HarmonizR 1.1 Software Instructions

Dependencies

The HarmonizR package depends on a couple of different R packages and of course the R environment for statistical computing itself. The necessary imports are included within the DESCRIPTION file, which is uploaded together with the rest of the package on GitHub and must not be changed by the user. The dependencies and imports are as follows:

Dependency	Required Version
R	>= 4.2.0
sva	>= 3.36.0
doParallel	>= 1.0.16
foreach	>= 1.5.1
janitor	>= 2.1.0
plyr	>= 1.8.6
seriation	>= 1.4.2

HarmonizR wraps around the ComBat and limma implementations from the 'sva' package.

All dependencies get installed automatically upon installing the HarmonizR package. It is recommended to accept the installation of the newest updated versions of these dependencies while installing the HarmonizR package. For the HarmonizR algorithm to function, these dependencies must be met. In case that the automatic installation of the dependencies does not work, please try installing them manually and separately under consideration of the minimum required version using `install.packages()`.

If the installation of the 'sva' package specifically does not work, it can also be installed separately. Here, Bioconductor needs to be used. It is easiest to use the BiocManager for this. Make sure to be within the R environment. The installed 'sva' package should have the version 3.36.0 or newer:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("sva")
```

Installation

The HarmonizR implementation is 100 % written in the programming language R. The easiest way to install it is using the package ‘devtools’ that can be installed from CRAN via `install.packages("devtools")` while in the R environment. For further information refer to the ‘devtools’ documentation in R.

Installation from Github Repository

The HarmonizR package (https://github.com/HSU-HPC/HarmonizR_v1.1 leads to the package as well as example data) can be installed from GitHub via the command `devtools::install_github("HSU-HPC/HarmonizR_v1.1")` while in the R software environment. In order to work correctly, R has to be installed on version >= 4.2.0 as well as the ‘devtools’ package.

Installation from HarmonizR.zip file

Please make sure to have ‘devtools’ installed. Download the code via the green Code button (https://github.com/HSU-HPC/HarmonizR_v1.1). Unzip the downloaded .zip file. The HarmonizR package is the folder called HarmonizR, which was within the .zip file. While in the R environment and while in the folder of the downloaded package, enter in the command line: `devtools::install()` to install the package.

Input and Output

The HarmonizR package expects two files to be passed as an argument: A tab-delimited (.tsv) file containing the raw, unadjusted data and a comma-separated (.csv) file describing the batch layout. We provide one correctly formatted example for both file types (‘murine_medulloblastoma_data.tsv’ and ‘murine_medulloblastoma_description.csv’). See the inst/ directory within the package.

After calling the `harmonizR()` function, the overall, most basic workflow of the algorithm looks like the following (Figure 1):

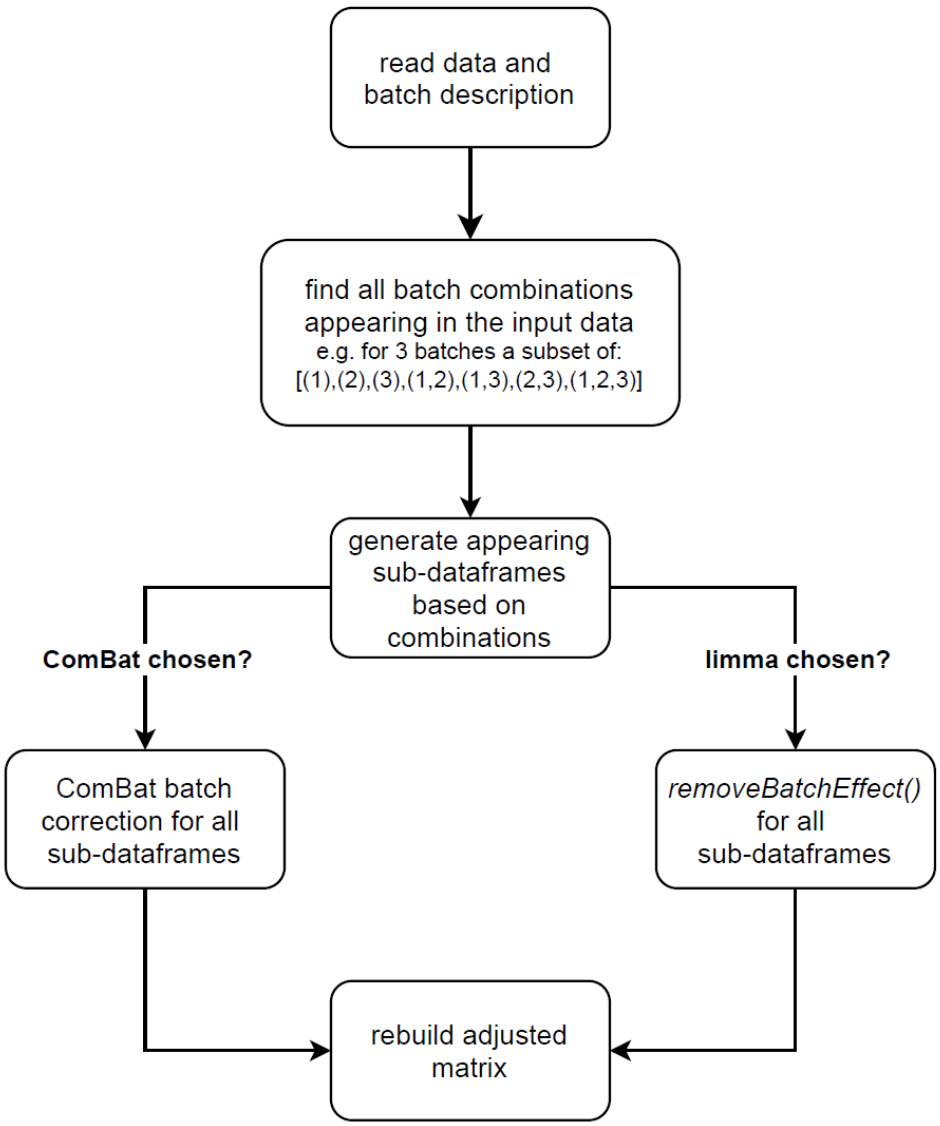

Figure 1: Overall, simplified workflow of the HarmonizR algorithm after calling the harmonizR() function.

The file with the raw data is expected to contain the samples in columns and the features in rows. The very first column should contain the feature IDs, whereas the first row should contain the sample IDs. Missing values should either be missing spaces between the tab delimiters or can alternatively be indicated via a string such as *NaN* like in the example file. Be aware, that zeroes (0) will not be treated as missing values.

An example (for a real example see 'murine_medulloblastoma_data.tsv'):

Protein.ID	Sample1	Sample2	Sample3	...
Protein1	1130	2231	718	...
Protein2	563	NaN	2112	...
Protein3	921	7997	901	
...	...	...		

Secondly, each sample has to be assigned to a batch. For this, a 'description' .csv (comma separated values) file is required, which holds the sample ID in column 1, the overall number of the sample in column 2 and the batch number it is assigned to in column 3, very much like what the default ComBat or limma algorithms would require as input. It is important to have the samples ordered like they are within the .tsv file and have all samples from batch 1 listed before listing batch 2 and so on.

An example (for a real example see 'murine_medulloblastoma_description.csv'):

ID,sample,batch

Sample1,1,1

Sample2,2,1

Sample3,3,2

…

The HarmonizR algorithm will write the resulting 'harmonized' matrix into a .tsv file directly. The resulting file (by default called 'cured_data.tsv') will be written to the directory the R file with the harmonizR function is executed in. The resulting .tsv file will closely mirror the input file, however, the batch effect within the data will be removed and the proteins will be reordered based on the missing values found during the computation.

Parameters and example usage

Load the HarmonizR package:

```
library(HarmonizR)
```

HarmonizR expects 2 mandatory and a total of nine optional arguments. First, the mandatory ones:

- data_as_input
- description_as_input

The first argument, 'data_as_input', is the *path* to the raw data, the second argument, 'description_as_input', is the *path* to the description file. Both input files can be given via their file path and do not have to be read in separately by hand. This method is recommended, as it ensures correct operation if the notes regarding Input above are followed. Alternatively, both parameters can be passed as dataframes or matrices as long as they are fitting the expected input layout.

Next will be four optional arguments found also within the already published HarmonizR:

- algorithm
- ComBat_mode
- plot
- output_file

The first optional argument is the algorithm of choice. ComBat will be used by default, but using the parameter 'algorithm', either "ComBat" or "limma" can be chosen for data adjustment. ComBat serves as the default. The second optional argument is the ComBat-mode. This parameter is only valid once ComBat is chosen for the adjustment.

The ComBat mode is abbreviated for simplicity by:

ComBat Mode	Corresponding ComBat Arguments
1 (default)	par.prior = TRUE, mean.only = FALSE
2	par.prior = TRUE, mean.only = TRUE
3	par.prior = FALSE, mean.only = FALSE
4	par.prior = FALSE, mean.only = TRUE

Please refer to the ComBat documentation for further details.

The third optional parameter is 'plot'. 'plot' can be set to either "samplemeans", "featuremeans" or "CV" and will show a boxplot with a box for each batch depicting the chosen method. This plot will also be saved to a .pdf. This will be either the mean for all samples, the mean for all features or the coefficient of variation. There will be a separate plot for the original, unaltered input dataset and for the ComBat/limma adjusted dataset. By default, this parameter is turned off. Since a log transformation is assumed, this will also be accounted for before plotting. Trying to plot data that has not been log transformed prior may lead to an unplottable result. The fourth optional parameter is 'output_file'. Setting this parameter will grant the user the ability to choose the name of their output .tsv file. Also, a path can be set. A string is expected as input. This parameter will default to "cured_data", yielding a file called 'cured_data.tsv'. (It may also be set to *FALSE* to suppress file creation.)

Further, the five new parameters will be explained briefly one-by-one:

- sort

'sort' takes one of three available sorting algorithms as input. Either "sparcity_sort", for a sparcity-based sorting, "seriation_sort", using the 'seriation' package and "jaccard_sort", using a Jaccard-index-based sorting approach. Sorting happens prior to the adjustment and may change the way blocking is executed on the data. Sorting does not yield any benefit when the 'block' parameter is unused.

- block

'block' takes an integer as input which dictates, how many batches should be packed together during matrix dissection. This parameter may greatly reduce the amount of sub-dataframes produced and therefore decrease the algorithm's runtime.

- verbosity

'verbosity' takes an integer with the lowest accepted integer being 0. The higher the number, the more feedback will the HarmonizR provide in the command line. A verbosity of 1 is the

default and should be sufficient. 0 or “mute” will prevent HarmonizR from showing anything in the command line.

- cores

‘cores’ gives the user the ability to control the number of cores used by their machine during HarmonizR execution. By default, all available cores will be used.

- ur

‘ur’, short for unique removal, can be set to either TRUE or FALSE to toggle the newly implemented removal of unique combinations for increased feature rescue. By default, this feature is applied, and it is strongly suggested to not turn it off. The parameter is available for result reproducibility only.

Call the ‘harmonizR’ function under usage of the provided example files:

```
harmonizR(“murine_medulloblastoma_data.tsv”,
“murine_medulloblastoma_description.csv”)
```

A call containing all optional parameters might for example look like this:

```
harmonizR(“murine_medulloblastoma_data.tsv”,
“murine_medulloblastoma_description.csv”, algorithm = “ComBat”, ComBat_mode = 1,
plot = “samplemeans”, output_file = “result_file”, sort = “sparsity_sort”, block = 2, verbosity
= 2, cores = 4, ur = TRUE)
```

Here we assume that the input files are in the same folder as the executed .r file. It is safest, however, to include the complete path. The result will be saved in ‘result_file.tsv’. The resulting plot looks like this (Figure 2):

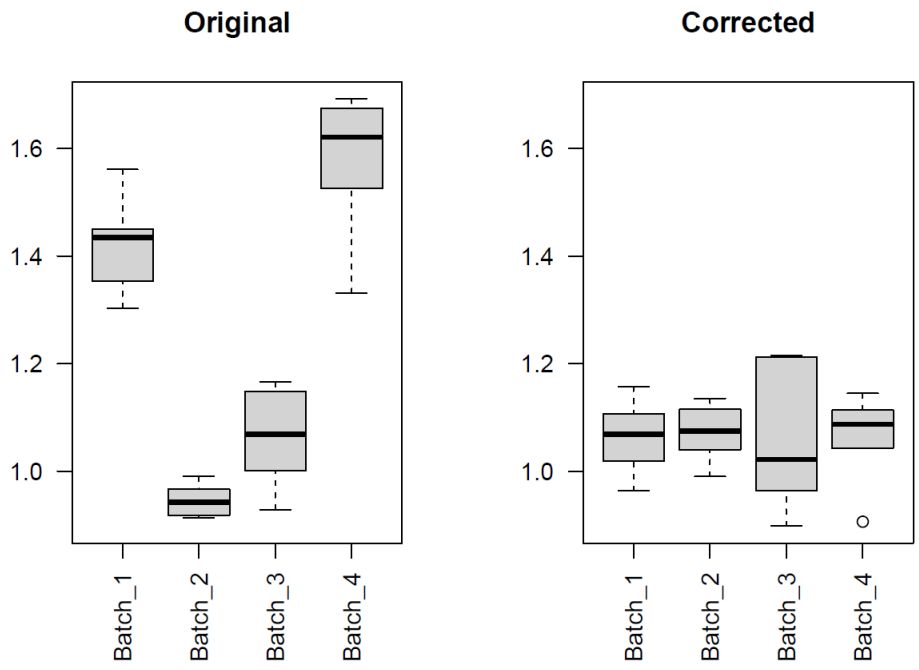

Figure 2: Example output plot when executing the harmonizR() function with the parameters shown above. The provided and referenced example dataset and description have been used to create the plot showing the CV difference for each batch for the original and the adjusted dataset.

While the HarmonizR algorithm will write the results directly to a .tsv file (in this example ‘result_file.tsv’ due to the set ‘output_file’ parameter above), it also directly returns the cured dataframe for further use in the user’s script.