Package ‘msImpute’

August 12, 2020

Type Package

Title Peptide imputation in label-free proteomics

Version 1.3.0

Description msImpute provides tools for matrix completion in label-free proteomics quantification at the peptide-level. Currently, msImpute completes missing values by low-rank approximation of the underlying data matrix.

Imports softImpute

License MIT

Encoding UTF-8

LazyData true

BugReports `https://github.com/DavisLaboratory/msImpute/issues`

RoxygenNote 7.0.2

R topics documented:

`computeStructuralMetrics`

Metrics for the assessment of post-imputation structural preservation

---

Description

For an imputed dataset, it computes within phenotype/experimental condition similarity (i.e. preservation of local structures), between phenotype distances (preservation of global structures), and the Gromov-Wasserstein (GW) distance between original (source) and imputed data.

Usage

```
computeStructuralMetrics(x, group = NULL, y = NULL, k = 2)
```

Arguments

`x`	numeric matrix. An imputed data matrix of log-intensity.
`group`	factor. A vector of biological groups, experimental conditions or phenotypes (e.g. control, treatment).
`y`	numeric matrix. The source data (i.e. the original log-intensity matrix), preferably subsetted on highly variable peptides (see `findVariableFeatures`).
`k`	numeric. Number of Principal Components used to compute the GW distance. default to 2.

Details

For each group of experimental conditions (e.g. treatment and control), the group centroid is calculated as the average of observed peptide intensities. Withinness for each group is computed as sum of the squared distances between samples in that group and the group centroid. Betweenness is computed as sum of the squared distances between group centroids. When comparing imputation approaches, the optimal imputation strategy should minimize the within group distances, hence smaller withinness, and maximizes between group distances, hence larger betweenness. The GW metric considers preservation of both local and global structures simultaneously. A small GW distance suggests that imputation has introduced small distortions to global and local structures overall, whereas a large distance implies significant distortions. When comparing two or more imputation methods, the optimal method is the method with smallest GW distance. The GW distance is computed on Principal Components (PCs) of the source and imputed data, instead of peptides. Principal components capture the geometry of the data, hence GW computed on PCs is a better measure of preservation of local and global structures. The PCs in the source data are recommended to be computed on peptides with high biological variance. Hence, users are recommended to subset the source data only on highly variable peptides (hvp) (see `findVariableFeatures`). Since the hvp peptides have high biological variance, they are likely to have enough information to discriminate samples from different experimental groups. Hence, PCs computed on those peptides should be representative of the original source data with missing values. If the samples cluster by experimental group in the first couple of PCs, then a choice of k=2 is reasonable. If the desired separation/clustering of samples occurs in later PCs (i.e. the first few PCs are dominated by batches or unwanted variability), then it is recommended to use a larger number of PCs to compute the GW metric. If you are interested in how well the imputed data represent the original data in all possible dimensions, then set k to the number of samples in the data (i.e. the number of columns in the intensity matrix). GW distance estimation requires `python`. See example. All metrics are on log scale.

Value

list of three metrics: withinness (sum of squared distances within a phenotype group), betweenness (sum of squared distances between the phenotypes), and gromov-wasserstein distance (if `xna` is not NULL). if `group` is NULL only the GW distance is returned. All metrics are on log scale.

Examples

```
# To compute the GW distance you need to have python installed
# then install the reticulate R package from CRAN
# install.packages("reticulate")
```

```
library(reticulate)
# create a virtual environment
virtualenv_create('r-reticulate')
py_available() # if this returns TRUE, you've access to python from R.
# See reticulate if you need to troubleshoot
# install scipy and POT python packages in this virtual environment
virtualenv_install("msImpute-reticulate","scipy")
virtualenv_install("msImpute-reticulate","POT")
# if this runs successfully, the installation has been successful:
scipy <- import("scipy")
# You can now run the computeStructuralMetrics() function to compute GW distance.
# This setup should only be done for the first use. For all subsequent usages
# load the virtual environment that you've created using:
library(reticulate)
use_virtualenv("msImpute-reticulate")
# you can then run the computeStructuralMetrics() function.
# Note that the reticulate package should be loaded before loading msImpute.
set.seed(101)
n=12000
p=10
J=5
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J,mean = 5,sd = 0.2),n,J)%*%matrix(rnorm(J*p, mean = 5,sd = 0.2),J,p)+
  matrix(rnorm(np,mean = 5,sd = 0.2),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
keep <- (rowSums(!is.na(xna)) >= 4)
xna <- xna[keep,]
rownames(xna) <- 1:nrow(xna)
y <- xna
xna <- scaleData(xna)
xcomplete <- msImpute(object=xna)
G <- as.factor(sample(1:3, p, replace = TRUE))
top.hvp <- findVariableFeatures(y)
computeStructuralMetrics(xcomplete, G, y[rownames(top.hvp)[1:50],], k = 2)
```

`CPD` *CPD*

Description

Spearman correlation between pairwise distances in the original data and imputed data. CPD quantifies preservation of the global structure after imputation. Requires complete datasets - for developers/use in benchmark studies only.

Usage

```
CPD(xorigin, ximputed)
```

Arguments

`xorigin`	numeric matrix. The original log-intensity data. Can not contain missing values.
`ximputed`	numeric matrix. The imputed log-intensity data. Can not contain missing values.

Value

numeric

`findVariableFeatures` *Find highly variable peptides*

Description

For each peptide, the total variance is decomposed into biological and technical variance using package `scran`

Usage

```
findVariableFeatures(y)
```

Arguments

`y`	numeric matrix giving log-intensity. Can contain NA values.

Details

A loess trend is fitted to total sample variances and mean intensities. For each peptide, the biological variance is then computed by subtracting the estimated technical variance from the loess fit from the total sample variance.

Value

A data frame where rows are peptides and columns contain estimates of biological and technical variances. Peptides are ordered by biological variance.

See Also

computeStructuralMetrics

KNC *k-nearest class means (KNC)*

Description

The fraction of k-nearest class means in the original data that are preserved as k-nearest class means in imputed data. KNC quantifies preservation of the mesoscopic structure after imputation. Requires complete datasets - for developers/use in benchmark studies only.

Usage

```
KNC(xorigin, ximputed, class, k = 3)
```

Arguments

`xorigin`	numeric matrix. The original log-intensity data. Can contain missing values.
`ximputed`	numeric matrix. The imputed log-intensity data.
`class`	factor. A vector of length number of columns (samples) in the data specifying the class/label (i.e. experimental group) of each sample.
`k`	number of nearest class means. default to k=3.

Value

numeric The proportion of preserved k-nearest class means in imputed data.

KNN *k-nearest neighbour (KNN)*

Description

The fraction of k-nearest neighbours in the original data that are preserved as k-nearest neighbours in imputed data. KNN quantifies preservation of the local, or microscopic structure. Requires complete datasets - for developers/use in benchmark studies only.

Usage

```
KNN(xorigin, ximputed, k = 3)
```

Arguments

`xorigin`	numeric matrix. The original log-intensity data. Can not contain missing values.
`ximputed`	numeric matrix. The imputed log-intensity data. Can not contain missing values.
`k`	number of nearest neighbours. default to k=3.

Value

numeric The proportion of preserved k-nearest neighbours in imputed data.

msImpute — *Peptide-level imputation in mass spectrometry label-free proteomics by low-rank approximation*

Description

Returns a completed peptide intensity matrix where missing values (NAs) are imputated by low-rank approximation of the input matrix. Non-NA entries remain unmodified. `msImpute` requires at least 4 non-missing measurements per peptide across all samples. It is assumed that peptide intensities (DDA), or MS1/MS2 normalised peak areas (DIA), are log2-transformed and normalised (e.g. quantile normalisation).

Usage

```
msImpute(object, rank.max = NULL, lambda = NULL, thresh = 1e-05,
  maxit = 100, trace.it = FALSE, warm.start = NULL,
  final.svd = TRUE)
```

Arguments

`object` Numeric matrix giving log-intensity where missing values are denoted by NA. Rows are peptides, columns are samples.

`rank.max` Numeric. This restricts the rank of the solution. is set to min(dim(`object`)-1) by default.

`lambda` Numeric. Nuclear-norm regularization parameter. Controls the low-rank property of the solution to the matrix completion problem. By default, it is determined at the scaling step. If set to zero the algorithm reverts to "hardImputation", where the convergence will be slower.

`thresh` Numeric. Convergence threshold. Set to 1e-05, by default.

`maxit` Numeric. Maximum number of iterations of the algorithm before the algorithm is converged. 100 by default.

`trace.it` Logical. Prints traces of progress of the algorithm.

`warm.start` List. A SVD object can be used to initialize the algorithm instead of random initialization.

`final.svd` Logical. Shall final SVD object be saved? The solutions to the matrix completion problems are computed from U, D and V components of final SVD.

Details

`msImpute` operates on the softImpute-ALS algorithm. For more details on the underlying algorithm, please see `softImpute` package.

Value

Missing values are imputed by low-rank approximation of the input matrix. If input is a numeric matrix, a numeric matrix of identical dimensions is returned. If `x` is a `MAList` object, the `E` component is replaced with the completed matrix, and the updated `MAList` object is returned. Non-NA entries remain unmodified.

See Also

selectFeatures, scaleData

Examples

```
set.seed(101)
n=12000
p=10
J=5
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J,mean = 5,sd = 0.2),n,J)%*%matrix(rnorm(J*p, mean = 5,sd = 0.2),J,p)+
  matrix(rnorm(np,mean = 5,sd = 0.2),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
keep <- (rowSums(!is.na(xna)) >= 4)
xna <- xna[keep,]
xna <- scaleData(xna)
xcomplete <- msImpute(object=xna)
```

`scaleData` *Standardize a matrix to have optionally row means zero and variances one, and/or column means zero and variances one.*

Description

Standardize a matrix to have optionally row means zero and variances one, and/or column means zero and variances one.

Usage

```
scaleData(object, maxit = 20, thresh = 1e-09, row.center = TRUE,
  row.scale = TRUE, col.center = TRUE, col.scale = TRUE,
  trace = FALSE)
```

Arguments

`object` numeric matrix giving log-intensity where missing values are denoted by NA. Rows are peptides, columns are samples.

`maxit` numeric. maximum iteration for the algorithm to converge (default to 20). When both row and column centering/scaling is requested, iteration may be necessary.

`thresh` numeric. Convergence threshold (default to 1e-09).

`row.center` logical. if row.center==TRUE (the default), row centering will be performed resulting in a matrix with row means zero. If row.center is a vector, it will be used to center the rows. If row.center=FALSE nothing is done.

`row.scale` if row.scale==TRUE, the rows are scaled (after possibly centering, to have variance one. Alternatively, if a positive vector is supplied, it is used for row centering.

`col.center` Similar to row.center

`col.scale` Similar to row.scale

`trace` logical. With trace=TRUE, convergence progress is reported, when iteration is needed.

Details

Standardizes rows and/or columns of a matrix with missing values, according to the `biScale` algorithm in Hastie et al. 2015. Data is assumed to be normalised and log-transformed.

Value

A list of two components: E and E.scaled. E contains the input matrix, E.scaled contains the scaled data

See Also

selectFeatures, msImpute

Examples

```
set.seed(101)
n=12000
p=10
J=5
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J,mean = 5,sd = 0.2),n,J)%*%matrix(rnorm(J*p, mean = 5,sd = 0.2),J,p)+
  matrix(rnorm(np,mean = 5,sd = 0.2),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
keep <- (rowSums(!is.na(xna)) >= 4)
xna <- xna[keep,]
xna <- scaleData(xna)
```

`selectFeatures` *Select features with high biological dropout rate*

Description

Fits a linear model to peptide dropout rate against peptide abundance. The selected features (peptides) can be used to determine if data is Missing Not At Random (MNAR). Users should note that `msImpute` assumes peptides are Missing At Random (MAR).

Usage

```
selectFeatures(object, n_features = 500, suppress_plot = FALSE)
```

Arguments

`object` Numeric matrix giving log-intensity where missing values are denoted by NA. Rows are peptides, columns are samples.

`n_features` Numeric, number of features with high dropout rate. 500 by default.

`suppress_plot` Logical show plot of dropouts vs abundances.

Value

A data frame with a logical column denoting the selected features

See Also

scaleData, msImpute

Examples

```
set.seed(101)
n=12000
p=10
J=5
np=n*p
missfrac=0.3
x=matrix(rnorm(n*J,mean = 5,sd = 0.2),n,J)%*%matrix(rnorm(J*p, mean = 5,sd = 0.2),J,p)+
  matrix(rnorm(np,mean = 5,sd = 0.2),n,p)/5
ix=seq(np)
imiss=sample(ix,np*missfrac,replace=FALSE)
xna=x
xna[imiss]=NA
keep <- (rowSums(!is.na(xna)) >= 4)
xna <- xna[keep,]
rownames(xna) <- 1:nrow(xna)
hdp <- selectFeatures(xna, n_features=500,  suppress_plot=FALSE)
# construct matrix M to capture missing entries
M <- ifelse(is.na(xna),1,0)
M <- M[hdp$msImpute_feature,]
# plot a heatmap of missingness patterns for the selected peptides
library(ComplexHeatmap)
hm <- Heatmap(M,
column_title = "dropout pattern, columns ordered by dropout similarity",
              name = "",
              col = c("#8FBC8F", "#FFEFDB"),
              show_row_names = FALSE,
              show_column_names = TRUE,
              cluster_rows = TRUE,
              cluster_columns = TRUE,
              show_column_dend = FALSE,
              show_row_dend = FALSE,
              row_names_gp =  gpar(fontsize = 7),
              column_names_gp = gpar(fontsize = 8),
              heatmap_legend_param = list(#direction = "horizontal",
              heatmap_legend_side = "bottom",
              labels = c("observed","missing"),
              legend_width = unit(6, "cm")),
         )
hm <- draw(hm, heatmap_legend_side = "left")
```

Index