Introduction to Creating Single-Cell Next-Generation Clustered Heat Maps (NG-CHMs) in R

Bradley Broom, Ganiraju Manyam, Rehan Akbani, John Weinstein

August 26, 2020

This is a sample R script that creates a single-cell Next-Generation Clustered Heat Map (NG-CHM). The goal is to create a demonstration NG-CHM with attached coordinates for dimension reduction plots (DRPs) and covariates of interest. A secondary goal is to demonstrate integration with the SingleCellExperiment data structure [1], which is commonly used in single-cell analysis in R. The published PDF report generated by the script was produced using the single-cell variant of our NG-CHM RStudio container. An accompanying Youtube video that demonstrates execution of this script and the resulting NG-CHM is available [2]. That video contains references to additional information on the key features of NG-CHMs, how we envisage their application to single-cell data, and our R package for building them. This script assumes familiarity with that background material.

Note that you don't need our RStudio environment. You can also use your own R environment provided the required packages are available. See the sessionInfo output at the end for details. Also, you don't strictly need to use SingleCellExperiment; alternative approaches would also work (but beyond the scope of this document).

This script uses the lung single-cell data published by Deprez et al. [3]: specifically,

- the paper was obtained from `https://www.biorxiv.org/content/10.1101/2019.12.21.884759v1`,
- the location of the data was obtained from `https://www.biorxiv.org/node/1071951.external-links.html`, and
- the data files were downloaded from `https://www.genomique.eu/cellbrowser/HCA/`.

All of the above were downloaded on July 6-7, 2020.

1 Overview of R Packages Used

This script uses the SingleCellExperiment [1] and NGCHM R packages extensively, so those packages are loaded and attached explicitly. The script also uses several other packages (notably RColorBrewer [4], fastcluster [5], Rtsne [6, 7], uwot [8]) in a few places.

```
library (SingleCellExperiment);
library (NGCHM);
```

2 Define the NG-CHM Generating Function genHM

The 'genHM' function creates an NG-CHM with the specified name from the specified SingleCellExperiment data object (sce). The function returns the generated NG-CHM, ready to be saved to a file or saved to a server. Auxiliary data required for generating the NG-CHM must be attached to the sce object prior to calling this function.

This function is presented now to give an overview of the steps involved in generating the NG-CHM and to provide context for the data preparation steps and helper functions described below.

Briefly,

- The function creates an NG-CHM with two layers: a row-centered copy of the data that is better for highlighting differences between (groups of) cells in a heat map and a layer containing the original data.

- It then attaches column covariate bars for the cell meta data, the pre-computed coordinates for the reduced dimension plots, and the pre-computed hierachical clusters.
- Finally, the row label type is set to "bio.gene.hugo" to enable gene-specific link outs from the row labels.

```
genHM <- function (name, sce) {
    # Get assay data.
    ad <- assay (sce);

    # Compute a row-centered copy of the data.
    rcd <- ad - rowData(sce)$Center;

    # Create an NGCHM with two data layers.
    hm <- chmNew (name,
                  chmNewDataLayer ('Row Centered', rcd),
                  chmNewDataLayer ('Original', ad));

    # Add visible column covariate bars for meta data values.
    hm <- addColumnCVs (hm, sce);
    # Add hidden column covariates for each reduced dimension.
    for (rd in names(reducedDims(sce))) {
        hm <- chmAddReducedDim (hm, "column", sce, rd, 2);
    }

    # Specify our precomputed hierarchical clusterings.
    chmColOrder(hm) <- attr (sce, 'col.hclust');
    chmRowOrder(hm) <- attr (sce, 'row.hclust');

    # Specify the type of the row labels to enable gene name specific linkouts.
    hm <- chmAddAxisType (hm, 'row', 'bio.gene.hugo');

    hm
};
```

The function is called at the end of the script once the helper functions have been defined, the single-cell data has been loaded, and the auxiliary data described above has been computed.

3 Loading the Data

This section of the script reads the data into R and creates a SingleCellExperiment object for it.

Since the data, in particular the expression data matrix, is so large, it takes a long time to load into R and process and uses a lot of memory. To reduce those time and memory requirements on subsequent executions of the script, the data will be subsetted and saved to an R data file that can be loaded and processed more quickly during iterative development.

3.1 Loading the saved data

The following fragment loads the saved data. It is normally executed on all runs but the first.

```
    load (file='deprez-sce-ngchm.Rsd');
```

3.2 Loading the full data for the first time

The code fragments in this section need to be executed the first time to input the data and create the saved data file.

Read the cell meta-data and the cluster marker genes. Both of these files are small and load quickly.

```
meta <- read.delim ("Deprez2019/meta.tsv", as.is=TRUE);
markers <- read.delim ("Deprez2019/markers.tsv", as.is=TRUE);
```

The following fragment reads the expression matrix and creates a SingleCellExperiment object. The code fragment also computes the means and standard deviations of each row (gene). This additional row data and the column data loaded above are included in the sce object. The complete sce object is saved to an R save file in case it's needed again.

```
    sce <- (function() {
        exprMatrix <- read.delim ("Deprez2019/exprMatrix.tsv", as.is=TRUE);
        # Create numeric matrix from column 2. Use column 1 for row names.
        tmp <- data.matrix (exprMatrix[,2:ncol(exprMatrix)]);
        rownames(tmp) <- exprMatrix[,1];
        # Compute mean and standard deviation of each gene (row).
        rcenter <- apply (tmp, 1, function (x) mean(x));
        rsd <- apply (tmp, 1, function(x) sd(x));
        rowmeta <- data.frame (Center=rcenter, StdDev=rsd,
                               row.names=rownames(tmp));
        # Create sce object
        SingleCellExperiment (tmp, rowData = rowmeta, colData = meta)
    })();
    save (sce, file='deprez-sce.Rsd')
```

3.3 Create data subsets

The following fragment creates subsets of sce of various sizes to enable faster iterative development. The cells in each subset are chosen at random. For the two smaller heat maps, the required number of the most variable (highest variance) genes are included. The largest heat map includes the 3000 genes in the second heat map plus all the genes in the markers file (that are in the data matrix).

```
    cell.order <- sample.int (ncol(sce), ncol(sce));
    gene.order <- order (rowData(sce)$StdDev, decreasing=TRUE);

    # 1000 most variable genes and 1000 random cells.
    sample.genes1k <- gene.order[1:1000];
    sample.cells1k <- cell.order[1:1000];
    sce1k <- sce[sample.genes1k,sample.cells1k];

    # 3000 most variable genes and 3000 random cells.
    sample.genes3k <- gene.order[1:3000];
    sample.cells3k <- cell.order[1:3000];
    sce3k <- sce[sample.genes3k,sample.cells3k];

    # 9518 unique marker genes in the data.
    marker.genes <- intersect (rownames(assay(sce)), unique (markers$genes));
    # Include any additional genes from sce3k. Brings total to 9653 genes.
    sample.genes10k <- unique (c (marker.genes, rownames(sce3k)));
    # Include approximately equal number of cells.
    sample.cells10k <- cell.order[1:10000];
    sce10k <- sce[sample.genes10k,sample.cells10k];
```

3.4 Create Reduced Dimensionality Data

The following section creates reduced dimensionality data for the SingleCellExperiment objects.

A relatively large number of principal components are generated so that they can be used as the basis for the other dimension reduction computations.

```
calcReducedDims <- function (sce) {
    pca_data <- prcomp (t(assay(sce)), rank=50);
    tsne_data <- Rtsne::Rtsne (pca_data$x, pca = FALSE);
    umap_data <- uwot::umap (pca_data$x);
    reducedDims(sce) <- list (UMAP=umap_data, TSNE=tsne_data$Y, PCA=pca_data$x)
    sce
};
```

```
    sce1k <- calcReducedDims (sce1k);
    sce3k <- calcReducedDims (sce3k);
    sce10k <- calcReducedDims (sce10k);
```

4 Attach the meta data as column covariates

This section of the script defines the function addColumnCVs that adds the experiment meta data to an NG-CHM.

The script first creates a colorTable that defines the colors to use for each covariate. The color table is defined using the meta data for the entire data set for consistency in colors between the different subsets of data. (This is most likely an issue for cell type, which has some rare values.)

The following helper function sets a colorTable entry for the specified covariate to the specified list of colors. For discrete covariates, the number of colors must equal the number of unique values for that covariate. Continuous covariates must have two colors (low and high).

```
setColors <- function (name, colors) {
    colorTable[[name]] <<- colors;
    if (mode(meta[[name]]) == "character") {
        names(colorTable[[name]]) <<- sort(unique (meta[[name]]));
        attr (colorTable[[name]], "mapType") <<- "discrete";
    } else {
        names(colorTable[[name]]) <<- c (min(meta[[name]]), max(meta[[name]]));
        attr (colorTable[[name]], "mapType") <<- "continuous";
    }
};
```

The following fragment actually creates the color table. The RColorBrewer package is used here to create colors for discrete covariates with many different possible values. In practice, many projects will use explicit color assignments.

```
    colorTable <- list();
    # Discrete covariates.
    setColors ("CellType",
                  c (RColorBrewer::brewer.pal (12, "Set3"),
                     RColorBrewer::brewer.pal (12, "Paired"),
                     RColorBrewer::brewer.pal (4, "Pastel2")));
    setColors ("Donor", RColorBrewer::brewer.pal (10, "Paired"));
    setColors ("Method", RColorBrewer::brewer.pal (3, "Set1")[c(1,3)]);
    setColors ("Position", RColorBrewer::brewer.pal (4, "Set1"));
    setColors ("Sex", c("#ff30ff", "#4030ff"));
    # Numeric covariates.
    setColors ("Age", c("#2020ff", "#ffff30"));
    setColors ("UMI.Count", c("#c7c7c7", "#101010"));
    setColors ("Expressed.Genes", c("#c7c7c7", "#101010"));
    setColors ("Percent.Mitochond.", c("#c7c7c7", "#f01010"));
    setColors ("Percent.Ribo", c("#c7c7c7", "#10f010"));
```

The getCovariate function creates a single NG-CHM covariate bar for the actual data in a specified SingleCellExperiment object (sce) and meta data column (name). The covariate bar will be given the same name unless an alternative is specified in covName.

```
getCovariate <- function (sce, name, covName) {
    if (missing (covName)) covName <- name;
    values <- sceGetNamedColumn (sce, name);
    if (attr (colorTable[[name]], "mapType") == "discrete") {
        uvalues <- sort (unique (values));
        cmap <- chmNewColorMap (uvalues, colorTable[[name]][uvalues]);
    } else {
            limits <- names(colorTable[[name]]);
            mode(limits) <- "numeric";
        cmap <- chmNewColorMap (limits, colorTable[[name]]);
    }
    chmNewCovariate (covName, values, cmap)
};
```

The following helper function gets the named column from the SingleCellExperiment's meta data and names it by the Id column.

```
sceGetNamedColumn <- function (sce, colname) {
    stopifnot (!missing(sce));
    stopifnot (!missing(colname));
    df <- colData (sce);
    cc <- df[,colname];
    names(cc) <- make.names(df[, "Id"]);
    cc
};
```

The addColumnCVs function is used by genHM to add each of the covariate bars to the heat map in the order in which they should appear.

CellType is added last, which will make it the default scatter plot color.

```
addColumnCVs <- function (hm, sce) {
    for (cv in list(
        getCovariate (sce, "Donor"),
        getCovariate (sce, "Method"),
        getCovariate (sce, "Position"),
        getCovariate (sce, "Sex"),
        getCovariate (sce, "Age"),
        getCovariate (sce, "UMI.Count", "UMI Count"),
        getCovariate (sce, "Expressed.Genes", "Expressed Genes"),
        getCovariate (sce, "Percent.Mitochond.", "Percent MitoChond"),
        getCovariate (sce, "Percent.Ribo", "Percent Ribo"),
        getCovariate (sce, "CellType", "Cell Type")
    )) {
        hm <- chmAddCovariateBar (hm, 'column', cv);
    }
    hm
};
```

5 Calculate hierarchical clusters for a SingleCellExperiment's row and column axes

This section of the script explicitly creates the hierarchical clusterings to be used on the axes of the NGCHMs. The clusterings are attached to the sce objects as attributes. The hierarchical clustering for the columns uses the same dimension-reduced principal components data as the other dimension reduction computations (t-SNE, UMAP).

```
calcHierClusters <- function (sce) {
    # Calculate the column hierarchical clustering based on the sce object's PC data.
    attr(sce, 'col.hclust') <- fastcluster::hclust.vector (reducedDim (sce, "PCA"),
                                                           method='ward');
    #attr(sce, 'col.hclust') <- hclust (dist (reducedDim (sce, "PCA")), method='ward.D2');

    # Calculate the row hierarchical clustering similarly, but we first need to
    # compute the PCs.
    gpca <- prcomp (assay(sce), rank=50);
    attr(sce, 'row.hclust') <- fastcluster::hclust.vector (gpca$x, method='ward');
    #attr(sce, 'row.hclust') <- hclust (dist (gpca$x), method='ward.D2');

    sce
};
```

```
    sce1k <- calcHierClusters (sce1k);
    sce3k <- calcHierClusters (sce3k);
    sce10k <- calcHierClusters (sce10k);
```

6 Save the SingleCellExperiment Subsets and Auxiliary data

The following fragment saves the subsetted SingleCellExperiment objects and auxiliary objects to another save file. It's much smaller than the save file containing the full SingleCellExperiment object.

```
    save (sce1k, sce3k, sce10k,
          meta, colorTable,
          cell.order, gene.order, marker.genes,
          file = 'deprez-sce-ngchm.Rsd');
```

7 Generate and save the NG-CHMs

The following fragment generates the NG-CHMs:

```
    hm1k <- genHM ("Deprez1000", sce1k);
    hm1k <- chmExportToFile (hm1k, 'Deprez1k.ngchm', TRUE);
    chmExportToPDF(hm1k, 'Deprez1k.pdf', TRUE);

    hm3k <- genHM ("Deprez3000", sce3k);
    hm3k <- chmExportToFile (hm3k, 'Deprez3k.ngchm', TRUE);
    chmExportToPDF(hm3k, 'Deprez3k.pdf', TRUE);

    hm10k <- genHM ("Deprez10000", sce10k);
    hm10k <- chmExportToFile (hm10k, 'Deprez10k.ngchm', TRUE);
    chmExportToPDF(hm10k, 'Deprez10k.pdf', TRUE);
```

Figure 1 shows a static rendering of the smallest heat map.

8 R Packages Used

```
    sessionInfo();

## R version 4.0.1 (2020-06-06)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04 LTS
##
## Matrix products: default
## BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/openblas-openmp/libopenblasp-r0.3.8.so
##
## 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=C
##  [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  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] NGCHM_0.12.8                SingleCellExperiment_1.10.1
##  [3] SummarizedExperiment_1.18.2 DelayedArray_0.14.1
##  [5] matrixStats_0.56.0          Biobase_2.48.0
##  [7] GenomicRanges_1.40.0        GenomeInfoDb_1.24.2
##  [9] IRanges_2.22.2              S4Vectors_0.26.1
## [11] BiocGenerics_0.34.0         knitr_1.29
##
## loaded via a namespace (and not attached):
##  [1] XVector_0.28.0         magrittr_1.5           zlibbioc_1.34.0
##  [4] lattice_0.20-41        R6_2.4.1               highr_0.8
##  [7] httr_1.4.1             stringr_1.4.0          tools_4.0.1
## [10] grid_4.0.1             xfun_0.15              tsvio_0.0.13.9004
## [13] digest_0.6.25          Matrix_1.2-18          GenomeInfoDbData_1.2.3
## [16] bitops_1.0-6           RCurl_1.98-1.2         evaluate_0.14
## [19] stringi_1.4.6          compiler_4.0.1         jsonlite_1.6.1
```

References

[1] Aaron Lun and Davide Risso. "SingleCellExperiment: S4 Classes for Single Cell Data. R package version 1.8.0." https://doi.org/doi:10.18129/B9.bioc.SingleCellExperiment, 2019.

[2] Next-Generation Clustered Heat Maps. "How to Create a Single-Cell Next-Generation Clustered Heat Map (NGCHM) in R." Youtube video. Retrieved from https://www.youtube.com/watch?v=WT0bD6vGNNc, July, 2020.

[3] Marie Deprez, Laure-Emmanuelle Zaragosi, Marin Truchi, Sandra Ruiz Garcia, Marie-Jeanne Arguel, Kevin Lebrigand, Agnès Paquet, Dana Pee'r, Charles-Hugo Marquette, Sylvie Leroy, Pascal Barbry. "A single-cell atlas of the human healthy airways." American Journal of Respiratory and Critical Care Medicine, doi: https://doi.org/10.1101/2019.12.21.884759, 2020.

[4] Erich Neuwirth. "RColorBrewer: ColorBrewer Palettes." https://cran.r-project.org/package=RColorBrewer.

[5] Daniel Müllner. "fastcluster: Fast Hierarchical, Agglomerative Clustering." Journal of Statistical Software, 53(9):1–18, 2013.

[6] L.J.P. van def Maaten and G.E. Hinton. "Visualizing High-Dimensional Data Using t-SNE." Journal of Machine Learning Research, 9:2579-2605, 2008.

[7] Jesse H. Krijthe. "Rtsne: T-Distributed Stochastic Neighbor Embedding Using Barnes-Hut Implementation." `https://cran.r-project.org/package=Rtsne`, 2015.

[8] James Melville, Aaron Lun, Mohamed Nadhir, Djekidel. "uwot: The Uniform Manifold Approximation and Projection (UMAP) Method for Dimensionality Reduction." `https://cran.r-project.org/package=uwot`

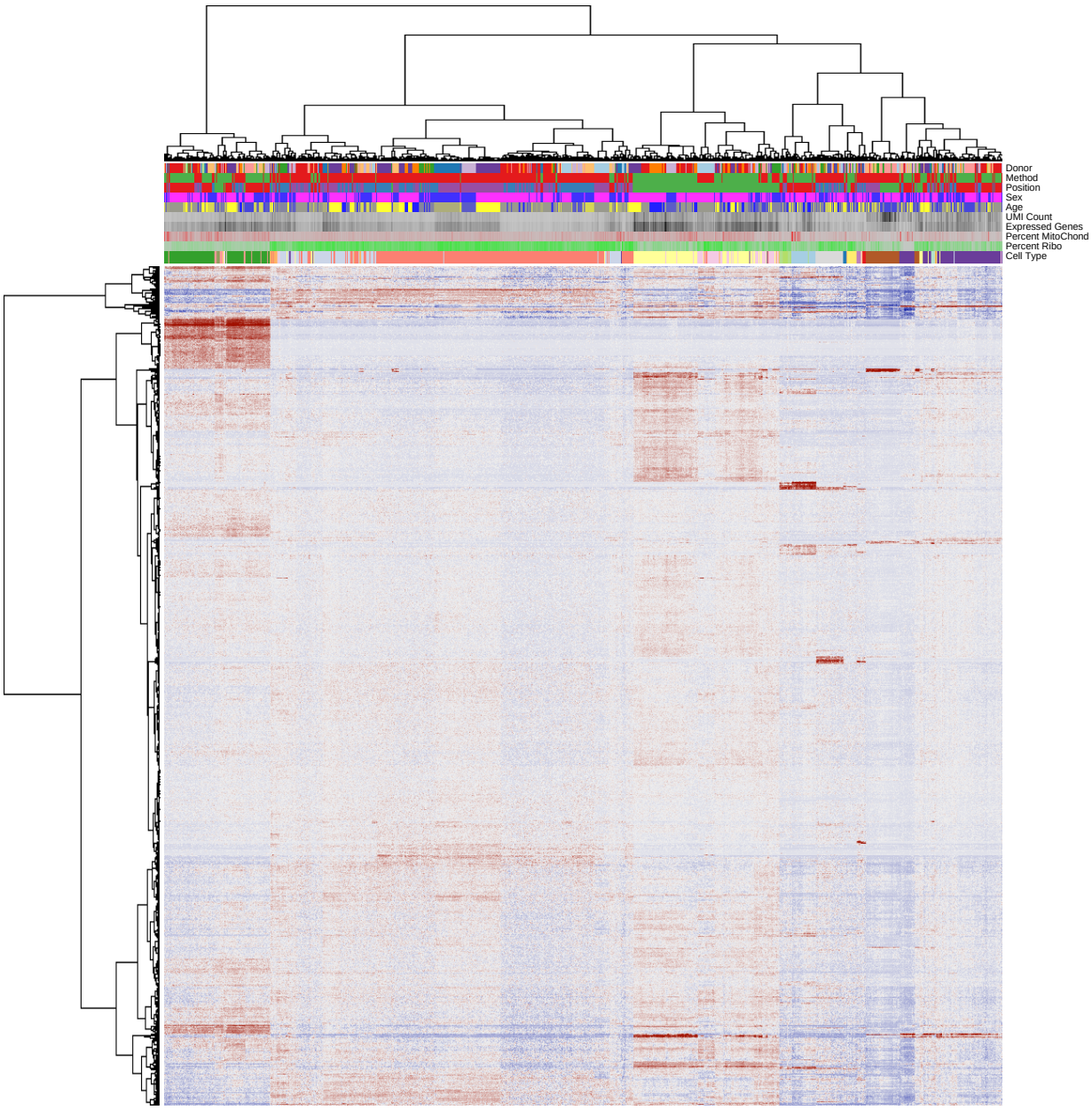

Figure 1: Static heat map for 3K random cells