BSTA 670 - Final Project 2019

Justin Lakkis

5/6/2019

mnnsim (Mutual Nearest Neighbors Simulation)

This package provides a utility to easily and conveniently perform simulations that compare the performance of various batch correction methods for single-cell data. The methods are Mutual Nearest Neighbors (https://www.nature.com/articles/nbt.4091), limma (https://www.ncbi.nlm.nih.gov/pubmed/25605792), and ComBat (https://academic.oup.com/biostatistics/article/8/1/118/252073). For each simulation replicate, we generate data and then cluster the data 4 times: once raw, once after mnn batch correction, once after limma batch correction, and once after ComBat batch correction. Silhoutte coefficients are then used to evaluate the quality of the clustering.

For each simulation replicate, three batches are simulated. A silhoutte score is generated for each cell. To summarize this information, the mean silhoutte score for each batch is saved, as well as the mean silhoutte score for all cells, giving four summary quantities per replicate. At the end of the simulation, the mean and standard deviation of each of these summary quantities over all replicates is reported.

Simulation Tutorial

To demonstrate use of the package, let's step through the workflow of a simulation.

Package Setup

The package can be installed with the following commands. If some dependencies fail to install, the user can change dependency.manualinstall variable to TRUE so that the dependencies which usually cause installation difficulties will be manually installed.

```
dependency.manualinstall=FALSE

if(dependency.manualinstall) {
    if (!requireNamespace("BiocManager", quietly = TRUE))
        install.packages("BiocManager")

    BiocManager::install("BiocSingular")
    BiocManager::install("genefilter")
    BiocManager::install("batchelor")
}

if("mnnsim" %in% rownames(installed.packages()) == FALSE) {
    if("devtools" %in% rownames(installed.packages()) == FALSE)
        install.packages("devtools")
```

```
    devtools::install_github('jlakkis/mnnsim')
}
```

```
##
##
   checking for file ‘/private/var/folders/qj/r5s1pq114m31ql_b5g752jx80000gn/T/RtmpZNe1RH/remot

v  checking for file ‘/private/var/folders/qj/r5s1pq114m31ql_b5g752jx80000gn/T/RtmpZNe1RH/remot
##

-  preparing ‘mnnsim’:
##

   checking DESCRIPTION meta-information ...

v  checking DESCRIPTION meta-information
##

-  checking for LF line-endings in source and make files and shell scripts
##

-  checking for empty or unneeded directories
## -  looking to see if a ‘data/datalist’ file should be added
##

     NB: this package now depends on R (>= 3.5.0)
##

     WARNING: Added dependency on R >= 3.5.0 because serialized objects in  serialize/load vers
##

-  building 'mnnsim_1.0.0.tar.gz'
##


##
```

```
library(mnnsim)
```

Choosing Simulation parameters

The first step is to define the parameters upon which we want to base our simulation. We can simulate replicates from multiple combinations of parameters and then compare the performance of various batch correction methods for data simulated under the different combinations of generative parameters.

Suppose we want to test n combinations of parameters. Then each parameter should be a list of

length n, whose ith element is the value of that parameter to be used for the ith set of simulation parameters. For a description of each of these paramters, please consult the documentation of the simstudy function via help("simstudy"). The exception is the seed, which hould always be a single integer, not a list.

```
parameternames=list('Original', 'Smaller Differences', 'More Genes')
nsims=list(5,5,5)
seed=0
cellcounts=list(500,500,500)
genecounts=list(100,100,5000)
xmeans=list(c(0,5,5),c(0,2,2),c(0,5,5))
xsdss=list(c(1,0.1,1),c(1,0.1,1),c(1,0.1,1))
ymeans=list(c(5,5,0),c(2,2,0),c(5,5,0))
ysdss=list(c(1,0.1,1),c(1,0.1,1),c(1,0.1,1))
propsbatch1=list(c(0.3,0.5,0.2),c(0.3,0.5,0.2),c(0.3,0.5,0.2))
propsbatch2=list(c(0.65,0.3,0.05),c(0.65,0.3,0.05),c(0.65,0.3,0.05))
```

Preliminary Simulation

A major challenge for a simulation study is determining the appropriate number of simulation replicates. One way to do this is to perform a preliminary simulation study. We can then compute the standard deviation of each quantity we simulate per replicate over the replicates of this preliminary simulation. The standard deviation of the mean of each quantity over n simulation replicates, which is the summary metric we are interested in, is then SD(quantiity)/sqrt(n).

Recall that there are four quantities for each replicate: mean silhoutte coefficient over all cells, and over cells from each of three batches. By setting tol=0.01, we choose n for each simulation parameter combination such that standard deviation of the mean of each of the four quantities over n simulation replicates is no more than 0.01.

We save the new sample sizes in an object called nsims, and we examine the sample sizes for each parameter combination.

In this tutorial, all simulations are done by parallelizing with four cores. The number of cores can be changed via the mycore argument.

```
prelimstudy(
    tol=0.01,parameternames=parameternames,
    nsims=nsims,seed=seed,
    cellcounts=cellcounts,
    genecounts=genecounts,
    xmeans=xmeans,xsdss=xsdss,ymeans=ymeans,
    ysdss=ysdss,propsbatch1=propsbatch1,
    propsbatch2=propsbatch2,mycore=2
)

nsims
```

Main Simulation Study

Now, we can proceed with the main simulation study by running the simstudy function. The arguments are exactly the same as for the prelimstudy function, but now we used the sample sizes estimated from the prelimstudy function, and we change the seed.

```
seed=215

mystudy=simstudy(
    parameternames=parameternames,
    nsims=nsims,seed=seed,
    cellcounts=cellcounts,
    genecounts=genecounts,
    xmeans=xmeans,xsdss=xsdss,ymeans=ymeans,
    ysdss=ysdss,propsbatch1=propsbatch1,
    propsbatch2=propsbatch2,mycore=2
)
```

Summarizing Results

The simstudy function saves the four quantities (mean of silhoutte scores over all cells, oer cells from each of three batches) for every replicate. To summarize the information, we can use the simresults function, which will report the monte carlo mean and standard deviation for each of these four quantities, and for each parameter combination. This concludes the simulation workflow.

```
simresults(mystudy)
```

Visualization

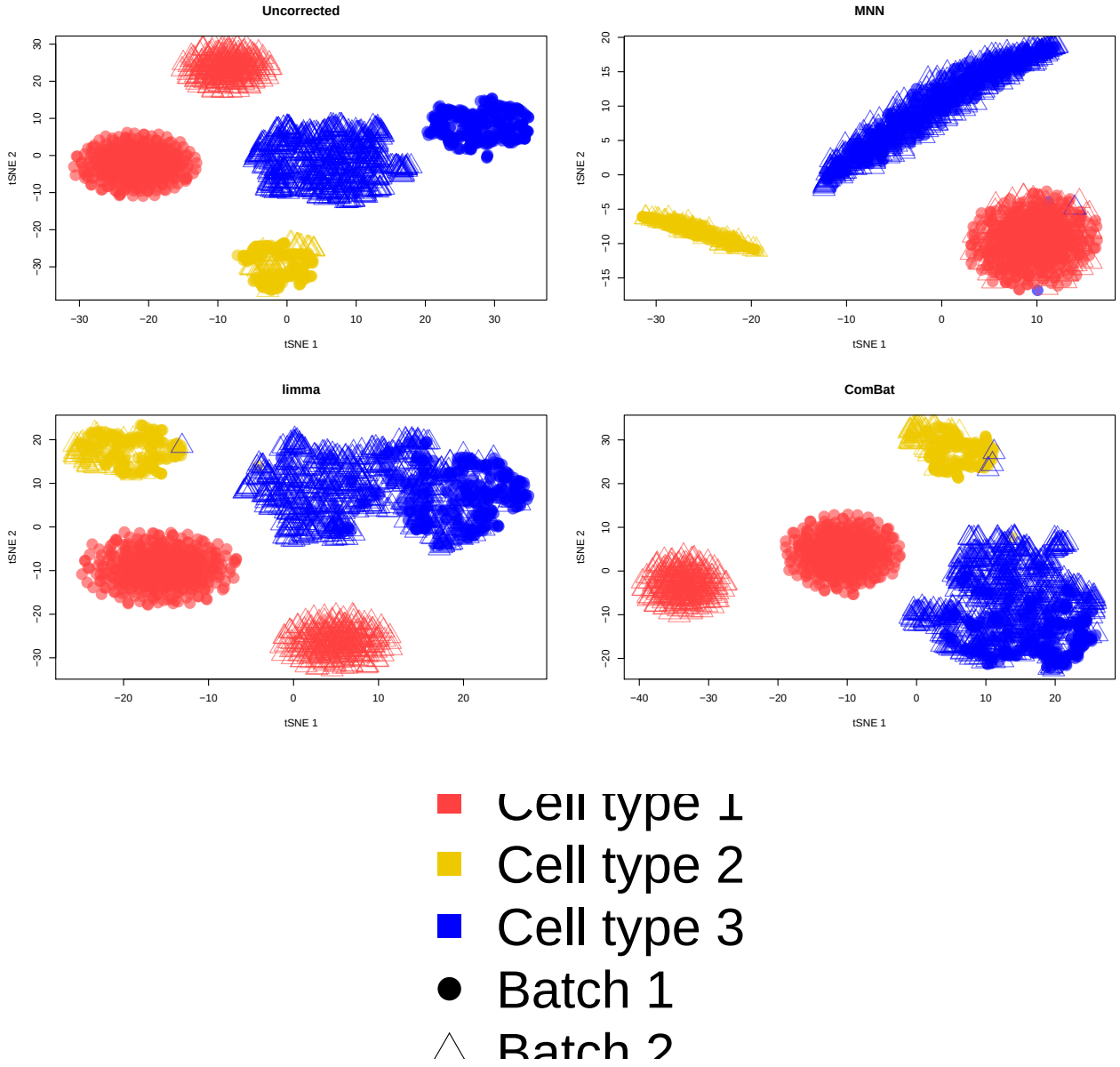
We can also visualize the effectiveness of various batch correction methods by simulating data, batch correcting it with each method, and then clustering the corrected data with tSNE and plotting the clustered data. This can be accomplished with the makeTSNE function.

Note a limitation: this function can only be run for one "replicate" at a time, and only for one combination of parameters. makeTSNE does not run a simulation: it should only be done to give a

```
seed=12345
ncells=1000
ngenes=300
xmus=c(0,5,5)
ymus=c(5,5,0)
xsds=ysds=c(1,0.1,1)
prop1=c(0.3,0.5,0.2)
prop2=c(0.65,0.3,0.05)

makeTSNE(
    seed,ncells,ngenes,
    xmus,xsds,
```

```
    ymus,ysds,
    prop1,prop2
)
```




```
seed=2150

nstudy=10
simtimes=matrix(nrow = nstudy, ncol = 2)
colnames(simtimes)=c('1 Core', 'Multi-Core')
ncore=4

parameternames=list('Set 1', 'Set 2')
nsims=list(30,30)
cellcounts=list(500,500)
```

```
genecounts=list(100,100)
xmeans=list(c(0,5,5),c(0,2,2))
xsdss=list(c(1,0.1,1),c(1,0.1,1))
ymeans=list(c(5,5,0),c(2,2,0))
ysdss=list(c(1,0.1,1),c(1,0.1,1))
propsbatch1=list(c(0.3,0.5,0.2),c(0.3,0.5,0.2))
propsbatch2=list(c(0.65,0.3,0.05),c(0.65,0.3,0.05))

for(i in c(1:nstudy)) {
    seed=abs(round(rnorm(1,10000,1000)))

    tictoc::tic()
    mystudy=simstudy(
        parameternames=parameternames,
        nsims=nsims,seed=seed,
        cellcounts=cellcounts,
        genecounts=genecounts,
        xmeans=xmeans,xsdss=xsdss,ymeans=ymeans,
        ysdss=ysdss,propsbatch1=propsbatch1,
        propsbatch2=propsbatch2,mycore=1
    )
    info=tictoc::toc()
    simtimes[i,1]=info$toc-info$tic

    tictoc::tic()
    mystudy=simstudy(
        parameternames=parameternames,
        nsims=nsims,seed=seed,
        cellcounts=cellcounts,
        genecounts=genecounts,
        xmeans=xmeans,xsdss=xsdss,ymeans=ymeans,
        ysdss=ysdss,propsbatch1=propsbatch1,
        propsbatch2=propsbatch2,mycore=ncore
    )
    info=tictoc::toc()
    simtimes[i,2]=info$toc-info$tic
}

colMeans(simtimes)
```