03_TNBC_SPE

2022-11-19

Contents

Overview

The document will perform the following:

1. Construct a count matrix with written function `create_countmat()`
2. Construct a colData dataframe with written function `create_colData()`
3. Construct a spatial coordinates matrix
4. Construct spatial experiment object with count matrix, colData and spatial information

Libraries

```
library(readr)
library(dplyr)
library(tidyverse)
library(SpatialExperiment)
```

Reading the CSV output

Read CSV

Read in the MATLAB revised TNBC output CSV files.

```
TNBC <- readr::read_csv("/Users/henzhwang/Desktop/TNBC_training/MIBI-TNBC_scdata_counts_mm_matlab_revis
```

```
## Rows: 179194 Columns: 64
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): SITE_02, RECURRENCE_LABEL, mm
## dbl (61): sample_id, patient_id, AGE_AT_DX, STAGE, LATERAL, GRADE, Survival_...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Construct count matrix

The aim of this section is to create a count matrix to be used in creating the spatial experiment where the rows are cluster number, and the columns are counts od each cluster number per sample. The expected dimension of this count matrix is 113 rows(113 cluster number) and `r 39 * 113` columns (39 samples).

Test case with sample 1 and sample 2

We first want to create a sample count matrix with only using sample 1 and sample 2. Expected dimension is 113 rows by 226 columns. We first start with sample 1.

```
# frequency of cluster number in sample 1
test_freq_1 <- TNBC %>%
  mutate(cluster_id = as.factor(cluster_id)) %>%
  filter(sample_id == 1) %>%
  with(table(cluster_id, sample_id)) %>% data.matrix()

# length of frequency
test_length_1 <- length(test_freq_1)

# construct a diagonal matrix with frequency as the digonal values
# dimension: 113 x 113
test_count_1 <- matrix(data = diag(as.numeric(test_freq_1)),
                       nrow = test_length_1, ncol = test_length_1,
                       dimnames = dimnames(test_freq_1)[1])

# check
#head(test_count_1)
dim(test_count_1)
```

```
## [1] 113 113
```

We want to repeat the same procedure for sample 2.

```
# frequency of cluster number in sample 2
test_freq_2 <- TNBC %>%
  mutate(cluster_id = as.factor(cluster_id)) %>%
  filter(sample_id == 2) %>%
  with(table(cluster_id, sample_id)) %>% data.matrix()
```

```
# length of frequency
test_length_2 <- length(test_freq_2)

# construct a diagonal matrix with frequency as the digonal values
# dimension: 113 x 113
test_count_2 <- matrix(data = diag(as.numeric(test_freq_2)),
                       nrow = test_length_2, ncol = test_length_2,
                       dimnames = dimnames(test_freq_2)[1])

# check
#head(test_count_2)
dim(test_count_2)
```

```
## [1] 113 113
```

Now we want to merge this two matricies into one matrix using `cbind()`.

```
test_count_1n2 <- cbind(test_count_1, test_count_2)
#head(test_count_1n2)
dim(test_count_1n2)
```

```
## [1] 113 226
```

The dimensions of the sample count matrix `test_count_1n2` is 113 rows by 226 columns. We shall proceed in writing a function in order to create a count matrix for the entire `TNBC` dataset.

Application to create count matrix

We want to apply the above test case of sample 1 and sample 2 to the entire `TNBC` dataset. We will write a function `create_countMat()` which takes the dataset as an argument and return a count matrix for all samples for the input dataset.

The function performs its functionality by the following steps:

1. The function takes in the dataset and factors the variable `cluster_id` for factor level which benefits in counting the number of different cluster numbers
2. It makes a list `sampleId_list` of distinct sample number present in the dataset and set `count_matrix` to `NULL`
3. It uses for loop of the length of the `sampleId_list`. It then subset the dataset with the current sample number
4. It creates a frequency table of each cluster number in the subset(for each dataset) `tab` and finds the length of the frequency table `tab_length`
5. It creates a diagonal matrix with frequency values as the diagonal values and its dimension is `length of`tab_length`by`tab_length‘
6. It defines `count_matrix` if the current sample is the first sample in the dataset or else it column combines the current `count_matrix` and curren `tab` to create a new `count_matrix`
7. The function returns the count matrix `count_matrix` after processing all samples in the dataset

```
create_countMat <- function(dataset) {

  # make clsuter_id as factor
  dataset <- dataset %>%
    dplyr::mutate(cluster_id = as.factor(cluster_id))

  # list of distinct sample_id in the dataset
  sampleId_list <- sort(unique(dataset$sample_id))

  # set count_matrix to NULL
  count_matrix <- NULL

  # for loop to construct count matrix
  for (k in sampleId_list) {

    # subset dataset by corresponding sample number
    df <- dataset %>%
      dplyr::filter(sample_id == k)

    # create frequency table
    tab <- with(df, table(cluster_id, sample_id)) %>% data.matrix()

    # length od frequency
    tab_length <- length(tab)

    # create diagonal matrix
    diag_matrix <- matrix(data = diag(as.numeric(tab)),
                          nrow = tab_length, ncol = tab_length,
                          dimnames = dimnames(tab)[1])

    # create count matrix
    if (is.null(count_matrix)) {
      count_matrix <- diag_matrix
    } else {
      count_matrix <- cbind(count_matrix, diag_matrix)
    }
  }

  return(count_matrix)
}
```

Test Case: Next, we want to test this function with sample 1 and sample 2 with comparison to the sample count matrix `test_count_1n2`. The results shows that these two count matrix are the same.

```
TNBC_12 <- TNBC %>%
  mutate(cluster_id = factor(cluster_id)) %>%
  dplyr::filter(sample_id <= 2)
test_countMat <- create_countMat(TNBC_12)

#head(test)
all.equal(test_countMat, test_count_1n2)
```

```
## [1] TRUE
```

Application: We now may apply the function to the entire `TNBC` dataset and obtain its count matrix.

```
count_matrix <- create_countMat(TNBC)
```

```
dim(count_matrix)
```

```
## [1]  113 4407
```

```
sum(count_matrix)
```

```
## [1] 179194
```

```
isTRUE(sum(count_matrix) == nrow(TNBC))
```

```
## [1] TRUE
```

By verification, we have created the count matrix with expected dimensions of 113 rows by 4407 columns, and the sum of the count matrix is 179194 which equals the number of cells in the `TNBC` dataset.

Construct colData

The aim of this section is to create colData which to be used in creating the spatial experiment. The colData will contains 8 columns from the `TNBC` dataset, which are `sample_id`, `cluster_id`, `centroidX`, `centroidY`, `cellSize`, `cellRadius`, `STAGE`, and `Survival_days_capped`. These variables will be the columns of colData, and the rows are the count of each cluster_number per sample, and thus the expected dimension is 4407 rows(same length as number of columns in the count matrix) by 8 columns.

Test case with sample 1 and sample 2

We first want to create a sample colData with only using sample 1 and sample 2. Expected dimension is 113 rows by 8 columns. We first start with sample 1.

```
# extract reuqired columns
TNBC_colData_1n2<- TNBC %>%
  mutate(cluster_id = factor(cluster_id)) %>%
  filter(sample_id <= 2) %>%
  dplyr::select(c("sample_id", "cluster_id","centroidX", "centroidY",
                  "cellSize", "cellRadius", "STAGE", "Survival_days_capped"))
head(TNBC_colData_1n2)
```

```
## # A tibble: 6 x 8
##   sample_id cluster_id centroidX centroidY cellSize cellRadius STAGE Survival_~1
##       <dbl> <fct>          <dbl>     <dbl>    <dbl>      <dbl> <dbl>       <dbl>
## 1         1 34             1646.      6.73      211       7.82    33        2612
## 2         1 11             1694.      9.33      184       7.20    33        2612
## 3         1 31             1814.      8.85      277       8.94    33        2612
## 4         1 33             1797.     17.6       564      13.0     33        2612
## 5         1 31              419.     18.7       402      11.1     33        2612
## 6         1 31              403.     28.5       705      14.7     33        2612
## # ... with abbreviated variable name 1: Survival_days_capped
```

```
TNBC %>%
  group_by(sample_id) %>%
  summarise(STAGE = unique(STAGE), Survival_days_capped = unique(Survival_days_capped))
```

```
## # A tibble: 39 x 3
##    sample_id STAGE Survival_days_capped
##        <dbl> <dbl>                <dbl>
##  1         1    33                 2612
##  2         2    32                  745
##  3         3    21                 3130
##  4         4    22                 2523
##  5         5    11                 1683
##  6         6    10                 2275
##  7         7    32                  584
##  8         8    21                  946
##  9         9    21                 3767
## 10        10    22                 3822
## # ... with 29 more rows
```

```
# metadata list
# metadata_list <- data.frame(sample_id = unique(TNBC_colData_1n2$sample_id),
#                             STAGE = unique(TNBC_colData_1n2$STAGE),
#                             Survival_days_capped = unique(TNBC_colData_1n2$Survival_days_capped))

metadata_list <- TNBC_colData_1n2 %>%
  group_by(sample_id) %>%
  summarise(STAGE = unique(STAGE), Survival_days_capped = unique(Survival_days_capped))

# create colData dataframe
test_colData_1n2 <- TNBC_colData_1n2 %>%
  group_by(sample_id, cluster_id, .drop = FALSE) %>%
  summarise(centroidX  = round(median(centroidX),  digits = 3),
            centroidY  = round(median(centroidY),  digits = 3),
            cellSize   = round(median(cellSize),   digits = 3),
            cellRadius = round(median(cellRadius), digits = 3)) %>%
  replace(is.na(.), 0) %>% as.data.frame() %>%
  left_join(x = ., y = metadata_list, by = "sample_id", copy = TRUE)
head(test_colData_1n2)
```

```
##   sample_id cluster_id centroidX centroidY cellSize cellRadius STAGE
## 1         1          1     0.000     0.000        0      0.000    33
## 2         1          2   970.333  1919.676      102      5.285    33
## 3         1          3  1026.693  1704.242      203      7.660    33
## 4         1          4     0.000     0.000        0      0.000    33
## 5         1          5  1383.571   411.074      177      7.091    33
## 6         1          6     0.000     0.000        0      0.000    33
##   Survival_days_capped
## 1                 2612
## 2                 2612
## 3                 2612
## 4                 2612
## 5                 2612
## 6                 2612
```

We are now created the sample colData for sample 1 and sample 2. We want to verify if the dataframe contains the correct information and correct dimensions. By verification, we found that the sample colData dataframe has a dimension of 226 rows by 8 columns, and metadata `STAGE` and `Survival_days_capped` are correctly inputted.

```
# Verification of test colData
dim(test_colData_1n2)
```

```
## [1] 226   8
```

```
# Check if metadata STAGE and Survival_days_capped are corrretly inputted
test_colData_1n2 %>%
  group_by(sample_id) %>%
  summarise(num_unique_stage = length(unique(STAGE)),
            STAGE = unique(STAGE),
            num_unique_Sur = length(unique(Survival_days_capped)),
            Survival_days_capped = unique(Survival_days_capped))
```

```
## # A tibble: 2 x 5
##   sample_id num_unique_stage STAGE num_unique_Sur Survival_days_capped
##       <dbl>            <int> <dbl>          <int>                <dbl>
## 1         1                1    33              1                 2612
## 2         2                1    32              1                  745
```

Application to create colData

We want to apply the above test case of sample 1 and sample 2 to the entire `TNBC` dataset. We will write a function `create_colData()` which takes the dataset as an argument and return a colData data frame for all samples for the input dataset.

The function performs its functionality by the following steps:

1. It takes in three arguments, dataset, function to compute the spatial information, and a boolean variable stating whether the dataset is partila or not (Note: if `partial = TRUE`, the dataset must factor `cluster_id` for factor levels with the full dataset before using the function to obtain colData[By this, the function is able to output the colData grouped by `sample_id` with all `cluster_id` present in the full dataset], if desire)
2. It factors the `cluster_id` to obtain factor levels, and selects the required columns from the dataset
3. `funct()` will react on which computation methods are desired for computing spatial information
4. It creates a metadata list `metadata_list` which contains `sample_id`, `STAGE`, and `Survival_days_capped` for each sample
5. It uses package `dplyr` to group the dataset by `sample_id` and `cluster_id`, and compute the spatial information by `funct()`. Then it replaces all `NA`s in the data frame to 0, merges with `metadata_list` to acquire the metadata information for each sample, and constructs a data frame.
6. The function returns the colData data frame `colData`

```
create_colData <- function(dataset, fun = c(median,mean), partial = FALSE) {

  # factor cluster_id in the dataset
  ## !! if partial == TRUE,
  ## !! dataset must be factor cluster_id first for cluster_id levels
  if (isFALSE(partial)) {
    dataset <- dataset %>%
    dplyr::mutate(cluster_id = factor(cluster_id)) %>%
    dplyr::select(c("sample_id", "cluster_id","centroidX", "centroidY",
                   "cellSize", "cellRadius", "STAGE", "Survival_days_capped"))
  } else {
    dataset <- dataset %>%
    dplyr::select(c("sample_id", "cluster_id","centroidX", "centroidY",
                   "cellSize", "cellRadius", "STAGE", "Survival_days_capped"))
  }

  # decide on which methods in computing spatial information
  if (fun == "mean") {
    funct <- function (x) {
      mean(x)
    }
  } else if (fun == "median") {
    funct <- function (x) {
      median(x)
    }
  }


  # metadata list
  metadata_list <- dataset %>%
    group_by(sample_id) %>%
    summarise(STAGE = unique(STAGE),
              Survival_days_capped = unique(Survival_days_capped))

  # create colData dataframe
  colData <- dataset %>%
    group_by(sample_id, cluster_id, .drop = FALSE) %>%
    summarise(
      centroidX  = round(funct(centroidX),  digits = 3),
      centroidY  = round(funct(centroidY),  digits = 3),
      cellSize   = round(funct(cellSize),   digits = 3),
      cellRadius = round(funct(cellRadius), digits = 3)
    ) %>%
    replace(is.na(.), 0) %>% as.data.frame() %>%
    left_join(x = .,y = metadata_list,by = "sample_id", copy = TRUE)

  return(colData)
}
```

Test Case: Next, we want to test this function with sample 1 and sample 2 with comparison to the sample count matrix `test_count_1n2`. The results shows that these two count matrix are the same.

```
test_colData <- create_colData(TNBC_12, fun = "median", partial = TRUE)

head(test_colData)
```

```
##   sample_id cluster_id centroidX centroidY cellSize cellRadius STAGE
## 1         1          1     0.000     0.000        0      0.000    33
## 2         1          2   970.333  1919.676      102      5.285    33
## 3          1           3  1026.693  1704.242       203       7.660    33
## 4          1           4     0.000     0.000         0       0.000    33
## 5          1           5  1383.571   411.074       177       7.091    33
## 6          1           6     0.000     0.000         0       0.000    33
##   Survival_days_capped
## 1                 2612
## 2                 2612
## 3                 2612
## 4                 2612
## 5                 2612
## 6                 2612
```

```
all.equal(test_colData, test_colData_1n2)
```

```
## [1] TRUE
```

Application: We now may apply the function to the entire `TNBC` dataset and obtain its colData.

```
colData <- create_colData(TNBC, fun = "median", partial = FALSE)
```

```
# colData preview
head(colData)
```

```
##   sample_id cluster_id centroidX centroidY cellSize cellRadius STAGE
## 1         1           1     0.000     0.000         0       0.000    33
## 2         1           2   970.333  1919.676       102       5.285    33
## 3         1           3  1026.693  1704.242       203       7.660    33
## 4         1           4     0.000     0.000         0       0.000    33
## 5         1           5  1383.571   411.074       177       7.091    33
## 6         1           6     0.000     0.000         0       0.000    33
##   Survival_days_capped
## 1                 2612
## 2                 2612
## 3                 2612
## 4                 2612
## 5                 2612
## 6                 2612
```

```
# dimensions check
dim(colData)
```

```
## [1] 4407    8
```

```
# Check if metadata STAGE and Survival_days_capped are corrretly inputted
colData %>%
  group_by(sample_id) %>%
  summarise(num_unique_stage = length(unique(STAGE)),
            STAGE = unique(STAGE),
            num_unique_Sur = length(unique(Survival_days_capped)),
            Survival_days_capped = unique(Survival_days_capped))
```

```
## # A tibble: 39 x 5
##    sample_id num_unique_stage STAGE num_unique_Sur Survival_days_capped
##        <dbl>            <int> <dbl>          <int>                <dbl>
##  1         1                1    33              1                 2612
##  2         2                1    32              1                  745
##  3         3                1    21              1                 3130
##  4         4                1    22              1                 2523
##  5         5                1    11              1                 1683
##  6         6                1    10              1                 2275
##  7         7                1    32              1                  584
##  8         8                1    21              1                  946
##  9         9                1    21              1                 3767
## 10        10                1    22              1                 3822
## # ... with 29 more rows
```

By verification, we have created the count matrix with expected dimensions of 4407 rows by 8 columns, and the metadata information are correctly inputted.

Spatial information

The aim of this section is to create the spatial information for the spatial experiment object. It requires either `SpatialCoords`, a numeric matrix containing spatial spatial coordinates, or `SpatialCoordsNames`, where a character vector specifying which `colData` fields correspond to spatial coordinates. We will construct the spatial experiment obejct using both of the methods.

If we are using `SpatialCoords` approaches, we will need to create a matrix which contains the `centroidX` and `centroidY` from the `colData` dataframe. The expected dimensions of this matrix is 4407 rows by 2 columns.

```
spatialCoords <- as.matrix(colData[, c("centroidX", "centroidY")])
```

We verified that the `spatialCoords` matrix dimension is 4407 rows by 2 columns.

```
head(spatialCoords)
```

```
##      centroidX centroidY
## [1,]     0.000     0.000
## [2,]   970.333  1919.676
## [3,]  1026.693  1704.242
## [4,]     0.000     0.000
## [5,]  1383.571   411.074
## [6,]     0.000     0.000
```

```
dim(spatialCoords)
```

```
## [1] 4407    2
```

Construct Spatial Experiment

We have constructed the count matrix, colData, and spatial coordinates information, and we now can construct the spatial experiment object for further analysis.

```r
# construct spatial experiment
spe <- SpatialExperiment(
  assay = count_matrix,
  colData = colData,
  spatialCoordsNames = c("centroidX", "centroidY")
  #spatialCoords = spatialCoords
)
```

```r
spe
```

```
## class: SpatialExperiment
## dim: 113 4407
## metadata(0):
## assays(1): ''
## rownames(113): 1 2 ... 187 211
## rowData names(0):
## colnames: NULL
## colData names(6): sample_id cluster_id ... STAGE Survival_days_capped
## reducedDimNames(0):
## mainExpName: NULL
## altExpNames(0):
## spatialCoords names(2) : centroidX centroidY
## imgData names(0):
```

Save Spatial experiment object

Saving constructed spe object as RDS file.

```r
save(spe, file = "03_TNBC_SPE.rds")
```