scImpute: a statistical method for accurate and robust imputation of scRNA-seq data

Wei Vivian Li, Jingyi Jessica Li

2017-05-24

The emerging single cell RNA sequencing (scRNA-seq) technologies enable the investigation of transcriptomic landscape at single-cell resolution. However, scRNA-seq analysis is complicated by the excess of zero or near zero counts in the data, which are the so-called dropouts due to low amounts of mRNA within each individual cell. Consequently, downstream analysis of scRNA-seq woule be severely biased if the dropout events are not properly corrected. `scImpute` is developed to accurately and efficiently impute the dropout values in scRNA-seq data.

`scImpute` can be applied to raw data count before the users perform downstream analyses such as

- dimension reduction of scRNA-seq data
- normalization of scRNA-seq data
- clustering of cell populations
- differential gene expression analysis
- time-series analysis of gene expression dynamics

Quick start

`scImpute` can be easily incorporated into existing pipeline of scRNA-seq analysis. Its only input is the raw count matrix with rows representing genes and columns representing cells. It will output an imputed count matrix with the same dimension. In the simplest case, the imputation task can be done with one single function `scimpute`:

```r
scimpute(# full path to raw count matrix
         count_path = system.file("extdata", "raw_count.csv", package = "scImpute"),
         infile = "csv",           # format of input file
         outfile = "csv",          # format of output file
         out_dir = "./",           # full path to output directory
         drop_thre = 0.5,          # threshold set on dropout probability
         ncores = 10)              # number of cores used in parallel computation
```

This function will create a new file `scImpute_count.csv` in `out_dir` to store the imputed count matrix.

Step-by-step description

The input file can be either a `.csv` file or `.txt` file. In both cases, the first column should give the gene names and the first row should give the cell names. We use the example files in the package as illustration. If the raw counts are stored in a `.csv` file, and we also hope to output the imputed matrix into a `.csv` file, then specify this information with

```r
# full path of the input file
count_path = system.file("extdata", "raw_count.csv", package = "scImpute")
infile = "csv"
outfile = "csv"
```

Similarly, If the raw counts are stored in a `.txt` file, and we also hope to output the imputed matrix into a `.txt` file, then specify this information with

```r
# full path of the input file
count_path = system.file("extdata", "raw_count.txt", package = "scImpute")
infile = "txt"
outfile = "txt"
```

Next, we need to set up the directory to store all the temporary and final outputs:

```r
# a '/' sign is necessary at the end of the path
out_dir = "~/output/"
```

We highly recommend using parallel computing with `scImpute`, which will significantly reduce the computation time. Suppose we would like to use 10 cores, then we can run the `scImpute` function with `ncores = 10`.

The only statistical parameter needed by `scImpute` is `drop_thre`. Only the values that have dropout probability larger than `drop_thre` are imputed by by `scImpute`. Without any preference, we can set `drop_thre = 0.5`. We will show later that it is very quick and convenient to re-run `scImpute` with a different `drop_thre`.

Now to get the imputed matrix, all we need is the main `scimpute` function

```r
drop_thre = 0.5
ncores = 10
scimpute(count_path, infile, outfile, out_dir, drop_thre, ncores)
```

If `outfile = "csv"`, this function will create a new file `scimpute_count.csv` in `out_dir` to store the imputed count matrix; if `outfile = "txt"`, this function will create a new file `scimpute_count.txt` in `out_dir`.

Re-apply scImpute with a different `drop_thre`

The most time-consuming part in `scImpute` is to estimate a mixture model for each gene. This step does not depend on `drop_thre` and only needed to be done once for each data set. Therefore, if users have already ran `scimpute` on their data sets for at least once, then we can use `scimpute_quick` to skip the time-consuming step and re-apply `scImpute` to the same data with a different `drop_thre`. The arguments in `scimpute_quick` are basically the same as in `scimpute`. The obly thing to note is that `out_dir` should be the same as the one used in previous runs, so that the function is able to locate intermediate files generated beforehand. All we need is

```r
scimpute_quick(
        count_path = system.file("extdata", "raw_count.csv", package = "scImpute"),
        infile = "csv",
        outfile = "csv",
        out_dir = out_dir,        # should be the same as used in scimpute()
        drop_thre = 0.3,          # a different threshold
        ncores = 10)
```

Apply scImpute with cell type information

Sometimes users may have the cell type (or subpopulation) information of the single cells and `scimpute` can take advantage of this information to impute among each cell type. To do this, we need a character vector `labels` specifying the cell type of each column in the raw count matrix. In other words, the length of `labels` equals the number of cells and the order of elements in `labels` should match the order of columns in the raw count matrix. Then we just need to specify `celltype = TRUE` in `scimpute` (default is `FALSE`) and specify the `labels` argument. The same rules also apply to `scimpute_quick`.

```
labels = readRDS(system.file("extdata", "labels.rds", package = "scImpute"))
labels[1:5]
> [1] "c1" "c1" "c1" "c2" "c2"

scimpute(count_path,
         infile = "csv",
         outfile = "csv",
         out_dir = out_dir,
         drop_thre = 0.5,
         celltype = TRUE,
         labels = labels,
         ncores = 10)
```

Note that we do not recommend using `labels` when the cell classes represent completely different tissues or cell types. In that case, it is suggested to apply `scimpute` to each data set separately (one `.csv` or `.txt` for one file). It would be better to use `labels` with similar cell subpopulations or developmental stages.

How to save computation time with `scImpute`

`scImpute` will benefit a lot from parallel computation, and each processor does not require heavy memory cost. For example, `scimpute` completes computation in 30 minutes when applied to a dataset with 10,000 genes and 100 cells, running with 10 cores. The memory requirement for this data set is around 2G. The running time mostly depends on

- number of processors (`ncores`)
- number of genes in the scRNA-seq data

Again, we note that for any data sets, `scimpute` should only be applied once to save time. Rerunning with different parameters should be applied with `scimpute_quick`.