4            gtools_3.8.1
## [111] assertthat_0.2.1         rhdf5_2.26.2
## [113] withr_2.1.2              sctransform_0.2.0
## [115] GenomeInfoDbData_1.2.0   hms_0.5.1
## [117] grid_3.5.0               rmarkdown_1.15
## [119] DelayedMatrixStats_1.4.0 Rtsne_0.15
## [121] lubridate_1.7.4          ggbeeswarm_0.6.0
```