Walkthrough – Aligning PBMC Data

Velina Kozareva

8/2/2019

This walkthrough steps through a basic analysis and alignment of two datasets of peripheral blood mononuclear cells (PBMCs), provided by 10X and Gierahn et al., 2017. We show how LIGER can be used to identify trends and clusters across both datasets, in addition to dataset-specific differences. We also demonstrate some of the package's visualization capabilities while reproducing some of the figure panels from the manuscript (Fig. 2).

The data can be downloaded from `vignettes/pbmc_alignment.zip`. First we set up the `liger` object using the 10X and Seqwell datasets provided in this directory.

```r
library(liger)
library(cowplot)

pbmc.10x <- read.table('~/Downloads/pbmc_alignment/pbmc_10X.expressionMatrix.txt',
                       sep="\t",stringsAsFactors=F,header=T,row.names = 1)
pbmc.seqwell <- read.table('~/Downloads/pbmc_alignment/pbmc_SeqWell.expressionMatrix.txt',
                           sep="\t",stringsAsFactors=F,header=T,row.names = 1)
pbmc.data = list(tenx=pbmc.10x, seqwell=pbmc.seqwell)

# Create liger object
a.pbmc <- createLiger(pbmc.data)
```

Data Preprocessing

Before running the factorization, we need to normalize the data to account for different numbers of UMIs per cell, select variable genes, and scale the data. Note that we do not center the data when scaling because nonnegative matrix factorization accepts only positive values.

The `selectGenes` function performs variable gene selection on each of the datasets separately, then takes the union of the result. The most variable genes are selected by comparing the variance of each gene's expression to its mean expression and keeping those with var/mean ratio above a certain threshold.

We can also simply pass pre-selected variable genes to the object if we've determined them from another source. In this case, we are using the variable genes determined by the Seurat package for their analysis of these two datasets (this was done in the manuscript for a cleaner comparison of the methods).

```r
a.pbmc <- normalize(a.pbmc)
# Can pass different var.thresh values to each dataset if one seems to be contributing significantly
# more genes than the other
a.pbmc <- selectGenes(a.pbmc, var.thresh = c(0.3, 0.875), do.plot = F)

# In this case, we have a precomputed set of variable genes
s.var.genes <- readRDS('~/Downloads/pbmc_alignment/var_genes.RDS')
a.pbmc@var.genes <- s.var.genes
a.pbmc <- scaleNotCenter(a.pbmc)
```

Factorization

Next we perform integrative non-negative matrix factorization in order to identify shared and distinct metagenes across the datasets and the corresponding factor/metagene loadings for each cell. The most important parameters in the factorization are `k` (the number of factors) and `lambda` (the penalty parameter which limits the dataset-specific component of the factorization). The default value of `lambda=5.0` usually provides reasonable results for most analyses, although the `suggestLambda` function can be used to determine a more appropriate `lambda` value for the desired level of dataset alignment.

To determine the appropriate number of factors to use, we can use the `suggestK` function which plots median K-L divergence from the uniform distribution in the factor loadings as a function of `k`. We want to look for the section of the plot where this metric stops increasing as sharply (the "elbow" of the plot). In general, we should expect a positive correlation between the number of subgroups we expect to find in the analysis and the appropriate number of factors to use. Since the `suggestK` function can take more than 10 minutes to run, it can sometimes be useful to run a quick preliminary analysis with `k=20` to get an idea of whether a much higher number of factors is needed. (Note that the command provided below requests 5 cores – adjust this value if you have fewer cores available.)

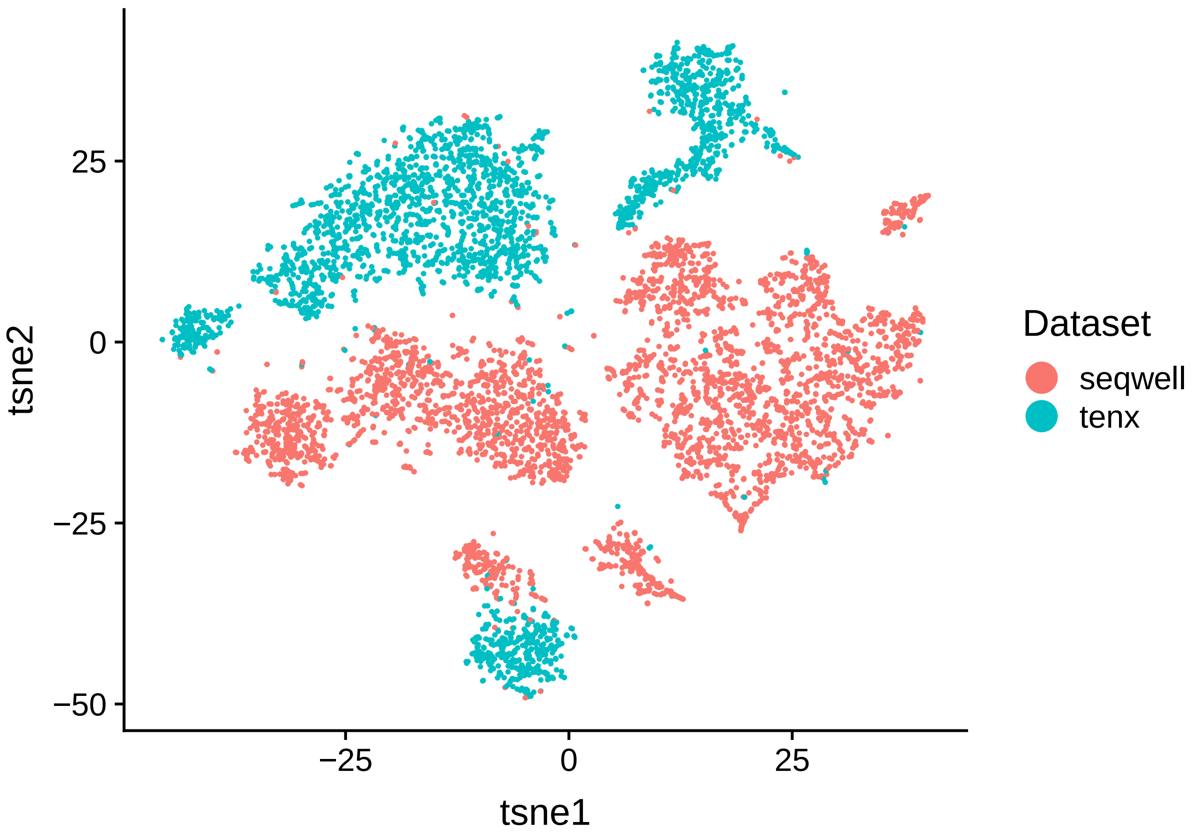
```r
# running suggestK on multiple cores can greatly decrease the runtime
k.suggest <- suggestK(a.pbmc, num.cores = 5, gen.new = T, return.results = T, plot.log2 = F,
                      nrep = 5)
```

For this analysis, we select a `k` of 22, though you can try various values in that range for similar results. We select the default `lambda=5.0`. (You can also run `suggestLambda` to get a plot of expected alignment vs. lambda for a given number of factors.)

```r
# Take the lowest objective of three factorizations with different initializations
# Multiple restarts are recommended for initial analyses since iNMF is non-deterministic
a.pbmc <- optimizeALS(a.pbmc, k=22, thresh = 5e-5, nrep = 3)
```

After the factorization, we still need to quantile align the factor loadings across the datasets. Notice that if we plot a t-SNE representation of the factor loadings, the data still cluster mainly by dataset.

```r
a.pbmc <- runTSNE(a.pbmc, use.raw = T)
p1 <- plotByDatasetAndCluster(a.pbmc, return.plots = T)
# Plot by dataset
print(p1[[1]])
```



To better integrate the datasets, we perform a quantile alignment step. This process first identifies similarly loading cells across datasets by building a similarity graph based on shared factor neighborhoods. Using Louvain community detection, we then identify clusters shared across datasets, and align quantiles within each cluster and factor. The key parameters in this step are the `resolution` (increasing this increases the number of communities detected) and `knn_k` (the number of dataset neighbors used in generating the shared factor neighborhood). In general, lowering `knn_k` will allow for more fine-grained identification of smaller groups with shared factor neighborhoods.

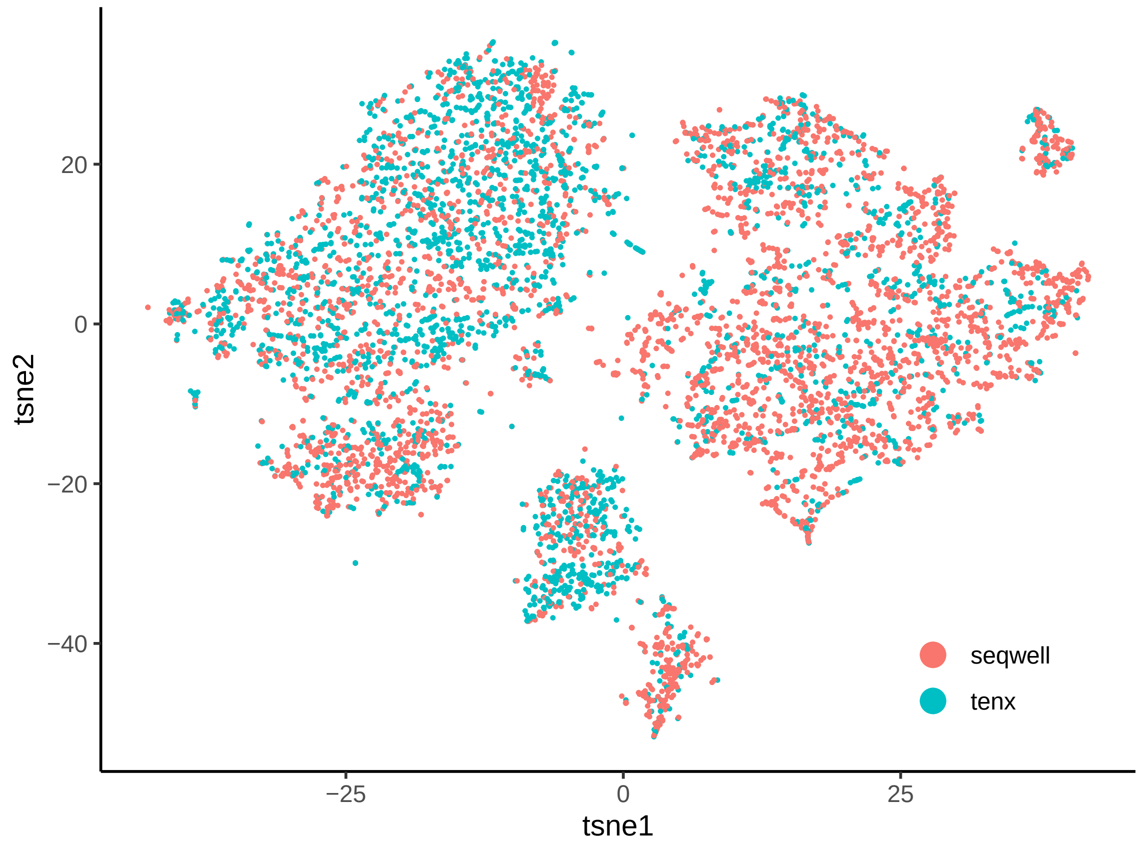
We can also try to extract even smaller clusters by setting the `small_clust_thresh` parameter equal to the current `knn_k`; we do this here since we expect a small group of megakaryocytes in the 10X dataset. We set the `resolution` to 0.4 to identify larger clusters, and use the default settings for the other parameters. (For a simpler analysis, and to perfectly reproduce the manuscript figures, you can use the default `small.clust.thresh = 0`, though this will prevent separation of the megakaryocytes.)

```r
a.pbmc <- quantileAlignSNF(a.pbmc, resolution = 0.4, small.clust.thresh = 20)
# Could also forgo small cluster extraction if desired
# a.pbmc <- quantileAlignSNF(a.pbmc, resolution = 0.4)
```

Visualization

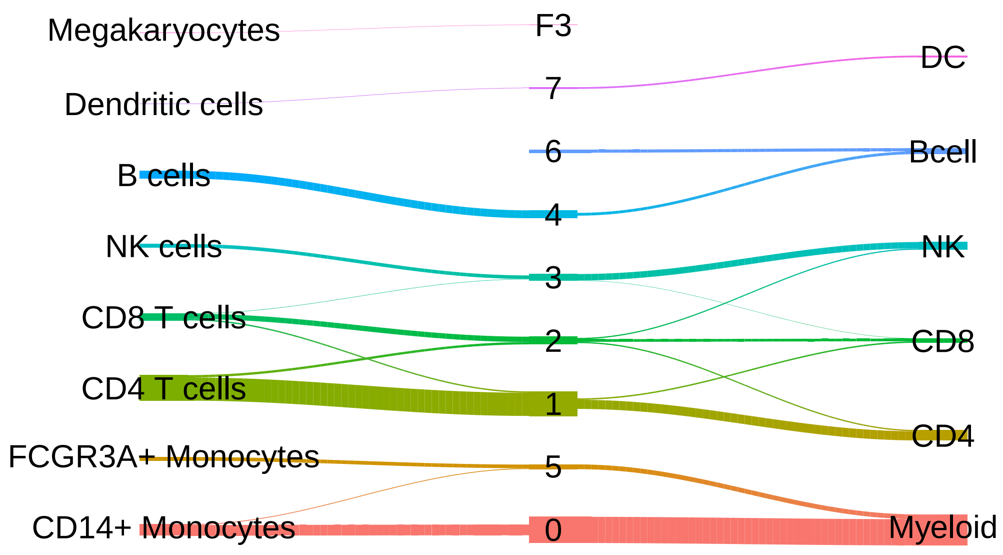
Now we can visualize the integrated data, and determine identities of the detected clusters.

```r
a.pbmc <- runTSNE(a.pbmc)
p_a <- plotByDatasetAndCluster(a.pbmc, return.plots = T)
# Modify plot output slightly
p_a[[1]] <- p_a[[1]] + theme_classic() + theme(legend.position = c(0.85, 0.15)) +
  guides(col=guide_legend(title = '', override.aes = list(size = 4)))

print(p_a[[1]])
```

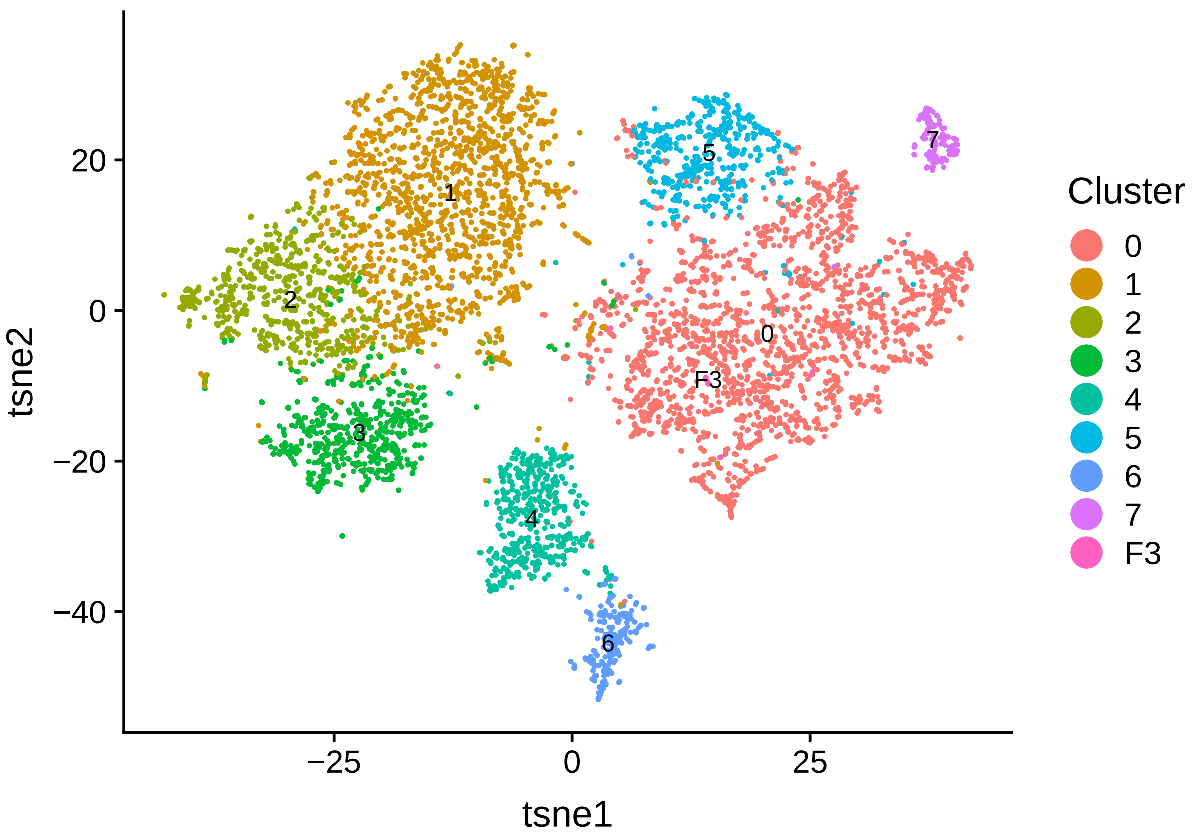


Since we have cluster information from the original publications/analyses for each of these two datasets, we can easily take a look at how our clustering compares using a river (i.e. Sankey) plot. Note that these clusterings were generated by individual (not joint) analyses of each dataset. The 10X labels come from the Seurat analysis detailed here and the SeqWell labels from the original SeqWell publication.

```
# load cluster labels from Seurat 10X PBMC analysis and original SeqWell publication
clusters_prior <- readRDS('~/Downloads/pbmc_alignment/tenx_seqwell_clusters.RDS')
tenx_c <- droplevels(clusters_prior[rownames(a.pbmc@H[[1]])])
seqwell_c <- droplevels(clusters_prior[rownames(a.pbmc@H[[2]])])

# Set specific node order for cleaner visualization (can also leave to default value for
# automatic ordering)
set_node_order = list(c(2, 6, 3, 4, 8, 1, 5, 7), c(1, 6, 2, 3, 4, 5, 7, 8, 9), c(5, 2, 3, 6, 1, 4))
makeRiverplot(a.pbmc, tenx_c, seqwell_c, min.frac = 0.05, node.order = set_node_order,
              river.usr = c(0, 1, -0.6, 1.6))
```



```r
# If using object with default small.clust.thresh settings, use this node order instead
# set_node_order = list(c(7, 2, 6, 3, 4, 8, 1, 5), c(1, 6, 2, 3, 4, 5, 7, 8), rev(c(4, 1, 6, 3, 2, 5)))

# plot t-SNE plot colored by clusters
print(p_a[[2]])
```



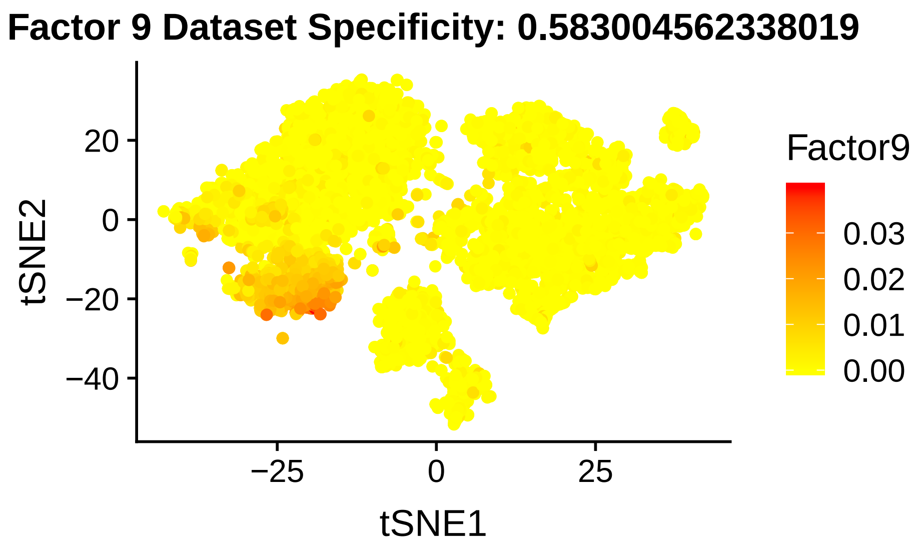
We see a good correspondance between analogous cluster labels from the two datasets and have successfully identified the small group of megakaryocytes!

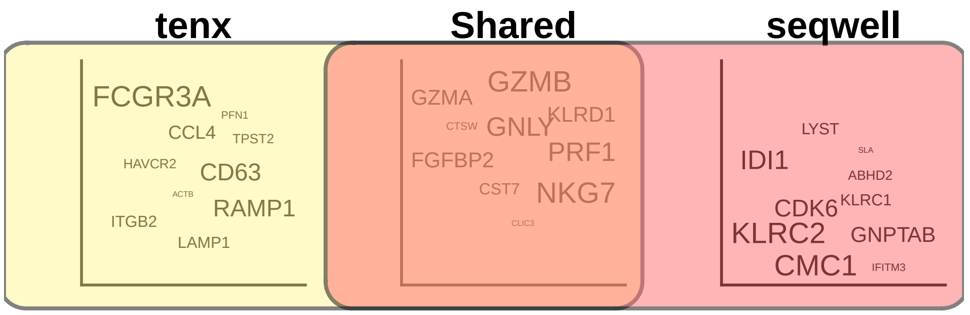
We can also identify some shared and dataset-specific markers for each factor and plot them to help in cluster annotation. This information can aid in finding cell-type specific dataset differences. We can return these in table format with `getFactorMarkers`, though an easy way to visualize these markers (and distribution of factor loadings) is with the package's `plotWordClouds` function. Here we can notice that in the plot of factor 9, which seems to correspond to the NK cell cluster, one of the most highly-loading dataset-specific markers is CD16 (FCGR3A).

```r
# This function uses the V loadings to identify dataset-specific genes and a combination of the W
# and V loadings to identify shared markers
markers <- getFactorMarkers(a.pbmc, dataset1='tenx', dataset2='seqwell', num.genes = 10)
# The first three elements in this list are the dataset1-specific, shared, and dataset2-specific
# markers respectively
# Let's take a look at factor 9
head(markers$tenx[markers$tenx$factor_num == 9, ])
```

```
##        factor_num   gene counts1  counts2    fracs1     fracs2   log2fc
## FCGR3A          9 FCGR3A     311 270.2475 0.8253968 0.17889908 3.369360
## RAMP1           9  RAMP1      38 105.1376 0.2301587 0.06422018 1.739324
## CD63            9   CD63     227 850.0802 0.7222222 0.43119266 1.239062
## CCL4            9   CCL4     486 950.6504 0.6904762 0.31192661 2.195697
## LAMP1           9  LAMP1      76 286.2575 0.4523810 0.17889908 1.282764
## ITGB2           9  ITGB2     278 712.4125 0.8253968 0.36697248 1.793409
##             p_value
## FCGR3A 8.306949e-37
## RAMP1  1.043517e-05
## CD63   2.896254e-10
## CCL4   2.117245e-14
## LAMP1  8.163031e-08
## ITGB2  2.078076e-19
```

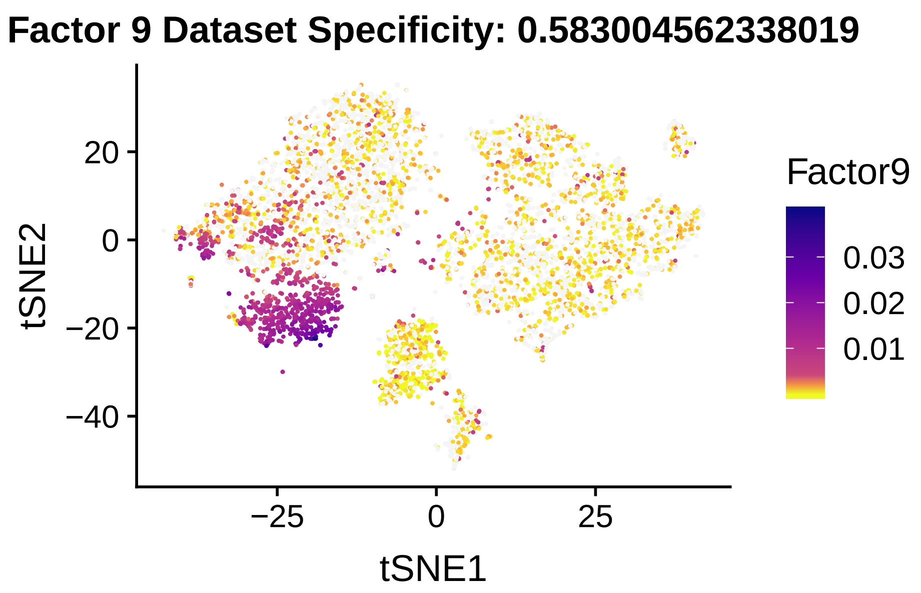
```
# Show top 10 highly loading dataset specific and shared genes
# Prevent text progress bar from printing
word_clouds <- plotWordClouds(a.pbmc, num.genes = 10, do.spec.plot = F, return.plots = T)
print(word_clouds[[9]])
```

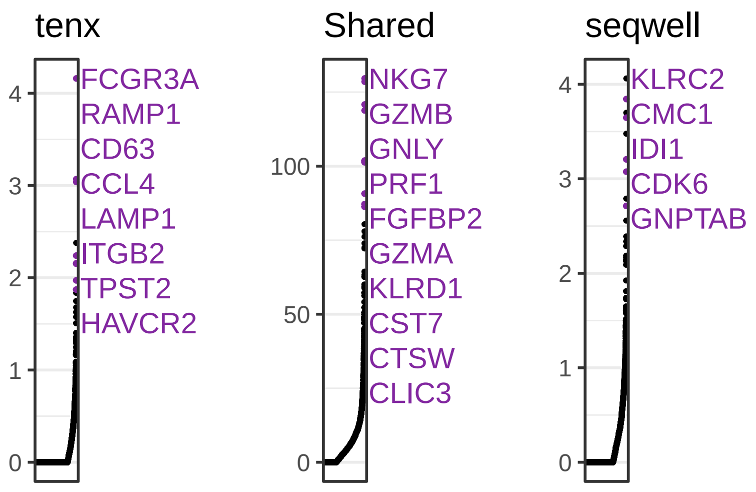




A related type of visualization is a gene loadings plot, which may allow for easier comparison of the most highly-loading genes:

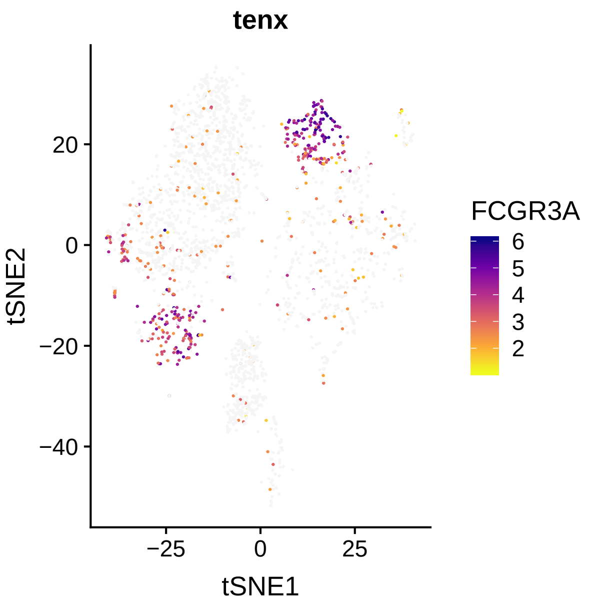
```
# Show top 10 highly loading dataset specific and shared genes
# with plots of gene loading values
gene_loadings <- plotGeneLoadings(a.pbmc, num.genes = 10, do.spec.plot = F, return.plots = T)
print(gene_loadings[[9]])
```

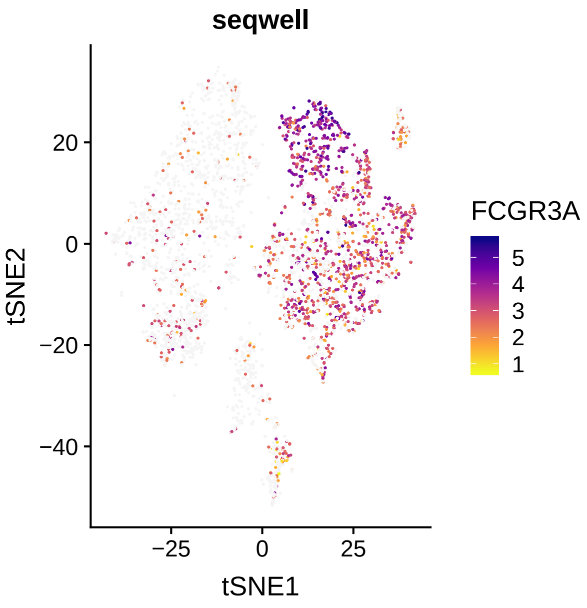




We can now plot this gene to better observe finer-grained dataset specific differences.

```
p_g2 <- plotGene(a.pbmc, 'FCGR3A', return.plots = T)
plot_grid(plotlist = p_g2)
```





Note that with these plots, we can confirm that the NK cluster in the seqwell dataset does not seem to express CD16 as highly as that from the 10X dataset (although as expected many of the myeloid cells express this marker). This might suggest some additional cytokine activation in the 10X patient.