Package ‘PhosR’

March 1, 2021

Type Package

Title A set of methods and tools for comprehensive analysis of phosphoproteomics data

Version 1.0.0

Description PhosR is a package for the comprehensive analysis of phosphoproteomic data. There are two major components to PhosR: processing and downstream analysis. PhosR consists of various processing tools for phosphoproteomics data including filtering, imputation, normalisation, and functional analysis for inferring active kinases and signalling pathways.

License GPL-3 + file LICENSE

Depends R (>= 4.0.0)

Imports ruv, e1071, calibrate, dendextend, limma, pcaMethods, stats, RColorBrewer, circlize, dplyr, igraph, heatmap, preprocessCore, tidyr, rlang, graphics, grDevices, utils

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

biocViews Software, ResearchField, SystemsBiology

Suggests testthat, knitr, GGally, network, reshape2, ClueR, directPA, ggplot2, ggpubr

git_url https://git.bioconductor.org/packages/PhosR

git_branch RELEASE_3_12

git_last_commit 39a78ff

git_last_commit_date 2020-10-27

Date/Publication 2021-02-28

Author Pengyi Yang [aut], Taiyun Kim [aut, cre], Jieun Hani Kim [aut]

Maintainer Taiyun Kim <taiyun.kim@sydney.edu.au>
createFrequencyMat

Create frequency matrix

Create frequency matrix
**frequencyScoring**

**Usage**

createFrequencyMat(substrates.seq)

**Arguments**

substrates.seq  A substrate sequence

**Value**

A frequency matrix of amino acid from substrates.seq.

**Examples**

data("phospho_L6_ratio")

# We will create a frequency matrix of Tfg S198 phosphosite.
rownames(phospho.L6.ratio)[1]
substrate.seq = unlist(lapply(strsplit(rownames(phospho.L6.ratio)[1],
    split = "-"), function(i) i[4]))
freq.mat = createFrequencyMat(substrate.seq)

---

**frequencyScoring**

**Frequency scoring**

**Description**

Frequency scoring

**Usage**

frequencyScoring(sequence.list, frequency.mat)

**Arguments**

sequence.list  A vector list of sequences
frequency.mat  A matrix output from 'createFrequencyMat'

**Value**

A vector of frequency score

**Examples**

data('phospho_L6_ratio')
data('KinaseMotifs')

# Extracting first 10 sequences for demonstration purpose
seqs = unlist(lapply(strsplit(rownames(phospho.L6.ratio), "-"),
    function(i) {i[4]}))
seqs = seqs[1:10]
# extracting flanking sequences
seqWin = mapply(function(x) {
    mid <- (nchar(x)+1)/2
    substr(x, start=(mid-7), stop=(mid+7))
}, seqs)

# The first 10 for demonstration purpose
phospho.L6.ratio = phospho.L6.ratio[1:10,]

# minimum number of sequences used for compiling motif for each kinase.
numMotif=5
motif.mouse.list.filtered <-
    motif.mouse.list[which(motif.mouse.list$NumInputSeq >= numMotif)]

# scoring all phosphosites against all motifs
motifScoreMatrix <-
    matrix(NA, nrow=nrow(phospho.L6.ratio),
           ncol=length(motif.mouse.list.filtered))
rownames(motifScoreMatrix) <- rownames(phospho.L6.ratio)
colnames(motifScoreMatrix) <- names(motif.mouse.list.filtered)

# Scoring phosphosites against kinase motifs
for(i in seq_len(length(motif.mouse.list.filtered))) {
    motifScoreMatrix[,i] <-
        frequencyScoring(seqWin, motif.mouse.list.filtered[[i]])
    cat(paste(i, '.', sep=''))
}

### KinaseFamily

<table>
<thead>
<tr>
<th>KinaseFamily</th>
<th>KinaseFamily</th>
</tr>
</thead>
</table>

**Description**

A summary table of kinase family

**Usage**

```r
data(KinaseFamily)
```

**Format**

An object of class `matrix` (inherits from `array` with 425 rows and 6 columns.
**kinaseSubstrateHeatmap**

**Kinase-substrate annotation prioritisation heatmap**

---

**Description**

Kinase-substrate annotation prioritisation heatmap

**Usage**

`kinaseSubstrateHeatmap(phosScoringMatrices, top = 3)`

**Arguments**

- `phosScoringMatrices`  
  a matrix returned from kinaseSubstrateScore.

- `top`  
  the number of top ranked phosphosites for each kinase to be included in the heatmap. Default is 1.

**Value**

a pheatmap object.

**Examples**

data('phospho_L6_ratio')
data('SPSs')
data('PhosphoSitePlus')

grps = gsub('_.+\.', ',', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', ',', sapply(strsplit(rownames(phospho.L6.ratio), '\~'),
  function(x){paste(toupper(x[2]), x[3], ',', sep=';')}))

phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
  colMeans))

phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)

phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
  ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6,
   grps = gsub('_.+', '', colnames(phosphoL6)))
aov <- matANOVA(mat=phosphoL6, grps=gsub('_.+', '', colnames(phosphoL6)))
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
   (rowSums(phosphoL6.mean > 0.5) > 5),,drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
rownames(L6.phos.std) <- sapply(strsplit(rownames(L6.phos.std), '-'),
   function(x){gsub('/', '/', paste(toupper(x[2]), x[3], '/', sep=';'))})
L6.phos.seq <- sapply(strsplit(rownames(phosphoL6.reg), '-'),
   function(x)x[4])
L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
   L6.phos.seq, numMotif = 5, numSub = 1)
kinaseSubstrateHeatmap(L6.matrices)

Description
A machine learning approach for predicting specific kinase for a given substrate. This prediction
framework utilise adaptive sampling.

Usage
kinaseSubstratePred(  
   phosScoringMatrices,  
   ensembleSize = 10,  
   top = 50,  
   cs = 0.8,  
   inclusion = 20,  
   iter = 5  )

Arguments
phosScoringMatrices An output of kinaseSubstrateScore.
ensembleSize An ensemble size.
top a number to select top kinase substrates.
sc Score threshold.
inclusion A minimal number of substrates required for a kinase to be selected.
iter A number of iterations for adaSampling.

Value
Kinase prediction matrix
Examples

data('phospho_L6_ratio')
data('SPSs')

grps = gsub('_+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '-'),
          function(x){paste(toupper(x[2]), x[3], '', sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
            colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
                                      ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6)

aov <- matANOVA(mat=phosphoL6, grps=gsub('_+', '', colnames(phosphoL6))
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
                       (rowSums(phosphoL6.mean > 0.5) > 0),,drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
rownames(L6.phos.std) <- sapply(strsplit(rownames(L6.phos.std), '-'),
                        function(x){gsub(' ', '', paste(toupper(x[2]), x[3], '', sep=';'))})

L6.phos.seq <- sapply(strsplit(rownames(phosphoL6.reg), '-'),
                        function(x)x[4])

L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
                                      L6.phos.seq, numMotif = 5, numSub = 1)
set.seed(1)
L6.predMat <- kinaseSubstratePred(L6.matrices, top=30)

kinaseSubstrateProfile

Kinase substrate profiling

Description

This function generates substrate profiles for kinases that have one or more substrates quantified in
the phosphoproteome data.
**Usage**

kinaseSubstrateProfile(substrate.list, mat)

**Arguments**

substrate.list  a list of kinases with each element containing an array of substrates.

mat  a matrix with rows correspond to phosphosites and columns correspond to samples.

**Value**

Kinase profile list.

**Examples**

data('phospho_L6_ratio')
data('SPSs')

grps = gsub('\.+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub('\s+', '', sapply(strsplit(rownames(phospho.L6.ratio), '\-'), function(x){paste(toupper(x[2]), x[3], '', sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites), colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6,
grps = gsub('\.+', '', colnames(phosphoL6)))
aov <- matANOVA(mat=phosphoL6, grps=gsub('\.+', '', colnames(phosphoL6)))
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
(rowSums(phosphoL6.mean > 0.5) > 0),,drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
rownames(L6.phos.std) <- sapply(strsplit(rownames(L6.phos.std), '\-'),
function(x){gsub('\s+', '', paste(toupper(x[2]), x[3], '', sep=';')}))

ks.profile.list <- kinaseSubstrateProfile(PhosphoSite.mouse, L6.phos.std)
kinaseSubstrateScore  

Description

This function generates substrate scores for kinases that pass filtering based on both motifs and dynamic profiles

Usage

kinaseSubstrateScore(substrate.list, mat, seqs, numMotif = 5, numSub = 1)

Arguments

substrate.list  a list of kinases with each element containing an array of substrates.
mat  a matrix with rows correspond to phosphosites and columns correspond to samples.
seqs  an array containing aa sequences surrounding each of all phosphosites. Each sequence has length of 15 (-7, p, +7).
numMotif  minimum number of sequences used for compiling motif for each kinase. Default is 5.
numSub  minimum number of phosphosites used for compiling phosphorylation profile for each kinase. Default is 1.

Value

A list of 4 elements. motifScoreMatrix, profileScoreMatrix, combinedScoreMatrix, ksActivityMatrix (kinase activity matrix) and their weights.

Examples

data('phospho_L6_ratio')
data('SPSs')
data('PhosphoSitePlus')

grps = gsub('_.+', ' ', colnames(phospho.L6.ratio))
# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', ',', sapply(strsplit(rownames(phospho.L6.ratio), '~'),
function(x){paste(toupper(x[2]), x[3], ' ', sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
matANOVA

Description
Performs an ANOVA test and returns its adjusted p-value

Usage
matANOVA(mat, grps)

Arguments
mat
An p by n matrix where p is the number of phosphosites and n is the number of samples

grps
A vector of length n, with group or time point information of the samples

Value
A vector of multiple testing adjusted p-values

Examples
data('phospho_L6_ratio')
data('SPSs')
grps = gsub('_+',' ', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
meanAbundance

L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '\-'),
    function(x){paste(toupper(x[2]), x[3], '',
        sep=';')})
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
    colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
    ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = gsub('_.+', '',
    colnames(phosphoL6)))
aov <- matANova(mat=phosphoL6, grps=gsub('_.+', '', colnames(phosphoL6)))

meanAbundance

Obtain average expression from replicates

Description
Obtain average expression from replicates

Usage
meanAbundance(mat, grps)

Arguments

mat          a matrix with rows correspond to phosphosites and columns correspond to samples.
grps         a string specifying the grouping (replicates).

Value
a matrix with mean expression from replicates

Examples
data('phospho_L6_ratio')
data('SPSs')
ggrps = gsub('_.+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
medianScaling

Description
Median centering and scaling of an input numeric matrix

Usage
medianScaling(mat, scale = TRUE, grps = NULL, reorder = FALSE)

Arguments
- mat: a matrix with rows correspond to phosphosites and columns correspond to samples.
- scale: a boolean flag indicating whether to scale the samples.
- grps: a string or factor specifying the grouping (replicates).
- reorder: whether to reorder by factor.

Value
A median scaled matrix

phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '-'),
  function(x){paste(toupper(x[2]), x[3], '', sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
  colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
  ctl = ctl)

phosphol6 = phospho.L6.ratio.RUV
rownames(phosphol6) = phospho.site.names

# filter for up-regulated phosphosites
phosphol6.mean <- meanAbundance(phosphol6, grps = gsub('.+\.', '',
  colnames(phosphol6)))
Examples

data('phospho.cells.Ins.sample')
grps = gsub('^[0-9]!', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
    scImpute(phospho.cells.Ins.filtered, 0.5, grps)[,colnames(phospho.cells.Ins.filtered)]

set.seed(123)
phospho.cells.Ins.impute[,1:5] <- ptiImpute(phospho.cells.Ins.impute[,6:10],
    phospho.cells.Ins.impute[,1:5], percent1 = 0.6,
    percent2 = 0, paired = FALSE)

phospho.cells.Ins.ms <-
    medianScaling(phospho.cells.Ins.impute, scale = FALSE)

minmax

Minmax scaling

Description

Perform a minmax standardisation to scale data into 0 to 1 range

Usage

minmax(mat)

Arguments

mat

A matrix with rows correspond to phosphosites and columns correspond to condition

Value

Minmax standardised matrix

Examples

data('phospho_L6.ratio')
data('SPSs')
grps = gsub('_.+',' ', colnames(phospho.L6.ratio))
# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ','', sapply(strsplit(rownames(phospho.L6.ratio), '~'),
    function(x){paste(toupper(x[2]), x[3], '}

minmax
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites), colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3, ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = gsub('_\+','', colnames(phosphoL6)))
aov <- matANOVA(mat=phosphoL6, grps=gsub('_\+','', colnames(phosphoL6)))
phosphoL6.reg <- phosphoL6[(aov < 0.05) & (rowSums(phosphoL6.mean > 0.5) > 0), drop = FALSE]

L6.phos.std <- standardise(phosphoL6.reg)
rownames(L6.phos.std) <- sapply(strsplit(rownames(L6.phos.std), '\~'), function(x){gsub('/\+/', '/\+', paste(toupper(x[2]), x[3], '/\+', sep=';\'))})

L6.phos.seq <- sapply(strsplit(rownames(phosphoL6.reg), '\~'), function(x)x[4])

numMotif = 5
numSub = 1

ks.profile.list <- kinaseSubstrateProfile(PhosphoSite.mouse, L6.phos.std)
motif.mouse.list = PhosR::motif.mouse.list

motif.mouse.list.filtered <- motif.mouse.list[which(motif.mouse.list$NumInputSeq >= numMotif)]
ks.profile.list.filtered <- ks.profile.list[which(ks.profile.list$NumSub >= numSub)]

# scoring all phosphosites against all motifs
motifScoreMatrix <- matrix(NA, nrow=nrow(L6.phos.std), ncol=length(motif.mouse.list.filtered))
rownames(motifScoreMatrix) <- rownames(L6.phos.std)
colnames(motifScoreMatrix) <- names(motif.mouse.list.filtered)

# extracting flanking sequences
seqWin = mapply(function(x) {
  mid <- (nchar(x)+1)/2
  substr(x, start=(mid-7), stop=(mid+7))
}, L6.phos.seq)

print('Scoring phosphosites against kinase motifs:
for(i in seq_len(length(motif.mouse.list.filtered))) {
motifScoreMatrix[,i] <-
')
frequencyScoring(seqWin, motif.mouse.list.filtered[[i]])
cat(paste(i, ',', sep=''))
}
motifScoreMatrix <- minmax(motifScoreMatrix)

mIntersect  Multi-intersection, union

Description
A recursive loop for intersecting multiple sets.

Usage
mIntersect(x, y, ...)
mIntersect(x, y, ...)
mUnion(x, y, ...)

Arguments
x, y, ... objects to find intersection/union.

Value
An intersection/union of input parameters

Examples

data('phospho_liverInsTC_RUV_sample')
data('phospho_L6_ratio')
site1 <- gsub('^[STY]', '-',
sapply(strsplit(rownames(phospho.L6.ratio), '-'),
function(x)(paste(toupper(x[2]), x[3], sep='-', sep=''))))
site2 <- rownames(phospho.liver.Ins.TC.ratio.RUV)

# step 2: rank by fold changes
tmp <- do.call(cbind, lapply(split(1:ncol(phospho.L6.ratio), gsub('_exp\d+', '', colnames(phospho.L6.ratio))),
function(i){rowMeans(phospho.L6.ratio[,i])}))
site1 <- t(sapply(split(data.frame(tmp), site1), colMeans))[-1]
tmp <- do.call(cbind, lapply(split(1:ncol(phospho.liver.Ins.TC.ratio.RUV),
gsub(     'Intensity\..(.*)\_Bio\d+','\"2',
colnames(phospho.liver.Ins.TC.ratio.RUV))),
function(i){rowMeans(phospho.liver.Ins.TC.ratio.RUV[,i])}))
site2 <- t(sapply(split(data.frame(tmp), site2), colMeans))
motif.rat.list

o <- mIntersect(site1, site2)

motif.human.list  List of human kinase motifs

Description
A list of human kinase motifs and their sequence probability matrix.

Usage
data(KinaseMotifs)

Format
An object of class list of length 380.

motif.mouse.list  List of mouse kinase motifs

Description
A list of mouse kinase motifs and their sequence probability matrix.

Usage
data(KinaseMotifs)

Format
An object of class list of length 250.

motif.rat.list  List of rat kinase motifs

Description
A list of rat kinase motifs and their sequence probability matrix.

Usage
data(KinaseMotifs)

Format
An object of class list of length 159.
**pathwayOverrepresent**  
*Gene set over-representation analysis*

### Description

This function performs gene set over-representation analysis using Fisher’s exact test.

### Usage

```r
pathwayOverrepresent(geneSet, annotation, universe, alter = "greater")
```

### Arguments

- `geneSet`: an array of gene or phosphosite IDs (IDs are gene symbols etc that match to your pathway annotation list).
- `annotation`: a list of pathways with each element containing an array of gene or phosphosite IDs.
- `universe`: the universe/background of all genes or phosphosites in your profiled dataset.
- `alter`: test for enrichment (’greater’, default), depletion (’less’), or ’two.sided’.

### Value

A matrix of pathways and their associated substrates and p-values.

### Examples

```r
library(limma)
data('phospho.L6.ratio')
data('SPSs')
grps = gsub('_', '+', colnames(phospho.L6.ratio))
# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '\~'),
  function(x){paste(toupper(x[2]), x[3], '','
    sep='';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)
# Construct a design matrix by condition
design = model.matrix(~ grps - 1)
# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
ctl = ctl)
# divides the phospho.L6.ratio data into groups by phosphosites
```
L6.sites <- gsub(' ','\ ', gsub('-[STY]', '\-', sapply(strsplit(rownames(phospho.L6.ratio.RUV), '-'), function(x){paste(toupper(x[2]), x[3], sep='--')})))
phospho.L6.ratio.sites <- t(sapply(split(data.frame(phospho.L6.ratio.RUV), L6.sites), colMeans))

# fit linear model for each phosphosite
f <- gsub('_exp\d', '\ ', colnames(phospho.L6.ratio.RUV))
X <- model.matrix(~ f - 1)
fit <- lmFit(phospho.L6.ratio.RUV, X)

# extract top-ranked phosphosites for each condition compared to basal
table.AICAR <- topTable(eBayes(fit), number=Inf, coef = 1)
table.Ins <- topTable(eBayes(fit), number=Inf, coef = 3)
table.AICARIns <- topTable(eBayes(fit), number=Inf, coef = 2)

DE1.RUV <- c(sum(table.AICAR[, 'adj.P.Val'] < 0.05),
sum(table.Ins[, 'adj.P.Val'] < 0.05),
sum(table.AICARIns[, 'adj.P.Val'] < 0.05))

# extract top-ranked phosphosites for each group comparison
contrast.matrix1 <- makeContrasts(fAICARIns-fIns, levels=X)
contrast.matrix2 <- makeContrasts(fAICARIns-fAICAR, levels=X)
fit1 <- contrasts.fit(fit, contrast.matrix1)
fit2 <- contrasts.fit(fit, contrast.matrix2)
table.AICARInsVSIns <- topTable(eBayes(fit1), number=Inf)
table.AICARInsVSAICAR <- topTable(eBayes(fit2), number=Inf)

DE2.RUV <- c(sum(table.AICARInsVSIns[, 'adj.P.Val'] < 0.05),
sum(table.AICARInsVSAICAR[, 'adj.P.Val'] < 0.05))

o <- rownames(table.AICARInsVSIns)
Tc <- cbind(table.Ins[o, 'logFC'], table.AICAR[o, 'logFC'],
table.AICARIns[o, 'logFC'])
rownames(Tc) = gsub('.+([A-Z]+)([0-9]+);', '\1;\3;', o)
colnames(Tc) <- c('Ins', 'AICAR', 'AICAR+Ins')

# summary phosphosite-level information to proteins for performing downstream
# gene-centric analyses.
Tc.gene <- phosCollapse(Tc, id=gsub('.+\ ', '\ ', rownames(Tc)),
stat=apply(abs(Tc), 1, max), by = 'max')
geneSet <- names(sort(Tc.gene[,1], decreasing = TRUE))[:round(nrow(Tc.gene) * 0.1)]
lapply(PhosphoSite.rat, function(x){gsub('[STY]', '\ ', x)})

# 1D gene-centric pathway analysis
path1 <- pathwayOverrepresent(geneSet, annotation=Pathways.reactome,
universe = rownames(Tc.gene), alter = 'greater')
## pathwayRankBasedEnrichment

### Description

This function performs gene set enrichment analysis using Wilcoxon Rank Sum test.

### Usage

```r
pathwayRankBasedEnrichment(geneStats, annotation, alter = "greater")
```

### Arguments

- **geneStats**: an array of statistics (e.g. log2 FC) of all quantified genes or phosphosite with names of the array as gene or phosphosite IDs.
- **annotation**: a list of pathways with each element containing an array of gene IDs.
- **alter**: test for enrichment ('greater', default), depletion ('less'), or 'two.sided'.

### Value

A matrix of pathways and their associated substrates and p-values.

### Examples

```r
library(limma)
data('phospho_L6_ratio')
data('SPSs')
grps = gsub('_\.+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub('[STY]', '\~', sapply(strsplit(rownames(phospho.L6.ratio), '~'),
  function(x){paste(toupper(x[2]), x[3], '\~', sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
  colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
  ctl = ctl)

# divides the phospho.L6.ratio data into groups by phosphosites
L6.sites <- gsub('_\d', '', gsub('^-[STY]', '-',
  sapply(strsplit(rownames(phospho.L6.ratio.RUV), '-'),
  function(x){paste(toupper(x[2]), x[3], sep='-'')})))
phospho.L6.ratio.sites <- t(sapply(split(data.frame(phospho.L6.ratio.RUV),
  L6.sites), colMeans))

# fit linear model for each phosphosite
f <- gsub('_exp\d', '', colnames(phospho.L6.ratio.RUV))
```
\texttt{X <- model.matrix(~ f - 1)}
\texttt{fit <- lmFit(phospho.L6.ratio.RUV, X)}

\texttt{# extract top-ranked phosphosites for each condition compared to basal}
\texttt{table.AICAR <- topTable(eBayes(fit), number=Inf, coef = 1)}
\texttt{table.Ins <- topTable(eBayes(fit), number=Inf, coef = 3)}
\texttt{table.AICARIns <- topTable(eBayes(fit), number=Inf, coef = 2)}

\texttt{DE1.RUV <- c(sum(table.AICAR[, \textquotesingle{}adj.P.Val\textquotesingle{}]) < 0.05),}
\texttt{sum(table.Ins[, \textquotesingle{}adj.P.Val\textquotesingle{}]) < 0.05),}
\texttt{sum(table.AICARIns[, \textquotesingle{}adj.P.Val\textquotesingle{}]) < 0.05))}

\texttt{# extract top-ranked phosphosites for each group comparison}
\texttt{contrast.matrix1 <- makeContrasts(fAICARIns-fIns, levels=X)}
\texttt{contrast.matrix2 <- makeContrasts(fAICARIns-fAICAR, levels=X)}
\texttt{fit1 <- contrasts.fit(fit, contrast.matrix1)}
\texttt{fit2 <- contrasts.fit(fit, contrast.matrix2)}
\texttt{table.AICARInsVSIns <- topTable(eBayes(fit1), number=Inf)}
\texttt{table.AICARInsVSAICAR <- topTable(eBayes(fit2), number=Inf)}

\texttt{DE2.RUV <- c(sum(table.AICARInsVSIns[, \textquotesingle{}adj.P.Val\textquotesingle{}]) < 0.05),}
\texttt{sum(table.AICARInsVSAICAR[, \textquotesingle{}adj.P.Val\textquotesingle{}]) < 0.05))}

\texttt{o <- rownames(table.AICARInsVSIns)}
\texttt{Tc <- cbind(table.Ins[o, \textquotesingle{}logFC\textquotesingle{}], table.AICAR[o, \textquotesingle{}logFC\textquotesingle{}],}
\texttt{table.AICARIns[o, \textquotesingle{}logFC\textquotesingle{}])}
\texttt{rownames(Tc) = gsub('(\d+)([A-Z])', '\\1;\3;', o)}
\texttt{colnames(Tc) <- c('Ins', 'AICAR', 'AICAR+Ins')}

\texttt{# summary phosphosite-level information to proteins for performing downstream}
\texttt{# gene-centric analyses.}
\texttt{Tc.gene <- phosCollapse(Tc, id=gsub(':', '+', '\', rownames(Tc)),}
\texttt{stat=apply(abs(Tc), 1, max), by = 'max')}

\texttt{# 1D gene-centric pathway analysis}
\texttt{path2 <- pathwayRankBasedEnrichment(Tc.gene[,1],}
\texttt{annotation=Pathways.reactome,}
\texttt{alter = 'greater'})

---

**Pathways.KEGG**

**KEGG pathway annotations**

**Description**

The data object contains the annotations of KEGG pathways.

**Usage**

\texttt{data(Pathways)}

**Format**

An object of class list of length 186.
Pathways.reactome

Pathways.reactome Reactome pathway annotations

Description
The data object contains the annotations of Reactome pathways.

Usage
data(Pathways)

Format
An object of class list of length 674.

phosCollapse Summarising phosphosites to proteins

Description
Summarising phosphosite-level information to proteins for performing downstream gene-centric analyses.

Usage
phosCollapse(mat, id, stat, by='min')

Arguments
mat a matrix with rows correspond to phosphosites and columns correspond to samples.
id an array indicating the groupping of phosphosites etc.
stat an array containing statistics of phosphosite such as phosphorylation levels.
by how to summarise phosphosites using their statistics. Either by 'min' (default), 'max', or 'mid'.

Value
A matrix summarised to protein level
library(limma)
data('phospho_L6_ratio')
data('SPSs')

grps = gsub('_\.+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '\-'),
    function(x){paste(toupper(x[2]), x[3], '', sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
    colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
    ctl = ctl)

# divides the phospho.L6.ratio data into groups by phosphosites
L6.sites <- gsub(' ', '', gsub('[STY]', '-', sapply(strsplit(rownames(phospho.L6.ratio.RUV), '\-'),
    function(x){paste(toupper(x[2]), x[3], sep='')})
phospho.L6.ratio.sites <- t(sapply(split(data.frame(phospho.L6.ratio.RUV),
    L6.sites), colMeans))

# fit linear model for each phosphosite
f <- gsub('_exp\d', '', colnames(phospho.L6.ratio.RUV))
X <- model.matrix(~ f - 1)
fit <- lmFit(phospho.L6.ratio.RUV, X)

# extract top-ranked phosphosites for each condition compared to basal
table.AICAR <- topTable(eBayes(fit), number=Inf, coef = 1)
table.Ins <- topTable(eBayes(fit), number=Inf, coef = 3)
table.AICARIns <- topTable(eBayes(fit), number=Inf, coef = 2)
DE1.RUV <- c(sum(table.AICAR[, 'adj.P.Val'] < 0.05),
    sum(table.Ins[, 'adj.P.Val'] < 0.05),
    sum(table.AICARIns[, 'adj.P.Val'] < 0.05))

# extract top-ranked phosphosites for each group comparison
contrast.matrix1 <- makeContrasts(fAICARIns-fIns, levels=X)
contrast.matrix2 <- makeContrasts(fAICARIns-fAICAR, levels=X)
fit1 <- contrasts.fit(fit, contrast.matrix1)
fit2 <- contrasts.fit(fit, contrast.matrix2)
table.AICARInsVSIns <- topTable(eBayes(fit1), number=Inf)
table.AICARInsVSAICAR <- topTable(eBayes(fit2), number=Inf)
DE2.RUV <- c(sum(table.AICARInsVSIns[, 'adj.P.Val'] < 0.05),
    sum(table.AICARInsVSAICAR[, 'adj.P.Val'] < 0.05))
sum(table.AICARInsVSAICAR[, 'adj.P.Val'] < 0.05))

o <- rownames(table.AICARInsVSAICAR)
Tc <- cbind(table.Ins[as.character(o), 'logFC'], table.AICAR[as.character(o), 'logFC'])
rownames(Tc) = gsub('(.*)(;[A-Z])([0-9]+);', '\1;\3;', o)
colnames(Tc) <- c('Ins', 'AICAR', 'AICAR+Ins')

# summary phosphosite-level information to proteins for performing downstream
# gene-centric analyses.
Tc.gene <- phosCollapse(Tc, id=gsub(';', '+', rownames(Tc)),
    stat=apply(abs(Tc), 1, max), by = 'max')

description phospho.cells.Ins

phospho.cells.Ins

A subset of phosphoproteomics dataset generated by Humphrey et al., [doi:10.1038/nbt.3327] from
two mouse liver cell lines (Hepa1.6 and FL38B) that were treated with either PBS (mock) or insulin.

Usage
data(phospho.cells.Ins.sample)

Format
An object of class matrix (inherits from array) with 49617 rows and 24 columns.

Source
doi: 10.1038/nbt.3327 (PXD001792)

References
Humphrey et al., 2015, doi: 10.1038/nbt.3327

description phospho.L6.ratio

phospho.L6.ratio

An L6 myotube phosphoproteome dataset (accession number: PXD019127).

Usage
data(phospho_L6_ratio)

Format
An object of class matrix (inherits from array) with 6660 rows and 12 columns.
Source

PRIDE accession number: PXD001792

phospho.liver.Ins.TC.ratio.RUV

Description

A subset of phosphoproteomics dataset integrated from two time-course datasets of early and intermediate insulin signalling in mouse liver upon insulin stimulation.

Usage

data(phospho_liverInsTC_RUV_sample)

Format

An object of class matrix (inherits from array) with 5000 rows and 90 columns.

Source

PRIDE accession number: PXD001792

References

Humphrey et al., 2015

PhosphoSite.human

PhosphoSitePlus annotations for human

Description

The data object contains the annotations of kinases and their corresponding substrates as phosphorylation sites in human. It is extracted from the PhosphoSitePlus database. For details of PhosphoSitePlus, please refer to the article: Hornbeck et al. Nucleic Acids Res. 40:D261-70, 2012

Usage

data(PhosphoSitePlus)

Format

An object of class list of length 379.

Source

https://www.phosphosite.org
PhosphoSite.mouse

PhosphoSitePlus annotations for mouse

Description
The data object contains the annotations of kinases and their corresponding substrates as phosphorylation sites in mouse. It is extracted from the PhosphoSitePlus database. For details of PhosphoSitePlus, please refer to the article: Hornbeck et al. Nucleic Acids Res. 40:D261-70, 2012

Usage
data(PhosphoSitePlus)

Format
An object of class list of length 260.

Source
https://www.phosphosite.org

PhosphoSite.rat

PhosphoSitePlus annotations for rat

Description
The data object contains the annotations of kinases and their corresponding substrates as phosphorylation sites in rat. It is extracted from the PhosphoSitePlus database. For details of PhosphoSitePlus, please refer to the article: Hornbeck et al. Nucleic Acids Res. 40:D261-70, 2012

Usage
data(PhosphoSitePlus)

Format
An object of class list of length 158.

Source
https://www.phosphosite.org
plotQC

A set of function for data QC plot

Description

The 'panel' parameter allows different kind of visualisation for output object from PhosR. 'panel = 0' is used to create a 2*2 panel of plots including the following. 'panel = 1' is used to visualise percentage of quantification after imputation. 'panel = 2' is used to visualise dendrogram (hierarchical clustering) of the input matrix. 'panel = 3' is used to visualise abundance level of samples from the input matrix. 'panel = 4' is used to show PCA plot.

Usage

plotQC(mat, cols, labels, panel, ...)

Arguments

mat
A p by n matrix, where p is the number of phosphosites and n is the number of samples.

cols
A vector of colours to be used in the plot. The length should be equal to the columns of the mat.

labels
A vector of sample names. Used the label points in PCA plot (panel=4)

panel
A numeric value (0-4) to choose the plot type. See description for details.

... Plotting parameters for base plots

Value

A graphical plot

Examples

# Imputation
data('phospho.cells.Ins.sample')
grps = gsub('_([0-9]{1})', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
  scImpute(
    phospho.cells.Ins.filtered,
    0.5,
    grps[,colnames(phospho.cells.Ins.filtered)]

set.seed(123)
phospho.cells.Ins.impute[,1:5] <- ptImpute(phospho.cells.Ins.impute[,6:10],
phospho.cells.Ins.impute[,1:5], percent1 = 0.6, percent2 = 0, paired = FALSE)

phospho.cells.Ins.ms <- medianScaling(phospho.cells.Ins.impute,
  scale = FALSE)

cols <- rep(c('#ED4024', '#7FBF42', '#3F61AD', '#9B822F'), each=6)
par(mfrow=c(1,2))
plotQC(phospho.cells.Ins.filtered,
      labels=colnames(phospho.cells.Ins.filtered),
      panel = 1, cols = cols)
plotQC(phospho.cells.Ins.ms,
      labels=colnames(phospho.cells.Ins.ms),
      panel = 1, cols = cols)

# Batch correction
data('phospho_L6_ratio')
data('SPSs')

grps = gsub('_\+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '~'),
                      function(x){paste(toupper(x[2]), x[3], '',
                          sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
                          colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
                                   ctl = ctl)

cs = rainbow(length(unique(grps)))
colorCodes = sapply(grps, switch, AICAR=cs[1], Ins=cs[2], AICARIns=cs[3])

# plot after batch correction
par(mfrow=c(1,2))
plotQC(phospho.L6.ratio, panel = 2, cols=colorCodes)
plotQC(phospho.L6.ratio.RUV, cols=colorCodes,
      labels = colnames(phospho.L6.ratio),
      panel=2, ylim=c(-20, 20), xlim=c(-30, 30))

par(mfrow=c(1,2))
plotQC(phospho.L6.ratio, panel = 4, cols=colorCodes,
      labels = colnames(phospho.L6.ratio),
      main='Before Batch correction')
plotQC(phospho.L6.ratio.RUV, cols=colorCodes,
      labels = colnames(phospho.L6.ratio),
      panel=4, ylim=c(-20, 20), xlim=c(-30, 30),
      main='After Batch correction')
Description

Impute the missing values for mat2 using tail imputation approach if mat1 has more than percent1 (percentage) of quantified values and mat2 has less than percent2 (percentage) quantified values, and vice versa if paired is set to be true. That is if mat2 has percentage of quantified values more than percent1 and mat1 has percentage quantified values less than percent2.

Usage

ptImpute(mat1, mat2, percent1, percent2, m, s, paired)

Arguments

mat1  a matrix with rows correspond to phosphosites and columns correspond to replicates within treatment1.
mat2  a matrix with rows correspond to phosphosites and columns correspond to replicates within treatment2.
percent1  a percent indicating minimum quantified percentages required for considering for imputation.
percent2  a percent indicating minimum quantified percentages required for considering for imputation.
m  a numeric number of for controlling mean downshifting.
s  a numeric number of for controlling standard deviation of downshifted sampling values.
paired  a flag indicating whether to impute for both treatment1 and treatment2 (default) or treatment2 only (if paired=FALSE).

Value

An imputed matrix

Examples

data('phospho.cells.Ins.sample')
grps = gsub('_[0-9]{1}', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <-
    scImpute(
        phospho.cells.Ins.filtered,
        0.5,
        grps)[,colnames(phospho.cells.Ins.filtered)]

set.seed(123)
phospho.cells.Ins.impute[,1:5] <- ptImpute(phospho.cells.Ins.impute[,6:10],
phospho.cells.Ins.impute[,1:5], percent1 = 0.6, percent2 = 0, paired = FALSE)
RUVphospho

**Description**

This is a wrapper implementation of RUVIII for phosphoproteomics data normalisation. This function will call tailImpute function to impute all the missing values (if there is any) in the phosphoproteomics data for applying RUVIII. It will then return the normalised values for quantified phosphosites and remove imputed values.

**Usage**

RUVphospho(mat, M, ctl, k = NULL, m = 1.6, s = 0.6, keepImpute = FALSE, ...)

**Arguments**

- `mat`: a matrix with rows correspond to phosphosites and columns correspond to samples.
- `M`: is the design matrix as defined in RUVIII.
- `ctl`: is the stable phosphosites (or negative controls as defined in RUVIII).
- `k`: is the number of unwanted factors as defined in RUVIII.
- `m`: a numeric number for controlling mean downshifting.
- `s`: a numeric number for controlling standard deviation of downshifted sampling values.
- `keepImpute`: a boolean to keep the missing value in the returned matrix.
- `...`: additional parameters that may be passed to RUVIII.

**Value**

A normalised matrix.

**Examples**

data('phospho.L6.ratio')
data('SPSs')
grps = gsub('_.+\.', '', colnames(phospho.L6.ratio))
  # Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '-'),
  function(x){paste(toupper(x[2]), x[3], '', sep=';')}))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
  colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)
  # Construct a design matrix by condition
design = model.matrix(~ grps - 1)
# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
ctl = ctl)

scImpute

## Site- and condition-specific (sc) impute

### Description

Impute the missing values for a phosphosite across replicates within a single condition (or treatment) if there are n or more quantified values of that phosphosite in that condition.

### Usage

scImpute(mat, percent, grps)

### Arguments

- **mat**: a matrix with rows correspond to phosphosites and columns correspond to replicates within a condition.
- **percent**: a percent from 0 to 1, specifying the percentage of quantified values in any treatment group.
- **grps**: a string specifying the grouping (replicates).

### Value

An imputed matrix

### Examples

data('phospho.cells.Ins.sample')
grps = gsub('[^[0-9][+]]', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)
set.seed(123)
phospho.cells.Ins.impute <-
  scImpute(phospho.cells.Ins.filtered,
  0.5,
  grps[,colnames(phospho.cells.Ins.filtered)]
)
selectGrps

*Select by treatment groups (replicate block)*

**Description**

Select phosphosites that have been quantified in a given percentage of treatment groups (e.g. 0.75 as 3 out of 4 replicates) in n groups.

**Usage**

```r
selectGrps(mat, grps, percent, n)
```

**Arguments**

- `mat`: a matrix with rows correspond to phosphosites and columns correspond to samples in replicates for different treatments.
- `grps`: a string specifying the grouping (replicates).
- `percent`: a percent from 0 to 1, specifying the percentage of quantified values in any treatment group.
- `n`: an integer indicating n or more replicates pass the percentage filtering for a phosphosite to be included.

**Value**

a filtered matrix with at least `percent` quantification in one or more conditions

**Author(s)**

Pengyi Yang, Taiyun Kim

**Examples**

```r
data('phospho.cells.Ins.sample')
grps = gsub('_([0-9]+)', '', colnames(phospho.cells.Ins))

phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)
```

---

selectOverallPercent

*Select phosphosite by percentage of quantification*

**Description**

Select phosphosites that have been quantified in more than a given percentage of samples.

**Usage**

```r
selectOverallPercent(mat, percent=NULL, n=NULL)
```

**Examples**

```r
data('phospho.cells.Ins.sample')
grps = gsub('_([0-9]+)', '', colnames(phospho.cells.Ins))

phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)
```
selectTimes

**Arguments**

- `mat`: a matrix with rows correspond to phosphosites and columns correspond to samples in replicates for different treatments.
- `percent`: a percent from 0 to 1, specifying the percentage of quantified values in across all samples for retaining a phosphosite for subsequent analysis.
- `n`: an integer indicating n or more quantified values required for retaining a phosphosite for subsequent analysis.

**Value**

a filtered matrix

**Examples**

data('phospho.cells.Ins.sample')

phospho.cells.Ins.filtered <- selectOverallPercent(phospho.cells.Ins, 0.5)

# Before filtering
dim(phospho.cells.Ins)
# After filtering
dim(phospho.cells.Ins.filtered)

---

selectTimes

**Description**

selectTimes

**Usage**

`selectTimes(mat, timepoint, order, percent, w)`

**Arguments**

- `mat`: a matrix with rows correspond to phosphosites and columns correspond to samples in replicates for different treatments.
- `timepoint`: a timepoint as factor with a length equal to the number of columns of `mat`.
- `order`: a vector specifying the order of timepoints.
- `percent`: a percent (decimal) from 0 to 1, to filter phosphosites with with missing value larger than percent per timepoint.
- `w`: a timepoint window for selection of phosphosites to remove.

**Value**

a filtered matrix
Signalomes

Examples

data("phospho_liverInsTC_RUV_sample")
timepoint = gsub("(\.*)(\d+[ms])(\.*)", "\2",
    colnames(phospho.liver.Ins.TC.ratio.RUV))
timepoint[which(timepoint == "0m")] = "0s"
timepoint = factor(timepoint)
timepointOrder = c("0s", "5s", "1m", "2m", "3m", "4m", "6m")

# For demonstration purpose, we introduce missing value at 0s

#table(timepoint)

phospho.liver.Ins.TC.sim = phospho.liver.Ins.TC.ratio.RUV
rmId = which(timepoint == "0s")

# We replace the values to NA for the first 26 (~60%) of the '0s' samples
# for the first 100 phosphosite as NA
phospho.liver.Ins.TC.sim[1:100,rmId[1:26]] = NA

phospho.liver.Ins.TC.sim = selectTimes(phospho.liver.Ins.TC.sim,
    timepoint, timepointOrder, 0.5,
    w = length(table(timepoint)))

# Before filtering
dim(phospho.liver.Ins.TC.ratio.RUV)
# After filtering
dim(phospho.liver.Ins.TC.sim)

---

Signalomes

Description

A function to generate signalomes

Usage

Signalomes(KSR, predMatrix, exprsMat, KOI, threskinaseNetwork=0.9,
    signalomeCutoff=0.5)

Arguments

KSR     kinase-substrate relationship scoring results
predMatrix     output of kinaseSubstratePred function
exprsMat     a matrix with rows corresponding to phosphosites and columns corresponding
to samples
KOI     a character vector that contains kinases of interest for which expanded sig-
nalomes will be generated
threskinaseNetwork     threshold used to select interconnected kinases for the expanded signalomes
signalomeCutoff     threshold used to filter kinase-substrate relationships
Value

A list of 3 elements. Signalomes, proteinModules and kinaseSubstrates

Examples

data('phospho_L6_ratio')
data('SPSs')
grps = gsub('_.+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), ' '),
  function(x){paste(toupper(x[2]), x[3], '', sep=';')})
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
  colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
  ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <- meanAbundance(phosphoL6, grps = gsub('_.+', '',
  colnames(phosphoL6)))
aov <- matANOVA(mat=phosphoL6, grps=gsub('_.+', '', colnames(phosphoL6)))
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
  (rowSums(phosphoL6.mean > 0.5) > 0), drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
rownames(L6.phos.std) <- sapply(strsplit(rownames(L6.phos.std), ' '),
  function(x){gsub(' ', '', paste(toupper(x[2]), x[3], '', sep=';'))})
L6.phos.seq <- sapply(strsplit(rownames(L6.phos.std), ' '),
  function(x)x[4])

L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
  L6.phos.seq, numMotif = 5, numSub = 1)
set.seed(1)
L6.predMat <- kinaseSubstratePred(L6.matrices, top=30)

kinaseOI = c('PRKAA1', 'AKT1')

Signalomes_results <- Signalomes(KSR=L6.matrices,
  predMatrix=L6.predMat,
  exprsMat=L6.phos.std,
  KOI=kinaseOI)
siteAnnotate  Phosphosite annotation

Description
This function plots the combined scores of each of all kinases for a given phosphosite.

Usage
siteAnnotate(site, phosScoringMatrices, predMatrix)

Arguments
- site: site the ID of a phosphosite
- phosScoringMatrices: output from function kinaseSubstrateScore()
- predMatrix: a prediction matrix from kinaseSubstratePred()

Value
A graphical plot

Examples

data('phospho_L6_ratio')
data('SPSs')
grps = gsub('_-+', '', colnames(phospho.L6.ratio))

# Cleaning phosphosite label
phospho.site.names = rownames(phospho.L6.ratio)
L6.sites = gsub(' ', '', sapply(strsplit(rownames(phospho.L6.ratio), '~'),
function(x){paste(toupper(x[2]), x[3], '', sep=';'})))
phospho.L6.ratio = t(sapply(split(data.frame(phospho.L6.ratio), L6.sites),
colMeans))
phospho.site.names = split(phospho.site.names, L6.sites)

# Construct a design matrix by condition
design = model.matrix(~ grps - 1)

# phosphoproteomics data normalisation using RUV
ctl = which(rownames(phospho.L6.ratio) %in% SPSs)
phospho.L6.ratio.RUV = RUVphospho(phospho.L6.ratio, M = design, k = 3,
ctl = ctl)

phosphoL6 = phospho.L6.ratio.RUV
rownames(phosphoL6) = phospho.site.names

# filter for up-regulated phosphosites
phosphoL6.mean <= meanAbundance(phosphoL6,
grps = gsub('_.+',' ', colnames(phosphoL6)))
aov <- matANOVA(mat=phosphoL6, grps=gsub('_.+',' ', colnames(phosphoL6)))
phosphoL6.reg <- phosphoL6[(aov < 0.05) &
(rowSums(phosphoL6.mean > 0.5) > 0),,drop = FALSE]
L6.phos.std <- standardise(phosphoL6.reg)
rownames(L6.phos.std) <- sapply(strsplit(rownames(L6.phos.std), '-'),
  function(x){gsub(' ','', paste(toupper(x[2]), x[3], '', sep=';'))})
L6.phos.seq <- sapply(strsplit(rownames(phosphoL6.reg), '-'),
  function(x)x[4])
L6.matrices <- kinaseSubstrateScore(PhosphoSite.mouse, L6.phos.std,
  L6.phos.seq, numMotif = 5, numSub = 1)
set.seed(1)
L6.predMat <- kinaseSubstratePred(L6.matrices, top=30)
# We will look at the phosphosite AAK1;S677 for demonstration purpose.
site = "AAK1;S677;"
siteAnnotate(site, L6.matrices, L6.predMat)

---

**SPSs**

A list of Stably Phosphorylated Sites (SPSs)

**Description**

A list of stably phosphorylated sites defined from a panel of phosphoproteomics datasets. For full list of the datasets used, please refer to our preprint for the full list.

**Usage**

data(SPSs)

**Format**

An object of class character of length 100.

---

**standardise**  

Standardisation

**Description**

Standardisation by z-score transformation.

**Usage**

standardise(mat)

**Arguments**

*mat*  
a matrix with rows correspond to phosphosites and columns correspond to samples.
### Description

Tail-based imputation approach as implemented in Perseus.

### Usage

`tImpute(mat, m, s)`
**Arguments**

- `mat`  
  a matrix with rows correspond to phosphosites and columns correspond to samples.

- `m`  
  a numeric number for controlling mean downshifting.

- `s`  
  a numeric number for controlling standard deviation of downshifted sampling values.

**Value**

An imputed matrix

**Examples**

```r
data('phospho.cells.Ins.sample')
grps = gsub('_\[0-9][1]', '', colnames(phospho.cells.Ins))
phospho.cells.Ins.filtered <- selectGrps(phospho.cells.Ins, grps, 0.5, n=1)

set.seed(123)
phospho.cells.Ins.impute <- tImpute(phospho.cells.Ins.filtered)
```
Index

* datasets
  - KinaseFamily, 4
  - motif.human.list, 16
  - motif.mouse.list, 16
  - motif.rat.list, 16
  - Pathways.KEGG, 20
  - Pathways.reactome, 21
  - phospho.cells.Ins, 23
  - phospho.l6.ratio, 23
  - phospho.liver.Ins.TC.ratio.RUV, 24
  - PhosphoSite.human, 24
  - PhosphoSite.mouse, 25
  - PhosphoSite.rat, 25
  - SPSs, 36

createFrequencyMat, 2
frequencyScoring, 3

KinaseFamily, 4
kinaseSubstrateHeatmap, 5
kinaseSubstratePred, 6
kinaseSubstrateProfile, 7
kinaseSubstrateScore, 9

matANOVA, 10
meanAbundance, 11
medianScaling, 12
minmax, 13
mIntersect, 15
motif.human.list, 16
motif.mouse.list, 16
motif.rat.list, 16
mUnion (mIntersect), 15

pathwayOverrepresent, 17
pathwayRankBasedEnrichment, 18
Pathways.KEGG, 20
Pathways.reactome, 21
phosCollapse, 21
phospho.cells.Ins, 23
phospho.l6.ratio, 23
phospho.liver.Ins.TC.ratio.RUV, 24
PhosphoSite.human, 24
PhosphoSite.mouse, 25
PhosphoSite.rat, 25
plotQC, 26
ptImpute, 27
RUVphospho, 29
RUVproteome (RUVphospho), 29
scImpute, 30
selectGrps, 31
selectOverallPercent, 31
selectTimes, 32
Signalomes, 33
siteAnnotate, 35
SPSs, 36
standardise, 36
tImpute, 37