**Package ‘ProteoMM’**

May 30, 2024

**Title**  Multi-Dataset Model-based Differential Expression Proteomics Analysis Platform

**Version**  1.22.0

**Description**  ProteoMM is a statistical method to perform model-based peptide-level differential expression analysis of single or multiple datasets. For multiple datasets ProteoMM produces a single fold change and p-value for each protein across multiple datasets.

ProteoMM provides functionality for normalization, missing value imputation and differential expression.

Model-based peptide-level imputation and differential expression analysis component of package follows the analysis described in “A statistical framework for protein quantitation in bottom-up MS based proteomics” (Karpievitch et al. Bioinformatics 2009).

EigenMS normalisation is implemented as described in "Normalization of peak intensities in bottom-up MS-based proteomics using singular value decomposition.” (Karpievitch et al. Bioinformatics 2009).

**Author**  Yuliya V Karpievitch, Tim Stuart and Sufyaan Mohamed

**Maintainer**  Yuliya V Karpievitch <yuliya.k@gmail.com>

**License**  MIT

**LazyData**  TRUE

**Depends**  R (>= 3.5)

**Encoding**  UTF-8

**RoxygenNote**  6.1.0

**Imports**  gdata, biomaRt, ggplot2, ggrepel, gtools, stats, matrixStats, graphics

**biocViews**  ImmunoOncology, MassSpectrometry, Proteomics, Normalization, DifferentialExpression

**Suggests**  BiocStyle, knitr, rmarkdown

**VignetteBuilder**  knitr

**git_url**  https://git.bioconductor.org/packages/ProteoMM
**convert_log2**

Convert values in a matrix to log2 transformed values

**Description**

`convert_log2` replaces 0's with NA's than does a log2 transformation. Replacing 0's with NA's is the correct approach to Proteomics data analysis as 0's are not values that should be left in the data where no observation was made, see citation below. Karpievitch et al. 2009 "Normalization of peak intensities in bottom-up MS-based proteomics using singular value decomposition". PMID: 19602524 Karpievitch et al. 2009 "A statistical framework for protein quantitation in bottom-up MS-based proteomics". PMID: 19535538
**eigen_pi**

**Usage**

```r
calculate_log2(mm, use_cols)
```

**Arguments**

- `mm`: a dataframe of raw intensities in format: (# peptides)x(# samples+possibly peptide & protein information (metadata))
- `use_cols`: vector of column indecies that make up the intensities usually in sequential order but do not have to be user is responsible for making sure that specified columns are indeed numeric and correspond to intensities for each sample

**Value**

matrix of log2 transformed intensities where 0’s were replaced with NA’s prior to transformation

**Examples**

```r
data(mm_peptides)
head(mm_peptides)
intsCols = 8:13
metaCols = 1:7
m_logInts = make_intencities(mm_peptides, intsCols)
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts) # 0's replaced with NAs and # log2 transform applied
```

---

**eigen_pi**

*Compute PI - proportion of observations missing completely at random*

**Description**

Compute PI - proportion of observations missing completely at random

**Usage**

```r
eigen_pi(m, toplot = TRUE)
```

**Arguments**

- `m`: matrix of abundances, numsamples x numpeptides
- `toplot`: TRUE/FALSE plot mean vs protportion missing curve and PI

**Value**

pi estimate of the proportion of observations missing completely at random

Contributed by Shelley Herbrich & Tom Taverner for Karpievitch et al. 2009
Examples

data(mm_peptides)
intsCols = 8:13
metaCols = 1:7
m_logInts = make_intencities(mm_peptides, intsCols)
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
my.pi = eigen.pi(m_logInts, toplot=TRUE)

Description


Usage

eig_norm1(m, treatment, prot.info, write_to_file = "")

Arguments

m number of peptides x number of samples matrix of log-transformed expression data, metadata not included in this matrix
treatment either a single factor indicating the treatment group of each sample i.e. [1 1 1 2 2 2 ...] or a data frame of factors, eg: treatment= data.frame(cbind(data.frame(Group), data.frame(Time)))prot.info 2+ column data frame, pepID, prID columns IN THAT ORDER. IMPORTANT: pepIDs must be unique identifiers and will be used as Row Names If normalizing non-proteomics data, create a column such as: paste('ID_'.seq_len(num_rows), sepe=") Same can be done for ProtIDs, these are not used for normalization but are kept for future analyseswrite_to_file if a string is passed in, 'complete' peptides (peptides with NO missing observations) will be written to that file name

Value

A structure with multiple components

m, treatment, prot.info, grp initial parameters passed into the function, returned for future reference
mysvd matrices produced by SVD
**eig_norm2**

**Description**


**Usage**

```r
eig_norm2(rv)
```
Arguments

rv return value from the eig_norm1 if user wants to change the number of bias trends that will be eliminated h.c in rv should be updates to the desired number

Value

A structure with multiple components

- **normalized**: matrix of normalized abundances with 2 columns of protein and peptdie names
- **norm_m**: matrix of normalized abundances, no extra columns
- **eigentrends**: trends found in raw data, bias trends up to h.c
- **norm.svd**: trends in normalized data, if one wanted to plot at later time
- **exPeps**: peptides excluded due to not enough peptides or exception in fitting a linear model

Examples

```
data(mm_peptides)
head(mm_peptides)
# different from parameter names as R uses outer name
# spaces if variable is undefined
intsCols = 8:13
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(mm_peptides, intsCols)
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG', 'mCG'))

set.seed(123) # set for repoducubility of eig_norm1
mm_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
mm_m_ints_eig1$h.c # check the number of bias trends detected
mm_m_ints_norm = eig_norm2(rv=mm_m_ints_eig1)
```

g.test

**G Test for presence - absence analysis**

Description

Log-likelihood test for independence & goodness of fit. g.test() performs Williams’ and Yates’ correction; Monte Carlo simulation of p-values, via gtestsim.c. MC requires recompilation of R. Written by Peter Hurd (V3.3 Pete Hurd Sept 29 2001, phurd AT ualberta.ca). Yuliya Karpievitch added comments for ease of understanding and incorporated into ProteoMM. G & q calculation from Sokal & Rohlf (1995) Biometry 3rd ed., TOI Yates correction taken from Mike Camanns 2x2 G-test function, GOF Yates correction as described in Zar (2000), more stuff taken from ctest’s chisq.test().
get_presAbs_prots

Usage

```r
g.test(x, y = NULL, correct = "none", p = rep(1/length(x), length(x)))
```

Arguments

- **x**: vector of boolean values corresponding to presence & absence eg: c(TRUE, TRUE, FALSE, FALSE) for present present absent absent values. Order of TRUE/FALSE does not matter, can be used interchangeably. Same length as parameter y
- **y**: vector treatments (factor) corresponding to values in x, same length as x eg: as.factor(c('grp1', 'grp1', 'grp2', 'grp2'))
- **correct**: correction to apply, options: "yates", "williams", "none" default: "none" NOTE: in ProteoMM we only tested & used correction = "none"
- **p**: default: rep(1/length(x), length(x)), used in Yates correction NOTE: in ProteoMM we only tested & used the default parameter value

Value

- htest object the following variables
  - **statistic**: value of the G statistic produced by g test
  - **parameter**: degrees of freedom of the test
  - **p.value**: p-value
  - **method**: method used to produce statistic and p-value
  - **data.name**: data passed in to the function
  - **observed**: matrix of observed counts
  - **expected**: matrix of expected counts

Examples

```r
g.test(c(TRUE, TRUE, FALSE, FALSE),
       as.factor(c('grp1', 'grp1', 'grp2', 'grp2')))  
```

---

get_presAbs_prots    Get Presence/Absence Proteins

Description

Function get_presAbs_prots() produces a subset of protein meta data and intensities for multiple datasets pass in as a list. If a single dataset is passed in (list of length one) it will be processed in the same way as longer lists.

Usage

```r
g.get_presAbs_prots(mm_list, prot.info, protnames_norm, prot_col_name)
```
get_presAbs_prots

Arguments

- **mm_list** list of matrices of intensities for each experiment. Dimentions: numpeptides x numsamples different for each dataset.
- **prot.info** list of protein and peptide metadata/mappings for each matrix in mm_list, data.frames "parallel" to matrices in mm_list.
- **protnames_norm** list of protein pidentifies to be used to determine peptides that will be placed into Presence/Absence analysis category due to too many missing peptides. Taken from the return value from eig_norm2().
- **prot_col_name** column name (string) that will be used to get ProteinIDs in the raw data matrices

Value

- list of lists of length 2
  - **intensities** list of intecities in the same order and of the same length as the number of datasets that were passed into the function
  - **protein metadata** list of protein metadata in the same order and of the same length as the number of datasets that as were passed into the function

Examples

```r
# Load mouse dataset
data(mm_peptides)
head(mm_peptides)
intsCols = 8:13
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(mm_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG'))
mm_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
mm_m_ints_eig1$h.c # check the number of bias trends detected
mm_m_ints_norm = eig_norm2(rv=mm_m_ints_eig1)

# Load human dataset
data(hs_peptides)
head(hs_peptides)
intsCols = 8:13
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(hs_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(hs_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG'))
hs_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
hs_m_ints_eig1$h.c # check the number of bias trends detected
hs_m_ints_norm = eig_norm2(rv=hs_m_ints_eig1)

# Set up for presence/absence analysis
raw_list = list()
norm_imp_prot.info_list = list()
```
hs_peptides

```
raw_list[[1]] = mm_m_ints_eig$m
raw_list[[2]] = hs_m_ints_eig$m
norm_imp_prot.info_list[[1]] = mm_m_ints_eig$prot.info
norm_imp_prot.info_list[[2]] = hs_m_ints_eig$prot.info

protnames_norm_list = list()
protnames_norm_list[[1]] = unique(mm_m_ints_norm$normalized$MatchedID)
protnames_norm_list[[2]] = unique(hs_m_ints_norm$normalized$MatchedID)

presAbs_dd = get_presAbs_prots(mm_list=raw_list,
prot.info=norm_imp_prot.info_list,
protnames_norm=protnames_norm_list,
prot_col_name=2)
```

---

### hs_peptides

**hs_peptides - peptide-level intensities for human**

**Description**

A dataset containing the protein and peptide information and peptide-level intensities for 6 samples:
3 CG and 3 mCG groups. There are 69 proteins. The columns are as follows:

**Usage**

```
data(hs_peptides)
```

**Format**

A data frame with 695 rows and 13 columns, comprising 7 columns of metadata and 6 columns of
peptide intensities. 69 proteins.

**Details**

- Sequence - peptide sequence - randomly chosen from a larger list of sequences
- MatchedID - numeric ID that links proteins in the two datasets, unnecessary if datasets are for
  the same species
- ProtID - protein ID, artificial protein ID, eg. Prot1, Prot2, ...
- GeneID - gene ID, artificial gene ID, eg. Gene1, Gene2, ...
- ProtName - artificial Protein Name
- ProtIDLong - long protein ID, full protein name, here artificially simulated
- GeneIDLong - long gene ID, full gene name, here artificially simulated
- CG1 - raw intensity column for sample 1 in CG group
- CG2 - raw intensity column for sample 2 in CG group
- CG3 - raw intensity column for sample 3 in CG group
- mCG1 - raw intensity column for sample 1 in mCG group
- mCG2 - raw intensity column for sample 2 in mCG group
- mCG3 - raw intensity column for sample 3 in mCG group
**makeLMFormula**

*String linear model formula suitable*

**Description**

Makes a string linear model formula suitable for the right hand side of the equation passed into `lm()`

**Usage**

```r
makeLMFormula(eff, var_name = "")
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eff</code></td>
<td>treatment group ordering for all samples being analysed. Single factor with 2+ treatment groups. Used to generate formula and contrasts for <code>lm()</code></td>
</tr>
<tr>
<td><code>var_name</code></td>
<td>string variable name to use in the formula</td>
</tr>
</tbody>
</table>

**Details**

`eig_norm1` and `eig_norm2` Here we incorporate the model matrix from EigenMS normalization to find the significant trends in the matrix of residuals.

**Value**

data structure with linear model formula and contrasts

- **lm.formula**: Linear model formula suitable for the right hand side of `~` in `lm()`, `~` is not included in the formula
- **lm.params**: contrasts for `lm()`, here sum-to-zero constraint only

**Examples**

```r
groups = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG', 'mCG'))
makeLMFormula(groups, 'TREATS')
```

---

**make_intencities**

*Subdivide data into intensities columns only*

**Description**

Subdivide a data frame of protein intensities and metadata into intensities only. No row names will be provided.

**Usage**

```r
make_intencities(mm, use_cols)
```
**make_meta**

Subdivide data into metadata columns only

**Description**
Subdivide a data frame of protein metadata and intensities into a data frame of meta data only.

**Usage**

```r
make_meta(mm, use_cols)
```

**Arguments**

- `mm` data frame of metadata and intensities as a single data frame
- `use_cols` column numbers to subset and return, no range checking on the column indeces is performed

**Value**

- `m_ints` data frame of intensities only

**Examples**

```r
data(mm_peptides)
head(mm_peptides)
intsCols = 8:13 # different from parameter names as R uses outer name # spaces if variable is undefined
m_logInts = make_intencities(mm_peptides, intsCols)
```

```r
data(mm_peptides)
head(mm_peptides)
metaCols = 1:7 # reusing this variable
m_prot.info = make_meta(mm_peptides, metaCols)
```
**MBimpute**

*Model-Based Imputation of missing values*

**Description**

Impute missing values based on information from multiple peptides within a protein. Expects the data to be filtered to contain at least one observation per treatment group. For experiments with lower overall abundances such as multiplexed experiments check if the imputed value is below 0, if so value is re-imputed until it is above 0.

**Usage**

```r
MBimpute(mm, treatment, prot.info, pr_ppos = 2, my.pi = 0.05,
compute.pi = FALSE)
```

**Arguments**

- `mm` : number of peptides x number of samples matrix of intensities
- `treatment` : vector indicating the treatment group of each sample eg as.factor(c('CG','CG','CG', 'mCG','mCG','mCG')) or c(1,1,1,2,2,2)
- `prot.info` : protein metadata, 2+ columns: peptide IDs, protein IDs, etc
- `pr_ppos` : column index for protein ID in prot.info
- `my.pi` : PI value, estimate of the proportion of peptides missing completely at random, as compared to censored at lower abundance levels default values of 0.05 is usually reasonable for missing completely at random values in proteomics data
- `compute.pi` : TRUE/FALSE (default=FALSE) estimate Pi is set to TRUE, otherwise use the provided value. We consider Pi=0.05 a reasonable estimate for observations missing completely at random in proteomics experiments. Thus values is set to NOT estimate Pi by default. Note: spline smoothing can sometimes produce values of Pi outside the range of possible values.

**Value**

A structure with multiple components

- `y_imputed` : number of peptides x m matrix of peptides with no missing data
- `imp_prot.info` : imputed protein info, 2+ columns: peptide ID, protein IDs, etc Dimensions should be the same as passed in

**Examples**

```r
data(mm_peptides)
head(mm_peptides)
intsCols = 8:13 # different from parameter names as R uses outer name spaces
# if variable is undefined
metaCols = 1:7 # reusing this variable
```
Description

A dataset containing the protein and petide information and peptide-level intensities for 6 samples: 3 CG and 3 mCG groups. There are 69 proteins. The columns are as follows:

Usage

data(mm_peptides)

Format

A data frame with 1102 rows and 13 columnns, compiring 7 columns of metadata and 6 columns of peptide intensities. 69 proteins.

Details

- Sequence - peptide sequence - randomly chosen from a larger list of sequences
- MatchedID - numeric ID that links proteins in the two datasets, unnecessary if datasets are for the same species
- ProtID - protein ID, artificial protein ID, eg. Prot1, Prot2, ...
- GeneID - gene ID, artificial gene ID, eg. Gene1, Gene2, ...
- ProtName - artificial Protein Name
- ProtIDLong - long protein ID, full protein name, here artificially simulated
- GeneIDLong - long gene ID, full gene name, here artificially simulated
- CG1 - raw intensity column for sample 1 in CG group
- CG2 - raw intensity column for sample 2 in CG group
peptideLevel_DE

Model-Based differential expression analysis

Description

Model-Based differential expression analysis is performed on peptide level as described in Karpievitch et al. 2009 "A statistical framework for protein quantitation in bottom-up MS-based proteomics" Bioinformatics.

Usage

peptideLevel_DE(mm, treatment, prot.info, pr_ppos = 2)

Arguments

- **mm**: m x n matrix of intensities, num peptides x num samples
- **treatment**: vector indicating the treatment group of each sample ie [1 1 1 1 2 2 2 2...]
- **prot.info**: 2+ column data frame of peptide ID, protein ID, etc. columns
- **pr_ppos**: column index for protein ID in prot.info. Can restrict to be #2...

Value

A data frame with the following columns:

- **ProtID**: protein identification information taken from prot.info, 1 column used to identify proteins
- **FC**: fold change
- **p-value**: p-value for the comparison between 2 groups (2 groups only here)
- **BH-adjusted p-value**: Benjamini-Hochberg adjusted p-values

Examples

data(mm_peptides)
head(mm_peptides)
# different from parameter names as R uses outer
# name spaces if variable is undefined
intsCols = 8:13
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(mm_peptides, intsCols)
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c("CG","CG","CG","mCG","mCG","mCG"))
peptideLevel_PresAbsDE

Presence/Absence peptide-level analysis

Description

Presence/Absence peptide-level analysis uses all peptides for a protein as IID to produce 1 p-value across multiple (2+) datasets. Significance is estimated using a g-test which is suitable for two treatment groups only.

Usage

peptideLevel_PresAbsDE(mm, treatment, prot.info, pr_ppos = 2)

Arguments

mm m x n matrix of intensities, num peptides x num samples

treatment vector indicating the treatment group of each sample ie [1 1 1 2 2 2 ...]

prot.info 2+ column data frame of peptide ID, protein ID, etc columns

pr_ppos - column index for protein ID in prot.info. Can restrict to be #2...

Value

A list of length two items:

ProtIDused protein identification information taken from prot.info, a column used to identify proteins

FC Approximation of the fold change computed as percent missing observations group 1 muns in percent missing observations group 2

P_val p-value for the comparison between 2 groups (2 groups only here)

BH_P_val Benjamini-Hochberg adjusted p-values
statistic statistic returned by the g-test, not very useful as depends on the direction of the test and can produce all 0’s

num_peptides number of peptides within a protein

metadata all columns of metadata from the passed in matrix

Examples

# Load mouse dataset
data(mm_peptides)
head(mm_peptides)
intsCols = 8:13 # different from parameter names as R uses
    # outer name spaces if variable is undefined
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(mm_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG', 'mCG'))

set.seed(135)
mm_m_ints_eig1 = eig_norm1(m=m_logInts,treatment=grps,prot.info=m_prot.info)
m_m_ints_eig1$h.c # check the number of bias trends detected
mm_m_ints_norm = eig_norm2(rv=mm_m_ints_eig1)

# Load human dataset
data(hs_peptides)
head(hs_peptides)
intsCols = 8:13 # different from parameter names as R
    # uses outer name spaces if variable is undefined
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(hs_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(hs_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG', 'mCG'))

set.seed(137) # different seed for different organism
hs_m_ints_eig1 = eig_norm1(m=m_logInts,treatment=grps,prot.info=m_prot.info)
hs_m_ints_eig1$h.c # check the number of bias trends detected
hs_m_ints_norm = eig_norm2(rv=hs_m_ints_eig1)

# Set up for presence/absence analysis
raw_list = list()
norm_imp.prot.info_list = list()
raw_list[[1]] = mm_m_ints_eig1$m
raw_list[[2]] = hs_m_ints_eig1$m
norm_imp.prot.info_list[[1]] = mm_m_ints_eig1$prot.info
norm_imp.prot.info_list[[2]] = hs_m_ints_eig1$prot.info
protnames_norm_list = list()
protnames_norm_list[[1]] = unique(mm_m_ints_norm$normalized$MatchedID)
protnames_norm_list[[2]] = unique(hs_m_ints_norm$normalized$MatchedID)
presAbs_dd = get_presAbs_prots(mm_list=raw_list,
Plot trends for a single protein

Description
Plot peptide trends for a protein

Usage
plot_1prot(mm, prot.info, prot_to_plot, prot_to_plot_col, gene_name, gene_name_col, colors, mylabs)

Arguments
- **mm**: matrix of raw intensities
- **prot.info**: metadata for the intensities in mm
- **prot_to_plot**: protein ID to plot
- **prot_to_plot_col**: protein ID column index
- **gene_name**: gene ID to plot
- **gene_name_col**: gene ID to plot column index
- **colors**: what colors to plot peptide abundances as, most commonly should be treatment groups
- **mylabs**: sample labels to be plotted on x-axis

Value
Nil
Description

Plot Raw, Normalized and Normalized & Imputed peptide trends for a protein

Usage

plot_3_pep_trends_NOfile(mm, prot.info, sorted_norm_m, sorted_prot.info,
    imp_mm, imp_prot.info, prot_to_plot, prot_to_plot_col, gene_name,
    gene_name_col, mylabs)

Arguments

mm          matrix of raw intensities
prot.info   metadata for the intensities in mm
sorted_norm_m normalized intensities, possibly fewer than in mm due to filtering out peptides
            with fewer than one observation per treatment group
sorted_prot.info metadata for the intensities in sorted_norm_m
imp_mm      imputed intensities (post normalization)
imp_prot.info metadata for the imputed intensities in imp_mm
prot_to_plot protein ID to plot
prot_to_plot_col protein ID column index
gene_name   gene ID to plot
gene_name_col gene ID to plot column index
mylabs      sample labels to be plotted on x-axis

Value

Nil

Examples

data("hs_peptides") # loads variable hs_peptides
intsCols = 8:13 # column indeces that contain intensities
m_logInts = make_intencities(hs_peptides, intsCols)
# replace 0's with NA's as NA's are more appropriate
# for analysis and log2 transform
m_logInts = convert_log2(m_logInts)
# column indices that contain metadata such as protein IDs and sequences
metaCols = 1:7
m_prot.info = make_meta(hs.peptides, metaCols)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG'))

set.seed(135)
hs_m_ints_eig1 = eig_norm1(m=m_logInts,treatment=grps,prot.info=m_prot.info)
hs_m_ints_eig1$h.c = 2 # looks like there are 2 bias trends at least
hs_m_ints_norm = eig_norm2(rv=hs_m_ints_eig1)
hs_prot.info = hs_m_ints_norm$normalized[,metaCols]
hs_norm_m = hs_m_ints_norm$normalized[,intsCols]

set.seed(125)
imp_hs = MBimpute(hs_norm_m, grps, prot.info=hs_prot.info,
                 pr_ppos=3, my.pi=0.05, compute_pi=FALSE)
mylabs = c( 'CG', 'CG', 'CG', 'mCG', 'mCG')
prot_to_plot = 'Prot32' # 43
gene_to_plot = 'Gene32'
plot_3_pep_trends_NOfile(as.matrix(hs_m_ints_eig1$m),
                          hs_m_ints_eig1$prot.info,
                          as.matrix(hs_norm_m),
                          hs_prot.info,
                          imp_hs$y_imputed,
                          imp_hs$imp_prot.info,
                          prot_to_plot, 3,
                          gene_to_plot, 4, mylabs)

plot_volcano

**Description**

Function plots fold changes and p-values as a volcano plot. Two lines are plotted for the p-value cutoff at \( p = PV\_cutoff \) (solid line) and \( p = 0.1 \) (dashed line).

**Usage**

```
plot_volcano(FC, PV, FC_cutoff = 2, PV_cutoff = 0.05, figtitle = "")
```

**Arguments**

- **FC**
  - vector of fold changes
- **PV**
  - vector of p-values, same length as FC
- **FC_cutoff**
  - fold change cutoff where to draw vertical cutoff lines, default = 2
- **PV_cutoff**
  - p-value cutoff where to draw a horizontal cutoff line, default = 0.05
- **figtitle**
  - title to display at the top of the figure, default = ""

**Value**

Nil
Examples

data(mm_peptides)
head(mm_peptides)
intsCols = 8:13 # different from parameter names as
           # R uses outer name spaces if variable is undefined
metaCols = 1:7
m_logInts = make_intencities(mm_peptides, intsCols)
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)

# Normalize data
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG', 'mCG'))

set.seed(123)
mm_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
mm_m_ints_eig1$h.c # check the number of bias trends detected

# Impute missing values
mm_m_ints_norm = eig_norm2(rv=mm_m_ints_eig1)
m_prot.info = mm_m_ints_norm$normalized[,1:7]
mm_norm_m = mm_m_ints_norm$normalized[,8:13]

set.seed(125) # needed for reproducibility of imputation
imp_mm = MBimpute(mm_norm_m, grps, prot.info = m_prot.info,
                   pr_ppos=2, my.pi=0.05, compute.pi=FALSE)
DE_res = peptideLevel_DE(imp_mm$y_imputed, grps, imp_mm$imp_prot.info,
                         pr_ppos=2)
plot_volcano(DE_res$FC, DE_res$BH_P_val, FC_cutoff=1.5,
             PV_cutoff=.05, figtitle = 'Mouse DE')

plot_volcano_wLab

Volcano plot with labels for the differentially expressed proteins

Description

Function plots fold changes and p-values as a volcano plot. Two lines are plotted for the p-value
cutoff at p = PV_cutoff (solid line) and p = 0.1 (dashed line).

Usage

plot_volcano_wLab(FC, PV, ProtID, FC_cutoff = 2, PV_cutoff = 0.05,
                  figtitle = '')

Arguments

FC vector of fold changes
PV vector of p-values, same length as FC
ProtID vector of protein IDs, can be gene IDs, same length as FC & PV. Names in this vector will be displayed in the volcano plot for differentially expressed proteins for this reason short names are preferred.

FC_cutoff fold change cutoff where to draw vertical cutoff lines, default = 2

PV_cutoff p-value cutoff where to draw a horizontal cutoff line, default = .05

figtitle title to display at the top of the figure, default = ""

Value Nil

Examples

data(mm_peptides)
head(mm_peptides)
intsCols = 8:13 # different from parameter names as
# R uses outer name spaces if variable is undefined
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(mm_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)

# Normalize data
grps = as.factor(c(’CG’, ’CG’, ’CG’, ’mCG’, ’mCG’, ’mCG’, ’mCG’))

set.seed(135)
mm_m ints eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
mm_m ints eig1$h.c # check the number of bias trends detected

# Impute missing values
mm_m ints norm = eig_norm2(rv=mm_m ints eig1)
mm_prot.info = mm_m ints norm$normalized[,1:7]
mm_norm_m = mm_m ints norm$normalized[,8:13]

set.seed(125)
imp_mm = MBimpute(mm_norm_m, grps, prot.info=mm_prot.info,
pr_ppos=2, my.pi=0.05, compute_pi=FALSE)
DE_res = peptideLevel_DE(imp_mm$y_imputed, grps, imp_mm$imp_prot.info,
pr_ppos=2)
plot_volcano_wLab(DE_res$FC, DE_res$BH_P_val, DE_res$ProtID, FC_cutoff=1.5,
PV_cutoff=.05, figtitle='Mouse DE')
**Description**

Multi-Matrix Presence Absence Analysis computes Model-Based statistics for each dataset and sums them up to produce the final statistic. The significance is determined via a permutation test which computes the same statistics and sums them after permuting the values across treatment groups, as is outlined in Karpievitch et al. 2018. Whenever possible proteins should be analysed using the Model-Based Differential Expression Analysis due to higher statistical power over the Presence Absence analysis.

**Usage**

```r
prot_level_multiMat_PresAbs(mm_list, treat, prot.info, prot_col_name, 
nperm = 500, dataset_suffix)
```

**Arguments**

- `mm_list`: list of matrices of intensities for each experiment, dimensions: numpeptides x numsamples
- `treat`: list of data frames with treatment information to compute the statistic, parallel to `mm_list` and `prot.info`
- `prot.info`: list of protein metadata for each matrix in `mm_list`, data.frame parallel to `mm_list` and `treat`
- `prot_col_name`: column names present in all datasets that identifies protein IDs across all datasets
- `nperm`: number of permutations
- `dataset_suffix`: a list of strings that will be appended to the column names for FC, PV, BHPV and numbers of peptides

**Value**

A data frame with the following columns:

- `protIDused`: protein metadata, peptide sequence if was passed in as one of the columns is the first peptide sequence encountered in the data for that protein
- `FCs`: Averages across all datasets of the approximation of the fold change computed as percent missing observations group 1 munis in percent missing observations group 2 in peptideLevel_PresAbsDE() function
- `P_val`: p-value for the comparison between 2 groups (2 groups only here) obtained from a permutation test
- `BH_P_val`: Benjamini-Hochberg adjusted p-values
- `statistic`: statistic returned by the g-test and summed across all datasets, not very useful as depends on the direction of the test and can produce all 0’s
- `u_prot_info`: column containing protein identifiers across all datasets
- `PV`: p-values produced by g-test for individual datasets
- `BHPV`: adjusted p-values produced by g-test for individual datasets
- `NUMPEP`: number of peptides observed for each protein in each of the datasets
Examples

# Load mouse dataset
data(mm_peptides)
head(mm_peptides)
intsCols = 8:13
metaCols = 1:7
m_logInts = make_intencities(mm_peptides, intsCols)  # will reuse the name
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG', 'mCG'))

set.seed(135)
mm_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
mm_m_ints_eig1$h.c  # check the number of bias trends detected
mm_m_ints_norm = eig_norm2(rv=mm_m_ints_eig1)

# Load human dataset
data(hs_peptides)
head(hs_peptides)
intsCols = 8:13
metaCols = 1:7
m_logInts = make_intencities(hs_peptides, intsCols)
m_prot.info = make_meta(hs_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG', 'mCG'))

set.seed(137)
hs_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
hs_m_ints_eig1$h.c  # check the number of bias trends detected
hs_m_ints_norm = eig_norm2(rv=hs_m_ints_eig1)

# Set up for presence/absence analysis
raw_list = list()
norm_imp_prot.info_list = list()
raw_list[[1]] = mm_m_ints_eig1$m
raw_list[[2]] = hs_m_ints_eig1$m
norm_imp_prot.info_list[[1]] = mm_m_ints_eig1$prot.info
norm_imp_prot.info_list[[2]] = hs_m_ints_eig1$prot.info

protnames_norm_list = list()
protnames_norm_list[[1]] = unique(mm_m_ints_norm$normalized$MatchedID)
protnames_norm_list[[2]] = unique(hs_m_ints_norm$normalized$MatchedID)
presAbs_dd = get_presAbs_prots(mm_list=raw_list,
    prot.info=norm_imp_prot.info_list,
    protnames.norm=protnames_norm_list,
    prot.col.name=2)

ints_presAbs = list()
protmeta_presAbs = list()
ints_presAbs[[1]] = presAbs_dd[[1]][[1]]  # Mouse
ints_presAbs[[2]] = presAbs_dd[[1]][[2]]  # HS
protmeta_presAbs[1] = presAbs_dd[[2]][[1]]
protmeta_presAbs[2] = presAbs_dd[[2]][[2]]

treats = list()
treats[[1]] = grps
treats[[2]] = grps

subset_presAbs = subset_proteins(mm_list=ints_presAbs,
prot.info=protmeta_presAbs, 'MatchedID')
nperm = 50  # set to 500+ for publication
set.seed(275937)
presAbs_comb = prot_level_multiMat_PresAbs(
    mm_list=subset_presAbs$sub_mm_list,
    treat=treats,
    prot.info=subset_presAbs$sub_prot.info,
    prot_col_name='MatchedID', nperm=nperm,
    dataset_suffix=c('MM', 'HS'))

plot_volcano(presAbs_comb$FC, presAbs_comb$BH_P_val,
    FC_cutoff=.5, PV_cutoff=.05,
    'Combined Pres/Abs CG vs mCG')

###

**prot_level_multi_part  Multi-Matrix Differential Expression Analysis**

**Description**

Multi-Matrix Differential Expression Analysis computes Model-Based statistics for each dataset, the sum of individual statistics is the final statistic. The significance is determined via a permutation test which computed the same statistics and sums them after permuting the values across treatment groups. As is outlined in Karpievitch et al. 2018. Important to set the random number generator seed for reproducibility with set.seed() function.

**Usage**

prot_level_multi_part(mm_list, treat, prot.info, prot_col_name,
    nperm = 500, dataset_suffix)

**Arguments**

- **mm_list**: list of matrices for each experiment, length = number of datasets to compare internal dataset dimensions: numpeptides x numsamples for each dataset
- **treat**: list of data frames with treatment information to compute the statistic in same order as mm_list
- **prot.info**: list of protein and peptide mapping for each matrix in mm_list, in same order as mm_list
prot_col_name column name in prot.info that contains protein identifiers that link all datasets together. Not that Protein IDs will differ across different organisms and cannot be used as the linking identifier. Function match_linker_ids() produces numeric identifiers that link all datasets together.

nperm number of permutations, default = 500, this will take a while, test code with fewer permutations.

dataset_suffix vector of character strings that corresponds to the dataset being analysed. Same length as mm_list. Names will be appended to the columns names that will be generated for each analysed dataset. For example, if analysing mouse and human data this vector may be: c(‘Mouse’, ‘Human’)

Value
data frame with the following columns:

- **protIDused** Column containing the protein IDs used to link proteins across datasets
- **FC** Average fold change across all datasets
- **P_val** Permutation-based p-val for the differences between the groups
- **BH_P_val** Multiple testing adjusted p-values
- **statistic** Statistic computed as a sum of statistics produced for each dataset
- **Protein Information** all columns passed into the function for the 1st dataset in the list
- **FCs** Fold changes for individual datasets, these values should average to the FC above. As many columns as there are datasets being analyzed.
- **PV** p-values for individual datasets. As many columns as there are datasets being analyzed.
- **BHPV** Multiple testing adjusted p-values for individual datasets. As many columns as there are datasets being analyzed.
- **NUMPEP** Number of peptides presents in each protein for each dataset. As many columns as there are datasets being analyzed.

Examples

# Load mouse dataset
data(mm_peptides)
head(mm_peptides)

intsCols = 8:13 # different from parameter names as R uses
# outer names if variable is undefined
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(mm_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c(‘CG’, ‘CG’, ‘CG’, ‘mCG’, ‘mCG’, ‘mCG’))
set.seed(135)
mm_m_ints_eig1 = eig_norm1(m=m_logInts,treatment=grps, prot.info=m_prot.info)
mm_m_ints_eig1$h.c # check the number of bias trends detected
mm_m_ints_norm = eig_norm2(rv=mm_m_ints_eig1)
mm_prot.info = mm_m_ints_norm$normalized[,1:7]
mm_norm_m = mm_m_ints_norm$normalized[,8:13]
set.seed(125) # Needed for reproducibility of results
imp_mm = MBimpute(mm_norm_m, grps, prot.info=mm_prot.info,
pr_ppos=2, my.pi=0.05, compute_pi=FALSE)

# Load human dataset
data(hs_peptides)
head(hs_peptides)
intsCols = 8:13 # different from parameter names as R uses
# outer name spaces if variable is undefined
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(hs_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(hs_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG'))
set.seed(1237) # needed for reproducibility
hs_m_ints_eig1 = eig_norm1(m=m_logInts,treatment=grps,prot.info=m_prot.info)
hs_m_ints_eig1$h.c # check the number of bias trends detected
hs_m_ints_norm = eig_norm2(rv=hs_m_ints_eig1)
hs_prot.info = hs_m_ints_norm$normalized[,1:7]
hs_norm_m = hs_m_ints_norm$normalized[,8:13]
set.seed(125) # or any value, ex: 12345
imp_hs = MBimpute(hs_norm_m, grps, prot.info=hs_prot.info,
pr_ppos=2, my.pi=0.05,
compute_pi=FALSE)

# Multi-Matrix Model-based differential expression analysis
# Set up needed variables
mms = list()
treats = list()
protinfos = list()
mms[[1]] = imp_mm$y_imputed
mms[[2]] = imp_hs$y_imputed
treats[[1]] = grps
treats[[2]] = grps
protinfos[[1]] = imp_mm$imp_prot.info
protinfos[[2]] = imp_hs$imp_prot.info
nperm = 50

# ATTENTION: SET RANDOM NUMBER GENERATOR SEED FOR REPRODUCIBILITY !!
set.seed(131) # needed for reproducibility
comb_MBDE = prot_level_multi_part(mm_list=mms, treat=treats,
prot.info=protinfos,
prot_col_name='ProtID', nperm=nperm,
dataset_suffix=c('MM', 'HS'))

# Analysis for proteins only present in mouse,
# there are no proteins suitable for
# Model-Based analysis in human dataset
subset_data = subset_proteins(mm_list=mms, prot.info=protinfos, 'MatchedID')
mm_dd_only = subset_data$sub_unique_mm_list[[1]]
subset_proteins

```
hs_dd_only = subset_data$sub_unique_mm_list[[2]]
protinfos_mm_dd = subset_data$sub_unique_prot.info[[1]]
DE_mCG_GG_mm_dd = peptideLevel_DE(mm_dd_only, grps,
    prot.info=protinfos_mm_dd, pr_ppos=2)
```

---

**subset_proteins**  
**Subset proteins**

**Description**

Subset proteins into ones common to all datasets passed into the function and unique to each dataset. Note: for 3+ datasets no intermediate combinations of proteins are returned, only proteins common to all datasets, the rest are returned as unique to each dataset.

**Usage**

```
subset_proteins(mm_list, prot.info, prot_col_name)
```

**Arguments**

- **mm_list**  
  list of matrices for each experiment, length = number of datasets to compare internal dataset dimensions: numpeptides x numsamples for each dataset

- **prot.info**  
  list of protein and peptide mapping for each matrix in mm_list, in same order as mm_list

- **prot_col_name**  
  column name in prot.info that contains protein identifiers that link all datasets together. Not that Protein IDs will differ across different organisms and cannot be used as the linking identifier. Function match_linker_ids() produces numeric identifiers that link all datasets together

**Value**

data frame with the following columns

- **sub_mm_list**  
  list of dataframes of intensities for each of the datasets passed in with proteins present in all datasets

- **sub_prot.info**  
  list of dataframes of metadata for each of the datasets passed in with proteins present in all datasets. Same order as sub_mm_list

- **sub_unique_mm_list**  
  list of dataframes of intensities not found in all datasets

- **sub_unique_prot.info**  
  list of dataframes of metadata not found in all datasets

- **common_list**  
  list of protein IDs common to all datasets
Examples

```r
# Load mouse dataset
data(mm_peptides)
head(mm_peptides)
# different from parameter names as R uses
# outer name spaces if variable is undefined
intsCols = 8:13
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(mm_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(mm_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG'))
set.seed(173)
mm_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
nm_m_ints_norm = eig_norm2(rv=mm_m_ints_eig1)
mm_prot.info = mm_m_ints_norm$normalized[,1:7]
mm_norm_m = mm_m_ints_norm$normalized[,8:13]
set.seed(131)
imp_mm = MBimpute(mm_norm_m, grps,
    prot.info=mm_prot.info, pr_ppos=2, my.pi=0.05,
    compute.pi=FALSE)

# Load human dataset
data(hs_peptides)
head(hs_peptides)
intsCols = 8:13
metaCols = 1:7 # reusing this variable
m_logInts = make_intencities(hs_peptides, intsCols) # will reuse the name
m_prot.info = make_meta(hs_peptides, metaCols)
m_logInts = convert_log2(m_logInts)
grps = as.factor(c('CG', 'CG', 'CG', 'mCG', 'mCG'))
hs_m_ints_eig1 = eig_norm1(m=m_logInts, treatment=grps, prot.info=m_prot.info)
hs_m_ints_norm = eig_norm2(rv=hs_m_ints_eig1)
hs_prot.info = hs_m_ints_norm$normalized[,1:7]
hs_norm_m = hs_m_ints_norm$normalized[,8:13]
set.seed(131)
imp_hs = MBimpute(hs_norm_m, grps,
    prot.info=hs_prot.info, pr_ppos=2,
    my.pi=0.05,
    compute.pi=FALSE)

# Multi-Matrix Model-based differential expression analysis
# Set up needed variables
mms = list()
treats = list()
protinfos = list()
mms[[1]] = imp_mm$y_imputed
mms[[2]] = imp_hs$y_imputed
treats[[1]] = grps
treats[[2]] = grps
```

protinfos[[1]] = imp_mm$imp_prot.info
protinfos[[2]] = imp_hs$imp_prot.info

subset_data = subset_proteins(mm_list=mms, prot.info=protinfos, 'MatchedID')
mms_mm_dd = subset_data$sub_unique_mm_list[[1]]
protinfos_mm_dd = subset_data$sub_unique_prot.info[[1]]
# Differential expression analysis for mouse specific proteins
DE_mCG_CG_mm_dd = peptideLevel_DE(mms_mm_dd, grps,
                                  prot.info=protinfos_mm_dd, pr_ppos=2)

sva.id

Surrogate Variable Analysis

Description

Surrogate Variable Analysis function used internally by eig_norm1 and eig_norm2. Here we incorporate the model matrix from EigenMS normalization to find the significant trends in the matrix of residuals.

Usage

sva.id(dat, n.u.treatment, lm.fm, B = 500, sv.sig = 0.05)

Arguments

dat number of peptides/genes x number of samples matrix of expression data with no missing values
n.u.treatment number of treatment groups
lm.fm formular for treatment to be use on the right side of the call to stats::lm() as generated by makeLMFormula()
B The number of null iterations to perform
sv.sig The significance cutoff for the surrogate variables

Value

A data structure with the following values:

nsv Number of significant surrogate variables
psv Significance for the returned surrogate variables
Index

* datasets
  * hs_peptides, 9
  * mm_peptides, 13
  * convert_log2, 2
  * eig_norm1, 4
  * eig_norm2, 5
  * eigen_pi, 3
  * g.test, 6
  * get_presAbs_prots, 7
  * hs_peptides, 9
  * make_intencities, 10
  * make_meta, 11
  * makeLMFormula, 10
  * MBimpute, 12
  * mm_peptides, 13
  * peptideLevel_DE, 14
  * peptideLevel_PresAbsDE, 15
  * plot_1prot, 17
  * plot_3_pep_trends_NOfile, 18
  * plot_volcano, 19
  * plot_volcano_wLab, 20
  * prot_level_multi_part, 24
  * prot_level_multiMat_PresAbs, 21
  * subset_proteins, 27
  * sva.id, 29