Package 'RUVIIIC'

April 23, 2021

Type Package

Title RUV-III-C

Version 1.0.19

Date 2021-04-14

Author Rohan Shah [aut, cre],
Sean Peters [ctb],
Qing Zhong [ctb],
Children's Medical Research Institute [cph]

Maintainer Rohan Shah <rshah@cmri.org.au>

Description Variations of Remove Unwanted Variation-III (RUV-III) known as RUV-III-C (RUV-III Complete). RUV-
III performs normalisation using negative control variables and replication. RUV-III-
C extends this method to cases where the data contains missing values, by applying RUV-
III to complete subsets of the data. Originally designed for SWATH-MS proteomics datasets. Poulos et al. (2020) <doi:10.1038/s41467-020-17641-3>.

License MIT + file LICENSE

Encoding UTF-8

LazyLoad yes

LazyData yes

SystemRequirements C++11

LinkingTo Rcpp, RcppEigen, RSpectra, RcppProgress

Suggests ruv, testthat

Depends R (>= 3.5), RSpectra, progress

Collate 'RcppExports.R' 'RUVIII_C.R' 'RUVIII_C_Varying.R' 'data.R' 'roxygen.R' 'threads.R'

RoxygenNote 7.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-04-23 15:00:09 UTC

R topics documented:

`omp_set_num_threads` *Set or get the number of OpenMP threads*

Description

Set or get the number of threads used by OpenMP for parallel regions.

Usage

```
omp_set_num_threads(num)

omp_get_num_threads()
```

Arguments

`num`	The desired number of threads

Details

These functions mostly exist because of the CRAN check, which only allows 1 - 2 threads. As there is a per-thread memory requirement for `RUVIII_C` and `RUVIII_C_Varying`, these functions can also be useful for controlling memory usage. However in that case, setting environment variable OMP_NUM_THREADS also works.

Value

In the case of `omp_get_num_threads`, the current number of threads.

peptideData *Peptide data from the cross-lab study of Collins et al (2017)*

Description

A dataset containing data on around 1,000 peptides from the cross-lab study of Collins et al (2017), originally published in Nature Communications. The data consists of a data.frame named `peptideData`, and a character vector sisPeptides, listing the peptides that should be analysed.

Usage

```
data(crossLab)
```

Format

A data.frame containing data on around 1,000 peptides, with peptides that were not detected in a mass-spectrometry acquisition coded as NA. All columns are peptides, except for `filename`, `site`, `mixture`, `grouping` and `Date`. The peptides include around 1,000 background peptides, and the stable isotope-labelled standard (SIS) peptides which have intensities that change across mass-spectrometry acquisitions.

RUVIII_C *RUV-III-C*

Description

Apply RUV-III-C, a variation of RUV-III that only uses non-missing values

Usage

```
RUVIII_C(
  k,
  Y,
  M,
  toCorrect,
  controls,
  withExtra = FALSE,
  withW = FALSE,
  withAlpha = FALSE,
  version = "CPP",
  progress = TRUE,
  ...
)
```

Arguments

k	The number of factors of unwanted variation to remove
Y	The input data matrix. Must be a matrix, not a data.frame. It should contain missing (NA) values, rather than zeros. No additional transformation is applied to the input data.
M	The design matrix containing information about technical replicates. It should not contain an intercept term!
toCorrect	The names of the variables to correct using RUV-III-C
controls	The names of the control variables which are known to be constant across the observations
withExtra	Should we generate extra information?
withW	Should we generate the matrices W giving information about the unwanted factors, for every peptide?
withAlpha	Should we generate, per-peptide, the matrix alpha giving the effects of the unwanted factors?
version	The version of the underlying code to use. Must be either "CPP" or "R"
progress	Should a progress bar be displayed?
...	Other arguments for the prototype R code. Supported values are filename for a checkpoint file, and batchSize for the frequency with which the checkpoint file is written.

Details

RUV-III is a sophisticated method for removing unwanted variation. The key difficulty in removing unwanted variation is distinguishing wanted from unwanted variation. RUV-III solves this by relying on technical replication, and a list of variables (known as negative control variables) which are known a priori to be constant across all observations. Any variation in the negative control variables across the dataset is (by assumption) unwanted. So we can distinguish wanted from unwanted variation, and therefore estimate the unwanted variation and remove it.

One problem with this approach is the presence of "missing" or zero values in certain application domains. For example, in proteomics it will sometimes be the case that a protein or peptide is not detected in a specific technical replicate of a sample, for purely technical reasons relating to data collection. These missing values are often not related to censoring or the limit of detection. Similar problems occur in metabolomics and single-cell transcriptomics. In all these cases, the metabolite, gene or peptide will be recorded as a zero in the data matrix. Where this type of variation occurs between technical replicates (e.g. one records a zero value and one records a non-zero value) is not correctable.

Regardless of the reason for these zeros, and whether they are accurate or not, zero values are not affected by technical variation, which breaks an assumption of the RUV-III model. In the case that a zero value is incorrect, more serious problems occur. The discrepancies between a pair of technical replicates due to zero values will appear to be much larger than the discrepancies due to other (correctable) technical factors. RUV-III will attempt to correct for the larger (uncorrectable) discrepancy, and ignore the correctable technical factors.

RUV-III-C is a variation of RUV-III that attempts to solve this problem, by applying RUV-III separately to every variable. If variable X is being corrected, we take the rows of the data matrix for

which X is non-missing. RUV-III is then applied, and the corrected values of X is retained. The corrected values of all other variables are discarded. Note that when we take a subset of the rows of the data matrix, other columns besides X will still have missing values. These values are replaced with zero in order to apply RUV-III. No additional transformation is applied to the input data matrix. If normalization should be applied on the log-scale, then logged data must be input.

There are two implementations of this function, the preferred C++ version and the original protoype R code. Select which version using the `version` argument, which must be either "CPP" or "R"

Value

If withExtra = FALSE, returns a matrix. If withExtra = TRUE, returns a list with entries named `newY`, `residualDimensions` and `W`.

Examples

```
data(crossLab)
#Design matrix containing information about which runs are technical replicates of each other.
#In this case, random pairings of mass-spec runs analysing the same sample, at different sites.
#Note that we specify no intercept term!
M <- model.matrix(~ grouping - 1, data = peptideData)
#Get out the list of peptides, both HEK (control) and peptides of interest.
peptides <- setdiff(colnames(peptideData), c("filename", "site", "mixture", "Date", "grouping"))
#Reduce the data matrix to only the peptide data
onlyPeptideData <- data.matrix(peptideData[, peptides])
#All the human peptides are potential controls. That is, everything that's not an SIS peptides.
potentialControls <- setdiff(peptides, sisPeptides)
#But we want to use controls that are always found
potentialControlsAlwaysFound <- names(which(apply(onlyPeptideData[, potentialControls], 2,
    function(x) sum(is.na(x))) == 0))
#Actually run correction
#Set number of threads for CRAN
try(RUVIIIC::omp_set_num_threads(2L), silent=TRUE)
results <- RUVIII_C(k = 11, Y = log10(onlyPeptideData), M = M, toCorrect =
    colnames(onlyPeptideData), controls = potentialControlsAlwaysFound)
```

`RUVIII_C_Varying` *RUV-III-C, varying controls*

Description

Apply RUV-III-C, a variation of RUV-III that only uses non-missing values

Usage

```
RUVIII_C_Varying(
  k,
  Y,
  M,
  toCorrect,
```

```
  potentialControls,
  withExtra = FALSE,
  withW = FALSE,
  withAlpha = FALSE,
  version = "CPP",
  progress = TRUE,
  ...
)
```

Arguments

k	The number of factors of unwanted variation to remove
Y	The input data matrix. Must be a matrix, not a data.frame. It should contain missing (NA) values, rather than zeros.
M	The design matrix containing information about technical replicates. It should not contain an intercept term!
toCorrect	The names of the variables to correct using RUV-III-C
potentialControls	The names of the control variables which are known to be constant across the observations
withExtra	Should we generate extra information?
withW	Should we generate the matrices W giving information about the unwanted factors, for every peptide?
withAlpha	Should we generate, per-peptide, the matrix alpha giving the effects of the unwanted factors?
version	The version of the underlying code to use. Must be either "CPP" or "R"
progress	Should a progress bar be displayed?
...	Other arguments for the prototype R code. Supported values are `filename` for a checkpoint file, and `batchSize` for the frequency with which the checkpoint file is written.

Details

See the documentation of RUVIII_C for more information about the RUV-III-C method. This function is identical, except in this case the set of negative control variables actually used varies depending on the target variable to be normalized. Instead of putting in a list of negative control variables, the user specifies a list of potential negatve control variables.

When normalizing variable X, the algorithm begins by selecting the rows of the data matrix for which X is non-missing. Out of the potential negative control peptides, it selects those that are always non-missing across the selected subset. The standard version of RUV-III is then applied to the subset, similar to RUVIII_C.

There are two implementations of this function, one in C++ and one in R. Select which version using the `version` argument, which must be either "CPP" or "R"

Value

If withExtra = FALSE, returns a matrix. If withExtra = TRUE, returns a list with entries named `newY`, `residualDimensions` and `W`.

Examples

```
data(crossLab)
#Design matrix containing information about which runs are technical replicates of each other.
#In this case, random pairings of mass-spec runs analysing the same sample, at different sites.
#Note that we specify no intercept term!
M <- model.matrix(~ grouping - 1, data = peptideData)
#Get out the list of peptides, both HEK (control) and peptides of interest.
peptides <- setdiff(colnames(peptideData), c("filename", "site", "mixture", "Date", "grouping"))
#Reduce the data matrix to only the peptide data
onlyPeptideData <- data.matrix(peptideData[, peptides])
#All the human peptides are potential controls. That is, everything that's not an SIS peptides.
potentialControls <- setdiff(peptides, sisPeptides)
#But we want to use controls that are *often* found
potentialControlsOftenFound <- names(which(apply(onlyPeptideData[, potentialControls], 2,
    function(x) sum(is.na(x))) <= 10))
#Set number of threads for CRAN
try(RUVIIIC::omp_set_num_threads(2L), silent=TRUE)
#Actually run correction
results <- RUVIII_C_Varying(k = 11, Y = log10(onlyPeptideData), M = M, toCorrect =
    colnames(onlyPeptideData), potentialControls = potentialControlsOftenFound)
```

RUVIII_C_Varying_residual_dimension

Compute amount of replication, assuming varying controls

Description

Compute amount of replication, assuming varying controls

Usage

```
RUVIII_C_Varying_residual_dimension(Y, M, potentialControls)
```

Arguments

`Y`	The input data matrix. Must be a matrix, not a data.frame. It should contain missing (NA) values, rather than zeros.
`M`	The design matrix containing information about technical replicates. It should not contain an intercept term!
`potentialControls`	The names of the control variables which are known to be constant across the observations

Details

When applying RUV-III-C with varying controls, the amount of replication available for normalisation depends on the variable being normalised. The amount of replication is measured as the number of technical replicates in which a variable is measured (rows of the design matrix) minus the number of biological samples (columns of the design matrix) in which a variable is measured. This value is useful as a filtering criteria, because variables with a low amount of replication will be poorly normalised by the RUV family of methods.

Value

A vector of integers, one per column of Y.

Examples

```
data(crossLab)
#Design matrix containing information about which runs are technical replicates of each other.
#In this case, random pairings of mass-spec runs analysing the same sample, at different sites.
#Note that we specify no intercept term!
M <- model.matrix(~ grouping - 1, data = peptideData)
#Get out the list of peptides, both HEK (control) and peptides of interest.
peptides <- setdiff(colnames(peptideData), c("filename", "site", "mixture", "Date", "grouping"))
#Reduce the data matrix to only the peptide data
onlyPeptideData <- data.matrix(peptideData[, peptides])
#All the human peptides are potential controls. That is, everything that's not an SIS peptides.
potentialControls <- setdiff(peptides, sisPeptides)
#But we want to use controls that are *often* found
potentialControlsOftenFound <- names(which(apply(onlyPeptideData[, potentialControls], 2,
    function(x) sum(is.na(x))) <= 10))
#Actually run correction
results <- RUVIII_C_Varying_residual_dimension(Y = log10(onlyPeptideData), M = M,
    potentialControls = potentialControlsOftenFound)
```

---

sisPeptides *List of stable isotope-labelled standard (SIS) peptides from the cross-lab study of Collins et al (2017)*

Description

List of stable isotope-labelled standard (SIS) peptides from the cross-lab study of Collins et al (2017)

Usage

```
data(crossLab)
```

Format

An object of class `character` of length 29.

Index