Working with type summaries in `lute`

Sean Maden

2022-12-21

This vignette provides working examples using the `SummarizedExperimentTypes` class introduced in `lute`.

Overview

Often when working with deconvolution experiments, we need to generate and compare multiple references datasets, represented as the matrix Z in the deconvolution problem $Y = Z * P$. To manage this, it is convenient to have a devoted object class specifically for containing the type-level signals for some Z, as well as any important summary information that may be used in quality control filters or bias corrections downstream. The new class `SummarizedExperimentTypes`, and related classes such as `RangedSummarizedExperimentTypes` can help with this.

Simulating data for a `SingleCellExperiment` object

Often we calculate the reference matrix from a single-cell RNA-seq dataset contained in a `SingleCellExperiment`-type object. The function `set_from_sce()` facilitates this. It takes a `SingleCellExperiment` object as input and returns a `SummarizedExperimentTypes` object. Below, we show how to do this with some simulated data.

You can generate a `SingleCellExperiment` object containing simulated scRNAseq data using the `random_sce()` function. Calling this with defaults produces a small random dataset. You can also vary the properties of the random data in the new `sce` object (see `?random_sce`).

Make the small object `sce` as follows:

```
sce <- random_sce()
```

Type-level summaries example

Make a new `SummarizedExperimentTypes` object

Let's call `set_from_sce()` to make the object `set`, a new object of type `SummarizedExperimentTypes`:

```
set <- set_from_sce(sce, typevar = "celltype", method = "mean")
class(set)
```

```
## [1] "SummarizedExperimentTypes"
## attr(,"package")
## [1] "lute"
```

```
nrow(set)
```

```
## [1] 20
```

```
identical(rownames(sce), rownames(set))
```

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

```
ncol(set)
```

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

```
colnames(set)
```

```
## [1] "type1" "type2"
```

We can see that `set` contains the same number and ordering of genes (rows) as our random `sce` dataset, but the number of columns now correspond to the unique groups from the variable `celltype`, which are `type1` and `type2`.

Access summary rowData

We can access rowdata, or gene-level metadata, from a `SummarizedExperimentTypes` object using `rowData()`, which is the same way as for a regular `SummarizedExperiment` object.

```
rd <- rowData(set)
colnames(rd)
```

```
## [1] "type1;var" "type1;sdv" "type1;max" "type1;min" "type2;var" "type2;sdv"
## [7] "type2;max" "type2;min"
```

We can see the columns in the `set` rowdata correspond to the summary statistics of "var" (for variances), "sdv" (for standard deviations), and "min" (for minimum value), grouped by type (either "type1" or "type2").

The full rowdata object looks like:

```
knitr::kable(rd, align = "c")
```

	type1.var	type1.sdv	type1.max	type1.min	type2.var	type2.sdv	type2.max	type2.min
gene1	14.3	3.781534	15	6	7.7	2.7748874	11	4
gene2	21.3	4.615192	15	3	10.7	3.2710854	14	5
gene3	10.0	3.162278	16	8	10.3	3.2093613	13	5
gene4	11.2	3.346640	13	5	0.7	0.8366600	12	10
gene5	8.3	2.880972	15	8	13.7	3.7013511	12	3
gene6	11.3	3.361547	12	3	0.5	0.7071068	10	8
gene7	20.5	4.527693	18	7	3.5	1.8708287	10	5
gene8	3.2	1.788854	10	6	8.3	2.8809721	15	9

	type1.var	type1.sdv	type1.max	type1.min	type2.var	type2.sdv	type2.max	type2.min
gene9	17.2	4.147288	15	5	9.5	3.0822070	15	7
gene10	28.7	5.357238	22	9	13.5	3.6742346	17	8
gene11	2.3	1.516575	10	7	6.7	2.5884358	14	8
gene12	9.5	3.082207	15	7	20.7	4.5497253	17	6
gene13	6.5	2.549510	14	7	7.3	2.7018512	14	8
gene14	3.5	1.870829	13	8	5.2	2.2803509	12	6
gene15	20.0	4.472136	18	6	4.5	2.1213203	14	9
gene16	3.5	1.870829	12	7	6.2	2.4899799	14	7
gene17	5.3	2.302173	14	9	20.7	4.5497253	18	7
gene18	8.0	2.828427	14	8	9.2	3.0331502	14	8
gene19	14.7	3.834058	14	5	3.7	1.9235384	11	6
gene20	2.0	1.414214	11	8	14.3	3.7815341	12	4

Access summary colData

As with the rowdata, we access coldata from `set` using `colData()`, which is again the same as for a regular `SummarizedExperiment` object.

```
cd <- colData(set)
colnames(cd)
```

```
## [1] "type"             "num.cells"        "num.allzeroexpr"  "mean.zerocount"
## [5] "median.zerocount" "var.zerocount"    "sd.zerocount"
```

In the coldata, we see the column `"type"` which specifies the type labels. These labels correspond to each of the assay columns. We also see `"num.cells"` which is the number of cells for each type, or the total `sce` columns used to make the type summary data.

Next, column `"num.allzeroexpr"` is the number of genes for which all cells had zero expression. The last three columns `"mean.zerocount"`, `"median.zerocount"`, and `"var.zerocount"` show the mean, median, and variance in the number of cells with zero counts across genes.

The full coldata looks like:

```
knitr::kable(cd, align = "c")
```

	type	num.cells	num.allzeroexpr	mean.zerocount	median.zerocount	var.zerocount	sd.zerocount
type1	type1	5	0	0	0	0	0
type2	type2	5	0	0	0	0	0

Group-level summaries example

We often want to produce group-level summaries from scRNA-seq datasets. For instance, we may need to summarize data by donor or subject in a multi-subject experiment. Let's now add group metadata to the new `sce` objectlike so:

```
colData(sce)$donor <- c(rep("donor1", 7), rep("donor2", 3))
```

For demonstration, not all groups are represented in all types. This can be viewed with a call to the `table()` function:

```
table(sce[["donor"]], sce[["celltype"]])
```

```
##
##          type1 type2
##   donor1     5     2
##   donor2     0     3
```

We can see that the `type1` does not contain any data from `donor2`. We are ready to generate our new `SummarizedExperimentTypes` object.

Let's regenerate the `set` object as before, but specifying the group variable corresponding to donor ID as `groupvar = "donor"`.

```
set <- set_from_sce(sce, groupvar = "donor")
```

Access group summary rowdata

We can access the new `set` rowdata as above:

```
rd <- rowData(set)
colnames(rd)
```

```
##  [1] "type1;var"                 "type1;sdv"
##  [3] "type1;max"                 "type1;min"
##  [5] "type1;donor1;num.entries" "type1;donor1;mean"
##  [7] "type1;donor1;median"      "type1;donor1;var"
##  [9] "type1;donor1;sd"          "type1;donor1;numzero"
## [11] "type1;donor2;num.entries" "type1;donor2;mean"
## [13] "type1;donor2;median"      "type1;donor2;var"
## [15] "type1;donor2;sd"          "type1;donor2;numzero"
## [17] "type2;var"                 "type2;sdv"
## [19] "type2;max"                 "type2;min"
## [21] "type2;donor1;num.entries" "type2;donor1;mean"
## [23] "type2;donor1;median"      "type2;donor1;var"
## [25] "type2;donor1;sd"          "type2;donor1;numzero"
## [27] "type2;donor2;num.entries" "type2;donor2;mean"
## [29] "type2;donor2;median"      "type2;donor2;var"
## [31] "type2;donor2;sd"          "type2;donor2;numzero"
```

There's a lot more information in the new rowdata! In addition to summary statistics specific to the type (e.g. `"type1"` and `"type2"`), there are new columns for summary statistics relating to the groups by type (e.g. `"type1;donor1;..."`, `"type1;donor2;..."`, `"type2;donor1;..."`, and `"type2;donor2;..."`). For each of these type-by-group categories, we generated the gene-wise means, medians, variances, standard deviations, and number of zero-value cells/columns.

The type-by-group summary statistics are specified as an additional argument in `set_from_sce()` function called `groupstat`, which is passed to the function `sce_groupstat()` to get the rowdata and coldata summaries (see `?sce_groupstat` for details). Fewer statistics can be specified in `groupstat` in order to speed up computation.

Access group summary coldata

We can access the new `set` coldata as above:

```r
cd <- colData(set)
colnames(cd)
```

```
##  [1] "type"                             "num.cells"
##  [3] "num.allzeroexpr"                  "mean.zerocount"
##  [5] "median.zerocount"                 "var.zerocount"
##  [7] "sd.zerocount"                     "donor1;colData_means;num.entries"
##  [9] "donor1;colData_means;mean"        "donor1;colData_means;median"
## [11] "donor1;colData_means;var"         "donor1;colData_means;sd"
## [13] "donor1;colData_means;numzero"     "donor2;colData_means;num.entries"
## [15] "donor2;colData_means;mean"        "donor2;colData_means;median"
## [17] "donor2;colData_means;var"         "donor2;colData_means;sd"
## [19] "donor2;colData_means;numzero"
```

Unlike with rowdata, rows in coldata correspond to columns in the `set` assay data such that we have two rows total. This means that cells for a given group are initially collapsed by means for each gene, and then the coldata summary statistics are generated across these gene expression means.

Importantly, we generate `NA` values for group categories that aren't present in a given type. In this case, we have `NA` values for `donor2` in `type1`. We can view this now:

```r
cdf <- as.data.frame(cd)
cdf <- as.data.frame(t(cdf))
knitr::kable(cdf, align = "c")
```

	type1	type2
type	type1	type2
num.cells	5	5
num.allzeroexpr	0	0
mean.zerocount	0	0
median.zerocount	0	0
var.zerocount	0	0
sd.zerocount	0	0
donor1.colData_means.num.entries	20	20
donor1.colData_means.mean	10.17	9.70
donor1.colData_means.median	10.10	10.25
donor1.colData_means.var	1.356947	3.984211
donor1.colData_means.sd	1.164881	1.996049
donor1.colData_means.numzero	0	0
donor2.colData_means.num.entries	NA	20
donor2.colData_means.mean	NA	10.25
donor2.colData_means.median	NA	10.66667
donor2.colData_means.var	NA	2.875731
donor2.colData_means.sd	NA	1.695798
donor2.colData_means.numzero	NA	0

Conclusions

This vignette showed how to simulate a small `SingleCellExperiment` object and produce a `SummarizedExperimentTypes` object with the function `set_from_sce()`.