Data Loading, Reduction, and Formatting

Data Loading

The following code loads a single cell RNA sequencing dataset, the Baron Pancreas dataset. Like the rest of the datasets, the Baron Pancreas dataset consists of several sample datasets. Each sample dataset in turn is made up of hundreds of cells.

```
#We load the necessary libraries
library(scRNAseq) #Needed to load scRNA-se data library
library(scran) #Needed for Normalization
```

```
#--- data loading ---#
sce.bar <- BaronPancreasData(ensembl = TRUE, location = TRUE)
```

```
## see ?scRNAseq and browseVignettes('scRNAseq') for documentation
```

```
## loading from cache
```

```
## see ?scRNAseq and browseVignettes('scRNAseq') for documentation
```

```
## loading from cache
## loading from cache
```

```
## require("ensembldb")
```

```
## Warning: Unable to map 1280 of 20125 requested IDs.
```

```
#--- normalization ---#
set.seed(1001)
clusters <- quickCluster(sce.bar)
sce.bar <- computeSumFactors(sce.bar, clusters=clusters)
sce.bar <- logNormCounts(sce.bar)
sce.bar <- sce.bar[,!is.na(sce.bar$label)]
table(sce.bar$label)#ground truth cell types
```

```
##
##             acinar activated_stellate              alpha               beta
##                958                284               2326               2525
##              delta             ductal        endothelial            epsilon
##                601               1077                252                 18
##              gamma         macrophage               mast quiescent_stellate
##                255                 55                 25                173
##            schwann             t_cell
##                 13                  7
```

```
table(sce.bar$donor) #unique sample Ids
```

```
##
## GSM2230757 GSM2230758 GSM2230759 GSM2230760
##       1937       1724       3605       1303
```

Dimension Reduction

Each dataset contains tens of thousans of genes. We use a very common tool of selecting 10% of the genes that drive the most variation. We then conduct PCA on these Highly Variable Genes (HVGs) using the denoisePCA() command. By default this command outputs as many principal components as needed to account for 90% of the variation .Assuming there are d principal components that account for 90% of the variation then each cell in the reduced dimension is represented by an d-dimensional vector. Assuming that there are N samples, we have that there are $m_1, m_2, ..., m_N)$ cells for samples $1, ..., N$ respectively. This means that there are a total of $m = m_1 + m_2 + ... + m_N$ cells in the dataset. Lastly note that sample i has n_i cell populations or clusters.

```
#--- variance-modelling ---#
set.seed(00010101)
dec.bar <- modelGeneVarByPoisson(sce.bar)
top.bar <- getTopHVGs(dec.bar, prop=0.1)

#--- dimensionality-reduction ---#
set.seed(101010011)
sce.bar <- denoisePCA(sce.bar, technical=dec.bar, subset.row=top.bar)
# m by d
dim(reducedDim(sce.bar))
```

```
## [1] 8569   50
```

Data Formatting

The data needs some formatting so that our method in Matlab can be applied. The following function takes the following inputs:

1. *sce.red*: Principal component data. For the Baron pancreas dataset we have $d = 50$ principal components and $m = 8569$ cells. The dimension of the reduced data is then 8569 by 50.
2. *samples*: A string vector of size $m = 8569$ for the baron pancreas dataset giving the sample each cell belongs to.
3. *disease*: A class vector of size $m = 8569$ indicating whether a cell belongs to a sample that is healthy or diseased. The Baron dataset does not have this vector available.
4. *cells*: A string vector of size $m = 8569$ containing the ground truth cell type name to which every cell belongs to.

For each cell population, $C_k^{(i)}$ where $i = 1, ..., N$ and $k = 1, ..., n_i$ (ie the clusters of all different samples) we compute the following:

1. A mean vector, $\mu_k^{(i)} \in \mathbb{R}^d$.
2. A $d \times d$ covariance matrix, $\Sigma_k^{(i)}$.
3. A proportion relative to its sample,$q_k^{(i)}$.

Note that there are $T = \sum_{i=1}^{N} n_i$ clusters in the entire dataset. The covariance matrix is vectorized and concatenated with the mean vector leading to a vector of length $d + d^2$. If we bind these columns horizontally we obtain a $(d + d^2) \times T$ matrix. We call this output matrix *supp* and store it in a list. To this same list we add:

- *ww*: a vector of length T, containing the proportion of each cluster relative to its sample.
- *stride*: a vector of length N, which has its entries the number of clusters per sample.

- Y: a vector of length N giving the class of sample, ie healthy vs diseased. This is not commonly available.
- *cellnames*: An array of length T containing the ground truth name of each cluster
- *numperclus*: A vector of length T containing the number of cells in each cluster.

```
gtData<-function(sce.red,samples,disease,cells){
  celltypes<-unique(cells)
  numtypes<-length(celltypes)
  donors<-unique(samples)
  cmeans <- c()
  props <- c()
  covars<-c()
  stride<-c()
  Y<-c()
  cellnames<-c()
  numperclus<-c()
  for (i in 1:length(unique(samples))){
    numperclus<-c(numperclus, sum(samples==donors[i]))
    if (!missing(disease)) {  #ie when disease per cells (hence per patient) is specified
      Y<-c(Y,unique(disease[samples==donors[i]]))
    }
    sce.donor<- sce.red[samples==donors[i],]
    cellsdonor<-cells[samples==donors[i]]
    nclust<-0
    for (j in 1:numtypes){
      cellsinij = sce.donor[cellsdonor==celltypes[j],, drop = FALSE ]
      if (nrow(cellsinij)>1){
        cellnames<-c(cellnames,celltypes[j])
        cmeans<-cbind(cmeans, colMeans(cellsinij))
        covars<-cbind(covars, c(cov(cellsinij)))
        props<-c(props,nrow(cellsinij)/sum(samples== donors[i]))
        nclust<-nclust+1
      }
      else if (nrow(cellsinij)==1){
        cmeans<-cbind(cmeans, t(cellsinij))
        covars<-cbind(covars, c(cov(sce.red)))
        props<-c(props,nrow(cellsinij)/sum(samples== donors[i]))
        cellnames<-c(cellnames,celltypes[j])
        nclust<-nclust+1
      }
    }
    stride<-c(stride, nclust)

  }
  if (!missing(disease)) {
    # Convert the list to a factor
    Y <- as.factor(unlist(Y))
    # Convert the factor to numeric
    Y <- as.numeric(Y)
    Y <- matrix(Y, nrow=length(Y), ncol=1)
  }else{
    Y<-"Unavailable"
  }
```

```
  supp = rbind(cmeans,covars)
  return(list("supp"=supp,"stride"=stride,"ww"=props,"Y"=Y,"cellnames"=cellnames,"numperclus"=numperclus
}
```

Let's apply the function to the PC Baron Pancreas dataset.

```
library(rmatio)
#setwd("C:/Users")
BaronResults<-gtData(reducedDim(sce.bar,"PCA"),sce.bar$donor,cells=sce.bar$label)
#write.mat(BaronResults, "baronpc.mat")
str(BaronResults)
```

```
## List of 6
##  $ supp      : num [1:2550, 1:56] 31.6907 -9.0719 0.0733 2.4466 -4.5184 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2550] "PC1" "PC2" "PC3" "PC4" ...
##   .. ..$ : chr [1:56] "" "" "" "" ...
##  $ stride    : num [1:4] 14 14 14 14
##  $ ww        : num [1:56] 0.0568 0.4502 0.1105 0.0263 0.062 ...
##  $ Y         : chr "Unavailable"
##  $ cellnames : chr [1:56] "acinar" "beta" "delta" "activated_stellate" ...
##  $ numperclus: int [1:4] 1937 3605 1724 1303
```

Simulated Samples

The Baron Pancreas dataset only has 4 samples and so we can randomly partition the dataset into 20 samples. We write a function that takes the PC scRNA-seq data and the number of desired samples. The output is a vector of length m which contains for each cell the sample to which it belongs to.

```
ss_Assigner<-function(sce,k){
# Get the number of columns (cells)
  num_cells <- ncol(sce)
# Create a vector of group labels
  group_labels <- rep(1:k, each = ceiling(num_cells / k))
# Randomly shuffle the group labels
  random_group_labels <- sample(group_labels, num_cells)
  return(random_group_labels[1:num_cells])
}
```

We apply the function and re-apply function *gtdata():*

```
k<-20
set.seed(191)
colData(sce.bar)$sim_samples<-ss_Assigner(sce.bar, k)
BaronResults <- gtData(reducedDim(sce.bar,"PCA"),sce.bar$sim_samples, cells = sce.bar$label)
str(BaronResults)
```

```
## List of 6
##  $ supp      : num [1:2550, 1:242] 35.6 -8 2.07 1.28 -3.22 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2550] "PC1" "PC2" "PC3" "PC4" ...
##   .. ..$ : chr [1:242] "" "" "" "" ...
##  $ stride    : num [1:20] 11 13 11 12 12 13 12 12 13 11 ...
##  $ ww        : num [1:242] 0.1329 0.2797 0.0769 0.0326 0.1166 ...
##  $ Y         : chr "Unavailable"
##  $ cellnames : chr [1:242] "acinar" "beta" "delta" "activated_stellate" ...
```

```
##  $ numperclus: int [1:20] 429 428 429 427 428 429 429 428 429 429 ...
```