Multimodal.R

```
library(Seurat)
```

```
## Warning: package 'Seurat' was built under R version 3.5.3
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
# Load in the RNA UMI matrix

# Note that this dataset also contains ~5% of mouse cells, which we can use as negative controls
# for the protein measurements. For this reason, the gene expression matrix has HUMAN_ or MOUSE_
# appended to the beginning of each gene.
# This file is too big!! use pre-collapsed data from maunish instead
#cbmc.rna <- as.sparse(read.csv(file = #"~/HarderLab/singlecellgenomicspractice/multimodal_tutorial/GSE
#    header = TRUE, row.names = 1))

# To make life a bit easier going forward, we're going to discard all but the top 100 most
# highly expressed mouse genes, and remove the 'HUMAN_' from the CITE-seq prefix
# Due to issue with file size, I'm just starting with the collapsed file.
cbmc.rna <- get(load("~/HarderLab/singlecellgenomicspractice/multimodal_tutorial/cbmc.RData"))

# Load in the ADT UMI matrix
cbmc.adt <- as.sparse(read.csv(file = "~/HarderLab/singlecellgenomicspractice/multimodal_tutorial/GSE10
    header = TRUE, row.names = 1))

# When adding multimodal data to Seurat, it's okay to have duplicate feature names. Each set of
# modal data (eg. RNA, ADT, etc.) is stored in its own Assay object.  One of these Assay objects
# is called the 'default assay', meaning it's used for all analyses and visualization.  To pull
# data from an assay that isn't the default, you can specify a key that's linked to an assay for
# feature pulling.  To see all keys for all objects, use the Key function.  Lastly, we observed
# poor enrichments for CCR5, CCR7, and CD10 - and therefore remove them from the matrix
# (optional)
cbmc.adt <- cbmc.adt[setdiff(rownames(x = cbmc.adt), c("CCR5", "CCR7", "CD10")), ]

cbmc <- CreateSeuratObject(counts = cbmc.rna)
```

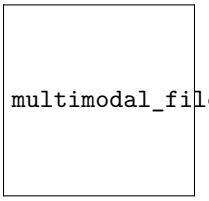
```
## Warning: Feature names cannot have underscores ('_'), replacing with dashes
## ('-')
```

```
# standard log-normalization
cbmc <- NormalizeData(cbmc)

# choose ~1k variable features
cbmc <- FindVariableFeatures(cbmc)

# standard scaling (no regression)
cbmc <- ScaleData(cbmc)
```

```
## Centering and scaling data matrix
```

```
# Run PCA, select 13 PCs for tSNE visualization and graph-based clustering
cbmc <- RunPCA(cbmc, verbose = FALSE)
```

```
plot(ElbowPlot(cbmc, ndims = 50))
```




```
cbmc <- FindNeighbors(cbmc, dims = 1:25)
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
cbmc <- FindClusters(cbmc, resolution = 0.8)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 8617
## Number of edges: 347548
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8592
## Number of communities: 19
## Elapsed time: 2 seconds
```

```
cbmc <- RunTSNE(cbmc, dims = 1:25, method = "FIt-SNE")

# Find the markers that define each cluster, and use these to annotate the clusters, we use
# max.cells.per.ident to speed up the process
cbmc.rna.markers <- FindAllMarkers(cbmc, max.cells.per.ident = 100, min.diff.pct = 0.3, only.pos = TRUE)
```

```
## Calculating cluster 0
```

```
## Calculating cluster 1
```

```
## Calculating cluster 2
```

```
## Calculating cluster 3
```

```
## Calculating cluster 4
```

```
## Calculating cluster 5
```

```
## Calculating cluster 6
```

```
## Calculating cluster 7
```

```
## Calculating cluster 8
```

```
## Calculating cluster 9
```

```
## Calculating cluster 10
```

```
## Calculating cluster 11
```

```
## Calculating cluster 12
```

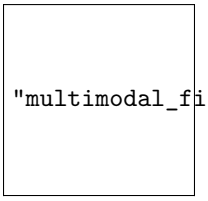
```
## Calculating cluster 13
```

```
## Calculating cluster 14
```

```
## Calculating cluster 15
```

```
## Calculating cluster 16

## Calculating cluster 17

## Calculating cluster 18
```

```
# Note, for simplicity we are merging two CD14+ Monocyte clusters (that differ in expression of
# HLA-DR genes) and NK clusters (that differ in cell cycle stage)
new.cluster.ids <- c("Memory CD4 T", "CD14+ Mono", "Naive CD4 T", "NK", "CD14+ Mono", "Mouse", "B",
    "CD8 T", "CD16+ Mono", "T/Mono doublets", "NK", "CD34+", "Multiplets", "Mouse", "Eryth", "Mk",
    "Mouse", "DC", "pDCs")
names(new.cluster.ids) <- levels(cbmc)
cbmc <- RenameIdents(cbmc, new.cluster.ids)
plot(DimPlot(cbmc, label = TRUE) + NoLegend())
```



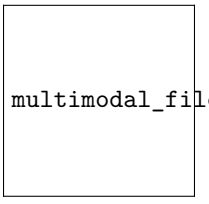

```
# We will define an ADT assay, and store raw counts for it

# If you are interested in how these data are internally stored, you can check out the Assay
# class, which is defined in objects.R; note that all single-cell expression data, including RNA
# data, are still stored in Assay objects, and can also be accessed using GetAssayData
cbmc[["ADT"]] <- CreateAssayObject(counts = cbmc.adt)

# Now we can repeat the preprocessing (normalization and scaling) steps that we typically run
# with RNA, but modifying the 'assay' argument.  For CITE-seq data, we do not recommend typical
# LogNormalization. Instead, we use a centered log-ratio (CLR) normalization, computed
# independently for each feature.  This is a slightly improved procedure from the original
# publication, and we will release more advanced versions of CITE-seq normalizations soon.
cbmc <- NormalizeData(cbmc, assay = "ADT", normalization.method = "CLR")
```

```
## Normalizing across features
```

```
cbmc <- ScaleData(cbmc, assay = "ADT")
```

```
## Centering and scaling data matrix
```

```
# in this plot, protein (ADT) levels are on top, and RNA levels are on the bottom
plot(FeaturePlot(cbmc, features = c("adt_CD3", "adt_CD11c", "adt_CD8", "adt_CD16", "CD3E", "ITGAX", "CD8
    "FCGR3A"), min.cutoff = "q05", max.cutoff = "q95", ncol = 4))
```

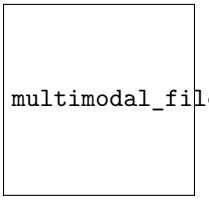



```
plot(RidgePlot(cbmc, features = c("adt_CD3", "adt_CD11c", "adt_CD8", "adt_CD16"), ncol = 2))
```

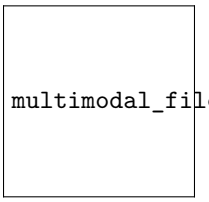
```
## Picking joint bandwidth of 0.0848

## Picking joint bandwidth of 0.1
```

```
## Picking joint bandwidth of 0.142
```

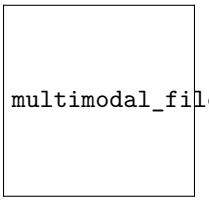
```
## Picking joint bandwidth of 0.0862
```



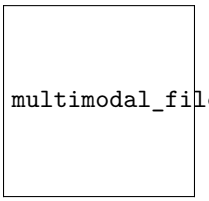

```
# Draw ADT scatter plots (like biaxial plots for FACS). Note that you can even 'gate' cells if
# desired by using HoverLocator and FeatureLocator
plot(FeatureScatter(cbmc, feature1 = "adt_CD19", feature2 = "adt_CD3"))
```




```
# Let's plot CD4 vs CD8 levels in T cells
tcells <- subset(cbmc, idents = c("Naive CD4 T", "Memory CD4 T", "CD8 T"))
plot(FeatureScatter(tcells, feature1 = "adt_CD4", feature2 = "adt_CD8"))
```




```
# # Let's look at the raw (non-normalized) ADT counts. You can see the values are quite high,
# particularly in comparison to RNA values. This is due to the significantly higher protein copy
# number in cells, which significantly reduces 'drop-out' in ADT data
plot(FeatureScatter(tcells, feature1 = "adt_CD4", feature2 = "adt_CD8", slot = "counts"))
```




```
# Downsample the clusters to a maximum of 300 cells each (makes the heatmap easier to see for
# small clusters)
cbmc.small <- subset(cbmc, downsample = 300)

# Find protein markers for all clusters, and draw a heatmap
adt.markers <- FindAllMarkers(cbmc.small, assay = "ADT", only.pos = TRUE)
```

```
## Calculating cluster Memory CD4 T
```

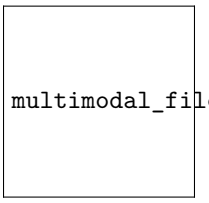
```
## Calculating cluster CD14+ Mono
```

```
## Calculating cluster Naive CD4 T
```

```
## Calculating cluster NK
```

```
## Calculating cluster Mouse
## Calculating cluster B
## Calculating cluster CD8 T
## Calculating cluster CD16+ Mono
## Calculating cluster T/Mono doublets
## Calculating cluster CD34+
## Calculating cluster Multiplets
## Calculating cluster Eryth
## Calculating cluster Mk
## Calculating cluster DC
## Calculating cluster pDCs
```

```
plot(DoHeatmap(cbmc.small, features = unique(adt.markers$gene), assay = "ADT", angle = 90) + NoLegend())
```

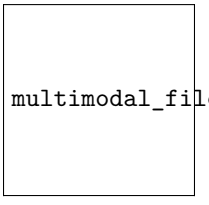



```
# You can see that our unknown cells co-express both myeloid and lymphoid markers (true at the
# RNA level as well). They are likely cell clumps (multiplets) that should be discarded. We'll
# remove the mouse cells now as well
# --> discard doublets
cbmc <- subset(cbmc, idents = c("Multiplets", "Mouse"), invert = TRUE)

# Because we're going to be working with the ADT data extensively, we're going to switch the
# default assay to the 'CITE' assay.  This will cause all functions to use ADT data by default,
# rather than requiring us to specify it each time
DefaultAssay(cbmc) <- "ADT"
cbmc <- RunPCA(cbmc, features = rownames(cbmc), reduction.name = "pca_adt", reduction.key = "pca_adt_",
    verbose = FALSE)
```

```
## Warning in irlba(A = t(x = object), nv = npcs, ...): You're computing too
## large a percentage of total singular values, use a standard svd instead.
```

```
## Warning: All keys should be one or more alphanumeric characters followed by
## an underscore '_', setting key to pca_
```

```
plot(DimPlot(cbmc, reduction = "pca_adt"))
```




```
# Since we only have 10 markers, instead of doing PCA, we'll just use a standard euclidean
# distance matrix here.  Also, this provides a good opportunity to demonstrate how to do
# visualization and clustering using a custom distance matrix in Seurat
```

```
adt.data <- GetAssayData(cbmc, slot = "data")
adt.dist <- dist(t(adt.data))


# Before we recluster the data on ADT levels, we'll stash the RNA cluster IDs for later
cbmc[["rnaClusterID"]] <- Idents(cbmc)

# Now, we rerun tSNE using our distance matrix defined only on ADT (protein) levels.
cbmc[["tsne_adt"]] <- RunTSNE(adt.dist, assay = "ADT", reduction.key = "adtTSNE_")
cbmc[["adt_snn"]] <- FindNeighbors(adt.dist)$snn
```

```
## Building SNN based on a provided distance matrix
```

```
## Computing SNN
```

```
cbmc <- FindClusters(cbmc, resolution = 0.2, graph.name = "adt_snn")
```

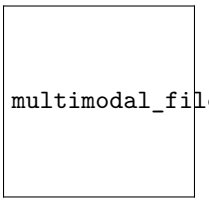
```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 7895
## Number of edges: 258146
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9491
## Number of communities: 11
## Elapsed time: 2 seconds
```

```
# We can compare the RNA and protein clustering, and use this to annotate the protein clustering
# (we could also of course use FindMarkers)
clustering.table <- table(Idents(cbmc), cbmc$rnaClusterID)
clustering.table
```

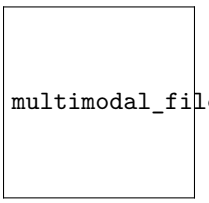
```
##
##      Memory CD4 T CD14+ Mono Naive CD4 T   NK    B CD8 T CD16+ Mono
##   0          1754          0        1217   29    0    27          0
##   1             0       2189           0    4    0     0         30
##   2             3          0           2  890    3     1          0
##   3             0          4           0    2  319     0          2
##   4            24          0          18    4    1   243          0
##   5             1         27           4  157    2     2         10
##   6             4          5           0    1    0     0          0
##   7             4         59           4    0    0     0          9
##   8             0          9           0    2    0     0        179
##   9             0          0           1    0    0     0          0
##   10            1          0           2    0   25     0          0
##
##      T/Mono doublets CD34+ Eryth   Mk   DC pDCs
##   0                5     2     4   24    1    2
##   1                1     1     5   25   55    0
##   2                0     1     3    7    2    1
##   3                0     2     2    3    0    0
##   4                0     0     1    2    0    0
##   5               56     0     9   16    6    2
##   6                1   113    81   16    5    0
##   7              117     0     0    2    0    1
##   8                0     0     0    1    0    0
```

```
##   9                0     0     0    0    1   43
##   10               2     0     0    0    0    0
```

```
new.cluster.ids <- c("CD4 T", "CD14+ Mono", "NK", "B", "CD8 T", "NK", "CD34+", "T/Mono doublets",
    "CD16+ Mono", "pDCs", "B")
names(new.cluster.ids) <- levels(cbmc)
cbmc <- RenameIdents(cbmc, new.cluster.ids)

tsne_rnaClusters <- DimPlot(cbmc, reduction = "tsne_adt", group.by = "rnaClusterID") + NoLegend()
tsne_rnaClusters <- tsne_rnaClusters + ggtitle("Clustering based on scRNA-seq") + theme(plot.title = el
tsne_rnaClusters <- LabelClusters(plot = tsne_rnaClusters, id = "rnaClusterID", size = 4)

tsne_adtClusters <- DimPlot(cbmc, reduction = "tsne_adt", pt.size = 0.5) + NoLegend()
tsne_adtClusters <- tsne_adtClusters + ggtitle("Clustering based on ADT signal") + theme(plot.title = el
tsne_adtClusters <- LabelClusters(plot = tsne_adtClusters, id = "ident", size = 4)

# Note: for this comparison, both the RNA and protein clustering are visualized on a tSNE
# generated using the ADT distance matrix.
plot(CombinePlots(plots = list(tsne_rnaClusters, tsne_adtClusters), ncol = 2))
```

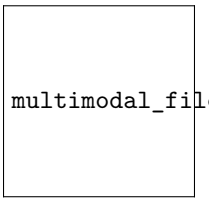



```
# Compare to RNA
tcells <- subset(cbmc, idents = c("CD4 T", "CD8 T"))
plot(FeatureScatter(tcells, feature1 = "CD4", feature2 = "CD8"))
```




```
plot(RidgePlot(cbmc, features = c("adt_CD11c", "adt_CD8", "adt_CD16", "adt_CD4", "adt_CD19", "adt_CD14"
    ncol = 2))
```

```
## Picking joint bandwidth of 0.0692

## Picking joint bandwidth of 0.0824

## Picking joint bandwidth of 0.0644

## Picking joint bandwidth of 0.0465

## Picking joint bandwidth of 0.0609

## Picking joint bandwidth of 0.055
```



```
pbmc10k.data <- Read10X(data.dir = "~/HarderLab/singlecellgenomicspractice/multimodal_tutorial/filtered
```

```
## 10X data contains more than one type and is being returned as a list containing matrices of each typ
```

```
rownames(x = pbmc10k.data[["Antibody Capture"]]) <- gsub(pattern = "_[control_]*TotalSeqB", replacement
    x = rownames(x = pbmc10k.data[["Antibody Capture"]]))

pbmc10k <- CreateSeuratObject(counts = pbmc10k.data[["Gene Expression"]], min.cells = 3, min.features =
pbmc10k <- NormalizeData(pbmc10k)
pbmc10k[["ADT"]] <- CreateAssayObject(pbmc10k.data[["Antibody Capture"]][, colnames(x = pbmc10k)])
pbmc10k <- NormalizeData(pbmc10k, assay = "ADT", normalization.method = "CLR")
```

```
## Normalizing across features
```

```
plot1 <- FeatureScatter(pbmc10k, feature1 = "adt_CD19", feature2 = "adt_CD3", pt.size = 1)
plot2 <- FeatureScatter(pbmc10k, feature1 = "adt_CD4", feature2 = "adt_CD8a", pt.size = 1)
plot3 <- FeatureScatter(pbmc10k, feature1 = "adt_CD3", feature2 = "CD3E", pt.size = 1)
CombinePlots(plots = list(plot1, plot2, plot3), ncol = 3, legend = "none")
```