Cell Ranger™ R Kit Tutorial: Secondary Analysis on 10x Genomics™ Single Cell 3' RNA-seq PBMC Data

10x Genomics™

July 18, 2017

Contents

1 Introduction

1.1 Overview of this tutorial

This tutorial provides instructions on how to perform exploratory secondary analysis on single cell 3' RNA-seq data produced by the 10x Genomics™ Chromium™ Platform, and processed by the Cell Ranger™ pipeline. We illustrate an example workflow using peripheral blood mononuclear cells (PBMCs) from a healthy donor, using two data sets: `pbmc3k` and `pbmc6k` (from the same donor) released by 10x Genomics™ . More information on these datasets can be found here:

http://support.10xgenomics.com/single-cell/datasets

In this tutorial, you will learn how to:

- Load processed data from the Cell Ranger™ pipeline or public data provided by 10x Genomics™
- Perform unsupervised clustering for *de novo* cell-type discovery
- Visualize clustering of single cells and cluster-specific gene expression signatures
- Combine data from multiple experiments

1.2 Additional resources

You can learn more about the Cell Ranger™ pipeline here:

`http://support.10xgenomics.com/single-cell/software/pipelines/latest/what-is-cell-ranger`

A larger number of other publicly available single cell data sets released by 10x Genomics™ are available here:

`http://support.10xgenomics.com/single-cell/datasets`

If you have any questions about the data and the analysis, please contact `support@10xgenomics.com`.

2 Getting Started

2.1 Installing Cell Ranger R Kit

You can follow the instructions at the link below to install R Kit and its dependencies:

`http://support.10xgenomics.com/single-cell/software/pipelines/latest/rkit`

2.2 Loading gene expression data for secondary analysis

The Chromium™ Cell Ranger™ pipeline produces processed gene-barcode matrices and primary results. By specifiying the pipeline output directory (or *pipestance path*), you can use the R kit to load these pipeline results into your local R environment. You can load the pipeline data by specifying a pipestance path in R as follows.

```
cellranger_pipestance_path <- "/path/to/your/pipeline/output/directory"
gbm <- load_cellranger_matrix(cellranger_pipestance_path)
analysis_results <- load_cellranger_analysis_results(cellranger_pipestance_path)
```

Alternatively, you can download the publicly available data to a local path using the `download_sample` function and treat this local path as your pipestance path.

```
pipestance_path <- "/path/to/your/local/directory/for/pbmc3k"
download_sample(sample_name="pbmc3k",sample_dir=pipestance_path,
                host="https://s3-us-west-2.amazonaws.com/10x.files/samples/cell/")
gbm <- load_cellranger_matrix(pipestance_path)
analysis_results <- load_cellranger_analysis_results(pipestance_path)
```

The variable `gbm` is an object based on the Bioconductor ExpressionSet class that stores the barcode filtered gene expression matrix and metadata, such as gene symbols and barcode IDs corresponding to cells in the data set. The gene expression values, gene information and cell barcodes can be accessed using:

```
exprs(gbm) # expression matrix
fData(gbm) # data frame of genes
pData(gbm) # data frame of cell barcodes
```

The variable `analysis_results` contains pre-computed results for principle component analysis (PCA) dimensional reduction, t-SNE (t-Distributed Stochastic Neighbor Embedding) projection, and k-means clustering. Because each cell is represented by a high-dimensional gene expression profile, one way to visualize cell-to-cell similarity is by reducing the data set to a two dimensional representation. There are many dimension reduction methods available for 2-D visualization. Here we include implementations of two popular approaches, PCA and t-SNE, with pre-computed results stored in `analysis_results`.

For instance, you can access the t-SNE projection and plot the cells colored by UMI counts as follows.

```
tsne_proj <- analysis_results$tsne
visualize_umi_counts(gbm,tsne_proj[c("TSNE.1","TSNE.2")],limits=c(3,4),marker_size=0.05)
```

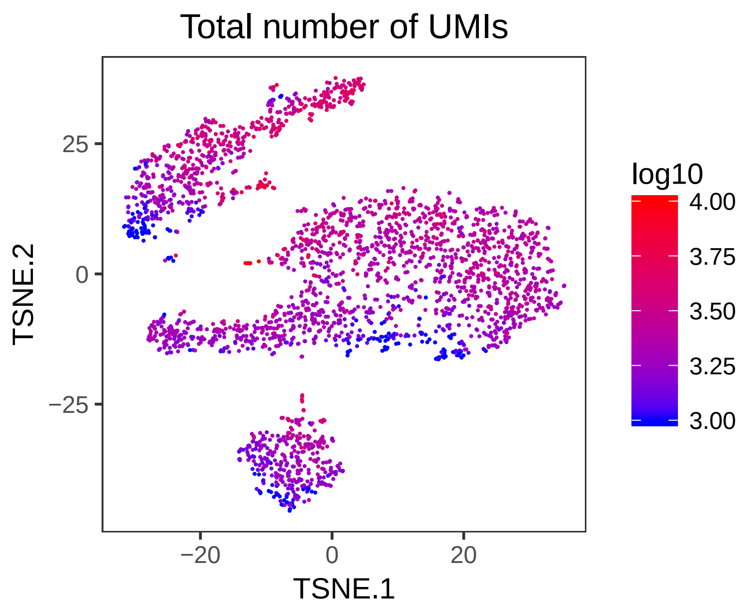

Figure 1: t-SNE projection where each cell is colored by log10 of UMI counts. Color scale represents log10 of UMI counts.

Each point in the scatter plot represents a cell in the coordinates specified by the two t-SNE components. The color of each point plotted by `visualize_umi_counts` (Figure 1) indicates the total number of UMIs for each cell, and these count values are displayed in log10 scale.

Cells with similar expression profile tend to appear closer in the 2-D space, so you may already see some structures in the data. However, to identify what the structures represent, you will need to rely on the gene signatures that each cell expresses to draw meaningful insights from the data. Instead of using raw UMI counts for downstream differential gene analysis, we recommend that you filter unexpressed genes, normalize the UMI counts for each barcode, and use the log-transformed gene-barcode matrix. After transformation, the gene-barcode matrix contains 16634 genes for 2700 cells.

```
use_genes <- get_nonzero_genes(gbm)
gbm_bcnorm <- normalize_barcode_sums_to_median(gbm[use_genes,])
gbm_log <- log_gene_bc_matrix(gbm_bcnorm,base=10)
print(dim(gbm_log))

## [1] 16634  2700
```

3 Visualizing signatures of known gene markers

One way to identify different cell types is by expression of specific genes in cells. You can visualize expression values of a set of genes to identify cells where particular genes are up-regulated. Here, we show how to simultaneously plot the expression of 6 gene markers, one for each subplot using the `visualize_gene_markers` function.

```
genes <- c("CD79A","NKG7","CD3D","CST3","CD8A","PF4")
tsne_proj <- analysis_results$tsne
visualize_gene_markers(gbm_log,genes,tsne_proj[c("TSNE.1","TSNE.2")],limits=c(0,1.5))
```

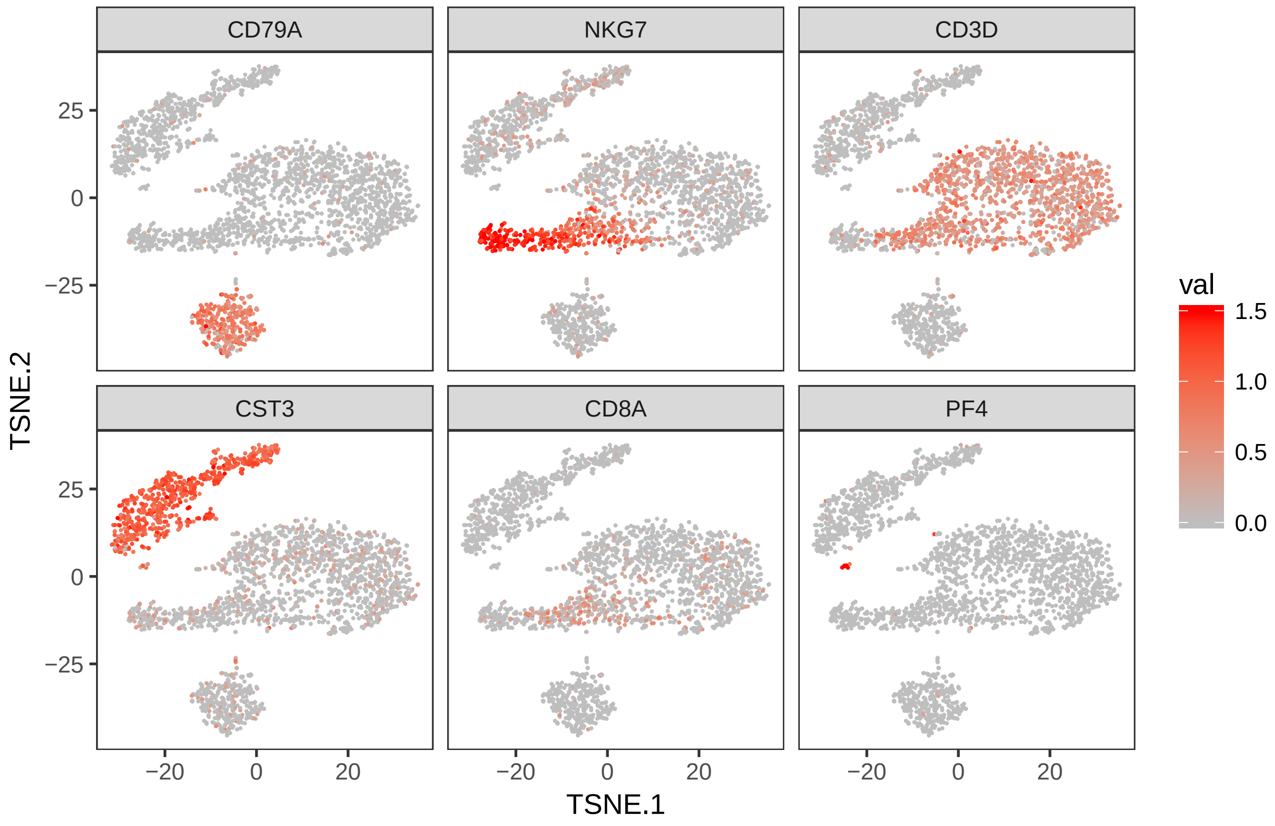

Figure 2: t-SNE projection where each cell is colored by normalized expression of the marker in the cell. Color scale represents the normalized expression of the marker.

Because the input matrix `gbm_log` is already normalized and log10-transformed, the color indicates the UMI counts for each gene under log10 scale (Figure 2). The specific gene markers already reveal cell types such as B cells, T cells, and natural killer (NK) cells. As expected, the cell-type-specific markers we have chosen are mostly unique to cells that are close together in the 2-D t-SNE projection.

4 Unbiased analysis using clustering results

4.1 Visualizing clustering results

For some data sets, *de novo* cell-type discovery may be desired as prior knowledge of relevant marker genes can be limited or one may not want to introduce biases. In these cases it is useful to use unsupervised analysis to first algorithmically specify groupings of cells using the full data set. This can be achieved by applying any clustering methods on the loaded gene expression matrix `exprs(gbm)` or the matrix in a reduced dimension space. In our implementation, for instance, we apply k-means clustering on the top 10 principle components of the gene expression matrix (after log-transformation, centering and scaling).

Because k-means clustering requires a specified number of clusters in the data set but the number of clusters in a data set may not be known *a priori*, it is helpful to consider multiple values of k. The output data from Cell Ranger™ includes the pre-computed cluster labels sweeping k from 2 to 10. So you can quickly visualize results for different values of k and pick the one that agrees with your intuition (Figure 3). Here each cell is colored by its cluster ID.

```
n_clu <- 2:10
km_res <- analysis_results$clustering # load pre-computed kmeans results
clu_res <- sapply(n_clu, function(x) km_res[[paste("kmeans",x,"clusters",sep="_")]]$Cluster)
colnames(clu_res) <- sapply(n_clu, function(x) paste("kmeans",x,sep="."))
visualize_clusters(clu_res,tsne_proj[c("TSNE.1","TSNE.2")])
```

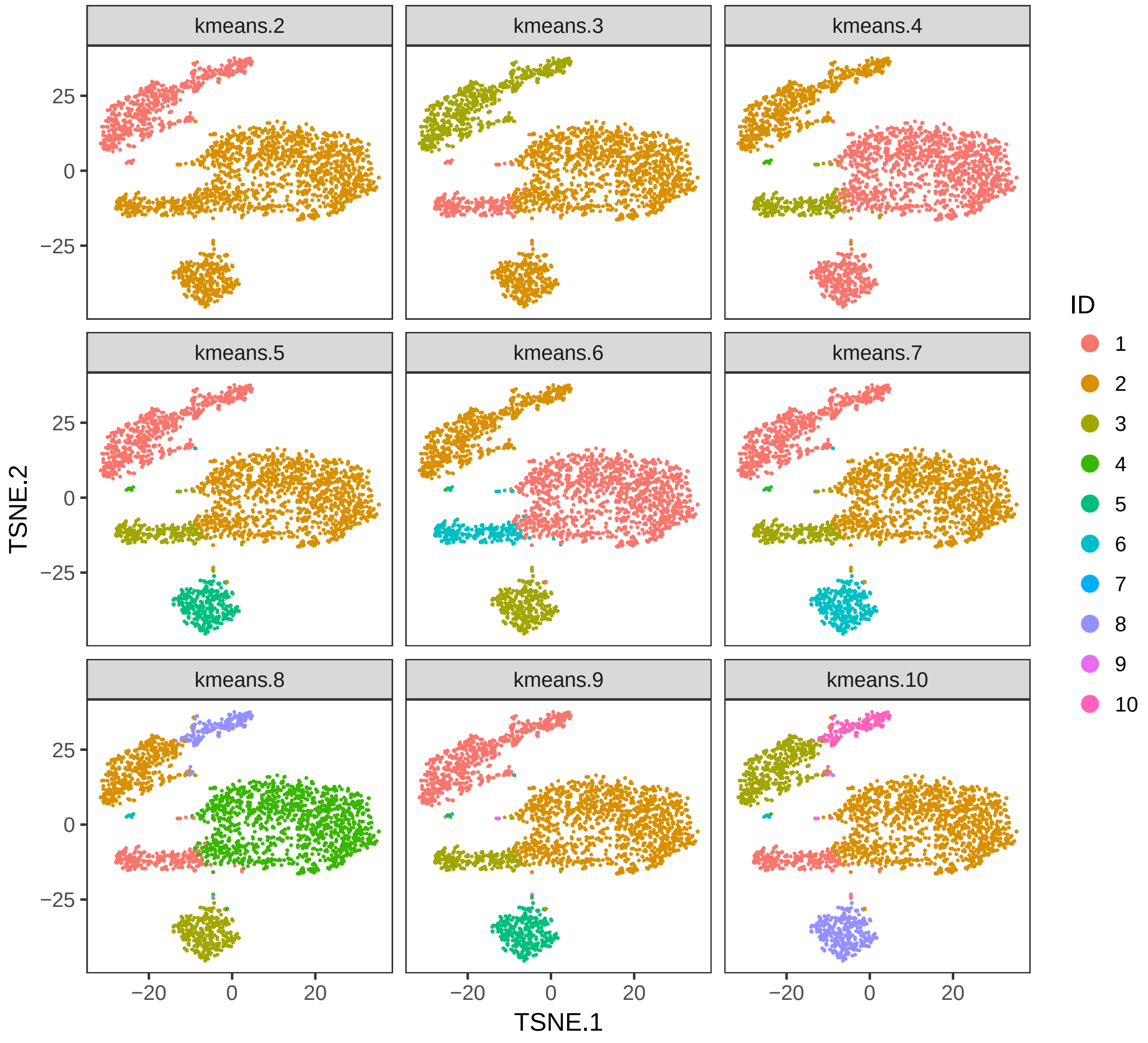

Figure 3: t-SNE projection where each cell is colored by cluster assigned by kmeans clustering. ID on the right represents the cluster ID.

4.2 Analyzing cluster specific genes

Using these integer cluster labels (or integer labels generated by any clustering algorithm), you can now perform differential gene analysis to identify gene markers that are specific to a particular cell population. For this example, we focus on the k-means clustering result above with 5 clusters (Figure 4). Standard statistical tests can be applied to identify which genes are most differentially expressed across different cell types.

```
example_K <- 5    # number of clusters (use "Set3" for brewer.pal below if example_K > 8)
example_col <- rev(brewer.pal(example_K,"Set2")) # customize plotting colors
cluster_result <- analysis_results$clustering[[paste("kmeans", example_K,"clusters",sep="_")]]
visualize_clusters(cluster_result$Cluster,tsne_proj[c("TSNE.1","TSNE.2")],colour=example_col)
```

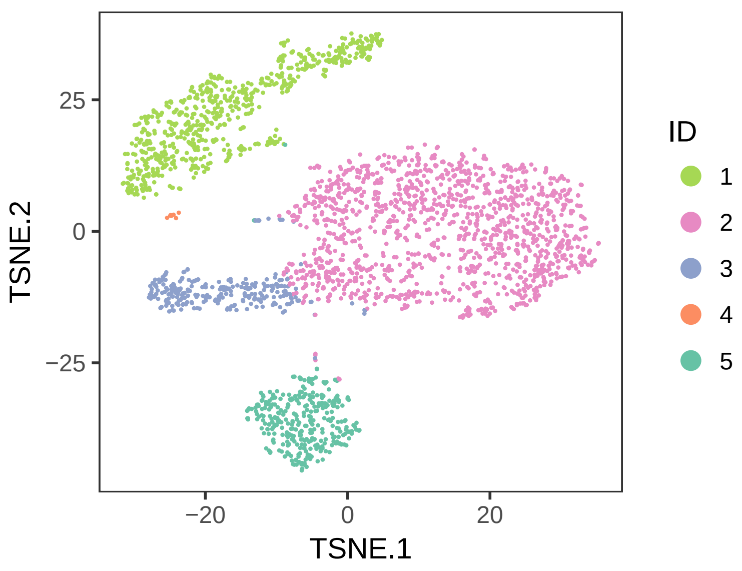

Figure 4: t-SNE projection where each cell is colored by cluster assigned by kmeans clustering. ID on the right represents the cluster ID.

You can compare the mean expression between a class of cells and the remaining ones, and then prioritize genes by how highly expressed they are in the class of interest. The function `prioritize_top_genes` identifies markers that are up-regulated in particular clusters of cells. Note that there are many other ways to identify cluster-specific genes. The approach used here is quick when looking for cluster-specific genes among ~ 20k candidate genes.

```
# sort the cells by the cluster labels
cells_to_plot <- order_cell_by_clusters(gbm, cluster_result$Cluster)
# order the genes from most up-regulated to most down-regulated in each cluster
prioritized_genes <- prioritize_top_genes(gbm, cluster_result$Cluster, "sseq", min_mean=0.5)

## Computing differential expression parameters...
## Comparing Cluster 1 against the other clusters...
## Comparing Cluster 2 against the other clusters...
## Comparing Cluster 3 against the other clusters...
## Comparing Cluster 4 against the other clusters...
## Comparing Cluster 5 against the other clusters...
```

Now that the genes for each cell type are prioritized, you can output the top genes specific to each cluster to a local folder. In this case, we output all the top 10 gene symbols for the 5 clusters to file.

```
output_folder <-"/path_to_your_local_folder/pbmc_data_public/pbmc3k/gene_sets"
write_cluster_specific_genes(prioritized_genes, output_folder, n_genes=10)
```

In addition, you can use the prioritized genes to plot a heat-map where the top three most up-regulated genes in each cluster are displayed. In addition, you can order the cells according to the cell IDs produced by k-means clustering to make the cell population signatures more apparent (Figure 5).

```
# create values and axis annotations for pheatmap
gbm_pheatmap(log_gene_bc_matrix(gbm), prioritized_genes, cells_to_plot,
             n_genes=3, colour=example_col, limits=c(-1,2))
```

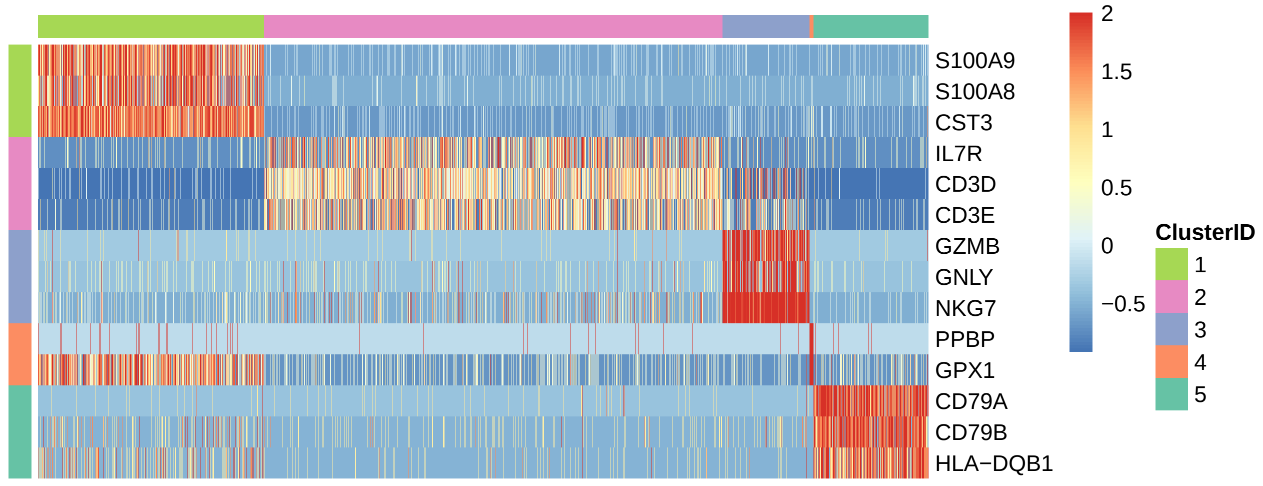

Figure 5: Heatmap of scaled expression of top 3 genes (row) in each cell (column). The horizontal and vertical bars around the heat map represents cluster ID assigned by kmeans clustering. Color scale represents scaled expression of the gene.

Notice that the top three genes for each cluster already give us an idea on the identity of each cluster. For instance,

- S100A9 is among the top 3 most expressed genes in Cluster 1, suggesting a population of monocytes;
- CD3D is among the top 3 most expressed genes in Cluster 2, suggesting a population of T cells;
- NKG7 is among the top 3 most expressed genes in Cluster 3, suggesting a population of NK cells;
- PPBP is among the top 3 most expressed genes in Cluster 4, suggesting a population of megakaryocytes;
- CD79A is among the top 3 most expressed genes in Cluster 5, suggesting a population of B cells.

You can also find out the size of each cluster. For example, about 13% cells were classified as B cells in this sample.

```
cell_composition(cluster_result$Cluster,
                 anno=c("monocytes","T cells","NK cells","megakaryocytes","B cells"))

## Cell composition:
##            1           2         3          4                5
## annotation "monocytes" "T cells" "NK cells" "megakaryocytes" "B cells"
## num_cells  "685"       "1391"    "264"      "12"             "348"
## proportion "0.2537"    "0.51519" "0.09778"  "0.00444"        "0.12889"
```

5 Analyzing multiple data sets

5.1 Merging multiple pbmc data sets

Note: the following section is deprecated. As of cellranger 1.2.0, you can use the `cellranger aggr` pipeline, which combines matrices and performs read-depth normalization for you. The `equalize_gbms` and `load_molecule_info` functions are no longer officially supported.

It is very common to pool data from different lanes of a 10x run or even different samples when analyzing single cell data sets. Because the output of these experiments will involve multiple Cell Ranger runs (one per Cell Ranger run), the pre-computed information such as t-SNE projections and k-means clustering need to be re-computed after pooling different data sets. Here we demonstrate how to merge two PBMC data sets where 2 libraries (pbmc3k and pbmc6k) were made and sequenced on different sequencers. You can download the data and store them in the format as you did for the single PBMC example and specify the path for each sample.

```
cellranger_pipestance_path_1 <- "/path/to/your/pipeline/output/directory1"
cellranger_pipestance_path_2 <- "/path/to/your/pipeline/output/directory2"
gbm1 <- load_cellranger_matrix(cellranger_pipestance_path_1)
gbm2 <- load_cellranger_matrix(cellranger_pipestance_path_2)
```

Alternatively, you can download the two datasets using the `download_sample` function. Notice that you will need to set the option `lite=FALSE` to download necessary molecule information in order to merge multiple data sets. The additional molecule information file can be a few hundred megabytes, and hence downloading the sample will take longer.

```
host <- "https://s3-us-west-2.amazonaws.com/10x.files/samples/cell/"
pipestance_path_1 <- "/path/to/your/local/directory/for/pbmc6k"
pipestance_path_2 <- "/path/to/your/local/directory/for/pbmc3k"
download_sample("pbmc6k",pipestance_path_1,host,lite=FALSE)
download_sample("pbmc3k",pipestance_path_2,host,lite=FALSE)
gbm1 <- load_cellranger_matrix(pipestance_path_1)
gbm2 <- load_cellranger_matrix(pipestance_path_2)
```

To merge multiple data sets, you can simply merge multiple `gbm` objects and re-normalize using the function: `concatenate_gene_bc_matrices`. However, we recommend equalizing read depths between samples before merging to reduce the batch effect that might have been introduced through sequencing. You can use `equalize_gbm` to sub-sample a list of `gbm` objects to generate new data sets with depth equal to that of the sample with the least reads. You can also apply any other batch effect removal strategy on this list of `gbm` objects to account for the differences across gene expression matrices in your study design. The cell pipestance indices are stored as batch labels using `merged_ID`.

```
set.seed(0)
gbm_list <- list(gbm1, gbm2)
gbm_list <- lapply(gbm_list,load_molecule_info) # load sample molecule information
gbm_list_equalized <- equalize_gbms(gbm_list)   # equalize the gene-barcode matrices
merged_gbm <- concatenate_gene_bc_matrices(gbm_list_equalized)
merged_ID <- unlist(lapply(1:length(gbm_list), function(x) rep(x,dim(gbm_list[[x]])[2])))
```

5.2 Re-running PCA, t-SNE and k-means clustering using R

Note: the following section is deprecated. As of cellranger 1.3.0, you can save your (modified) matrix to a file and use the `cellranger reanalyze` pipeline, which re-runs the secondary analysis for you.

When two different samples are merged, you need to recompute the the PCA dimension reduction, t-SNE projection, and k-means clustering (or other methods of your choice).

```
set.seed(0)
n_clust <- 5
pca_result <- run_pca(merged_gbm)
tsne_result <- run_tsne(pca_result)
kmeans_result <- run_kmeans_clustering(pca_result,k=n_clust)
# included re-named cell IDs, t-SNE and k-means result in a merged data frame
merged_tsne_clust <- data.frame(Barcode=as.factor(1:tsne_result$N),
                                TSNE.1=tsne_result$Y[,1],TSNE.2=tsne_result$Y[,2],
                                Cluster=kmeans_result$cluster,Batch=merged_ID)
```

To further investigate potential batch effects in the data, we recommend coloring the same projection by batch labels (Figure 6). This is because clustering results can sometimes reflect technical effects depending on the nature of the study. In our case, the two samples consist of healthy PBMCs from the same donor, and the two batches are well-mixed in the data. In other cases where the batches are not limited to technical effects, it is reasonable to see some cell populations seperated by batches.

```
p_c <- visualize_clusters(merged_tsne_clust$Cluster,merged_tsne_clust[c("TSNE.1","TSNE.2")],
        title="k-means clustering labels")
p_b <- visualize_clusters(merged_tsne_clust$Batch,merged_tsne_clust[c("TSNE.1","TSNE.2")],
        title="batch labels")
multiplot(p_c, p_b, cols=2)
```

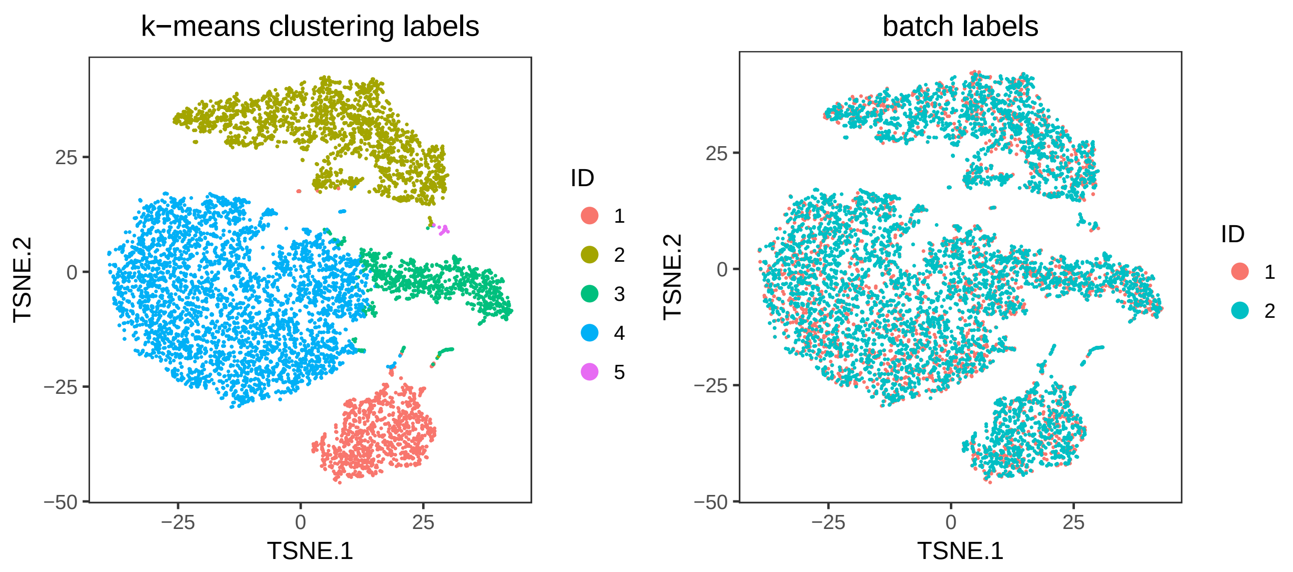

Figure 6: t-SNE projection where (Left) each cell is colored by cluster assigned in kmeans clustering, (Right) each cell is colored by sample ID, where 1 represents pbmc3k, and 2 represents pbmc6k.

Finally, you can repeat the analysis shown in the previous section on `merged_gbm` to identify cluster specific genes from the new clustering result.

6 Subsetting the matrix and reanalyzing

You can select submatrices of the matrix either by cell or by gene as if it were a standard matrix in R. The first dimension (rows) consists of genes and the second dimension (columns) consists of cell-barcodes.

```
subset_by_cell <- gbm[,c("AAACATACAACCAC-1", "AAACATTGAGCTAC-1")]
subset_by_gene_id <- gbm["ENSG00000167286",]
subset_by_gene_symbol <- gbm[which(fData(gbm)$symbol == 'CD3D'),]
subset_by_cell_and_gene <- gbm["ENSG00000167286", c("AAACATACAACCAC-1", "AAACATTGAGCTAC-1")]

# You can then use the exprs, fData and pData functions on this subsetted object
table(exprs(subset_by_gene_symbol)[1,])

##
##    0    1    2    3    4    5    6    7    8    9   10   11   12   15   24
## 1295  422  338  273  155   98   51   34   11    6    7    2    2    1    1
##   27   29   36   40
##    1    1    1    1
```

6.1 Re-running PCA, t-SNE and clustering using ”cellranger reanalyze”

After manipulating a matrix, for example by subsetting barcodes or genes, you can save the matrix to a new HDF5 file and pass it to the `cellranger reanalyze` pipeline. This requires access to a Linux machine where Cell Ranger is installed.

```
# Here, subset_by_cell_and_gene is a GeneBCMatrix that has been manipulated as shown above.
save_cellranger_matrix_h5(subset_by_cell_and_gene, 'new_matrix.h5', 'hg19')

# Outside of R, on the command line:
# cellranger reanalyze --id=subsetted_analysis --matrix=new_matrix.h5

# Back in R, you can load in the results of cellranger reanalyze
subset_results <- load_cellranger_analysis_results('subsetted_analysis')
```

7 Session Info

```
sessionInfo()

## R version 3.3.2 (2016-10-31)
## Platform: x86_64-redhat-linux-gnu (64-bit)
## Running under: Amazon Linux AMI 2017.03
##
## 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  methods   stats     graphics  grDevices utils     datasets
## [8] base
##
## other attached packages:
##  [1] cellrangerRkit_2.0.0 Rmisc_1.5            plyr_1.8.4
##  [4] lattice_0.20-35      bit64_0.9-7          bit_1.1-12
##  [7] ggplot2_2.2.1        RColorBrewer_1.1-2   Biobase_2.34.0
## [10] BiocGenerics_0.20.0  Matrix_1.2-10        knitr_1.16
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.11      magrittr_1.5      zlibbioc_1.20.0
##  [4] munsell_0.4.3     colorspace_1.3-2  rlang_0.1.1
##  [7] highr_0.6         stringr_1.2.0     tools_3.3.2
## [10] grid_3.3.2        rhdf5_2.18.0      data.table_1.10.4
## [13] gtable_0.2.0      irlba_2.2.1       digest_0.6.12
## [16] lazyeval_0.2.0    tibble_1.3.3      reshape2_1.4.2
## [19] Rtsne_0.13        evaluate_0.10     labeling_0.3
## [22] pheatmap_1.0.8    stringi_1.1.5     scales_0.4.1
```