Package ‘COMPASS’

April 22, 2016

Type Package

Title Combinatorial Polyfunctionality Analysis of Single Cells

Version 1.8.1

Date 2014-07-11

Author Lynn Lin, Kevin Ushey, Greg Finak, Ravio Kolde (pheatmap)

Description COMPASS is a statistical framework that enables unbiased analysis of antigen-specific T-cell subsets. COMPASS uses a Bayesian hierarchical framework to model all observed cell-subsets and select the most likely to be antigen-specific while regularizing the small cell counts that often arise in multi-parameter space. The model provides a posterior probability of specificity for each cell subset and each sample, which can be used to profile a subject's immune response to external stimuli such as infection or vaccination.

License Artistic-2.0

BugReports https://github.com/RGLab/COMPASS/issues

VignetteBuilder knitr

Depends R (>= 3.0.2)

LinkingTo Rcpp (>= 0.11.0)

Maintainer Greg Finak <gfinak@fhcrc.org>

Imports Rcpp, data.table, RColorBrewer, scales, grid, plyr, knitr, abind, clue, grDevices, utils

Suggests flowWorkspace (>= 3.9.66), flowCore, ncdfFlow, shiny, testthat, devtools, Kmisc

LazyLoad yes

LazyData yes

biocViews FlowCytometry

NeedsCompilation yes

R topics documented:

COMPASS-package . . . 2
categories . . . 3
CellCounts . . . 3
Combinations . . . 5
COMPASS . . . 5
COMPASSContainer . . . 8
COMPASSContainer-data . . . 9
COMPASSContainerFromGatingSet . . . 10
COMPASSDescription . . . 11
COMPASSResult-accessors . . . 12
COMPASSResult-data . . . 12
FunctionalityScore . . . 13
GetThresholdedIntensities . . . 14
markers . . . 16
melt_ . . . 16
merge.COMPASSContainer . . . 17
metadata . . . 17
pheatmap . . . 18
plot.COMPASSResult . . . 22
plot2 . . . 24
PolyfunctionalityScore . . . 25
Posterior . . . 25
print.COMPASSContainer . . . 26
print.COMPASSResult . . . 27
shinyCOMPASS . . . 27
shinyCOMPASSDeps . . . 28
SimpleCOMPASS . . . 29
subset.COMPASSContainer . . . 30
summary.COMPASSContainer . . . 31
summary.COMPASSResult . . . 31
TotalCellCounts . . . 32
transpose_list . . . 33
UniqueCombinations . . . 33

Index **35**

COMPASS-package *COMPASS (Combinatorial Polyfunctionality Analysis of Single-Cells)*

Description

This package implements a model for the analysis of polyfunctionality in single-cell cytometry experiments. The model effectively identifies combinations of markers that are differentially expressed between samples of cells subjected to different stimulations.

See Also

- `COMPASSContainer`, for information on getting your cytometry data into a suitable format for use with `COMPASS`,
- `COMPASS`, for the main model fitting routine.

---

categories — *Categories*

Description

Returns the categories matrix in a `COMPASSResult` object.

Usage

```
categories(x, counts)
```

Arguments

`x`	A `COMPASSResult` object.
`counts`	Boolean; if `TRUE` we return the counts (degree of functionality) as well.

CellCounts — *Compute Number of Cells Positive for Certain Cytokine Combinations*

Description

Compute the number of cells expressing a particular combination of markers for each sample.

Usage

```
CellCounts(data, combinations)
```

Arguments

`data`	Either a `COMPASSContainer`, or a list of matrices. Each matrix `i` is of dimension `N_i` cells (rows) by `K` common markers (columns).
`combinations`	A list of 'combinations', used to denote the subsets of interest. See the examples for usage.

See Also

`Combinations`

Examples

```
set.seed(123)
## generate 10 simulated matrices of flow data
K <- 6 ## number of markers
data <- replicate(10, simplify=FALSE, {
  m <- matrix( rnorm(1E4 * K, 2000, 1000 ), ncol=K )
  m[m < 2500] <- 0
  colnames(m) <- c("IL2", "IL4", "IL6", "Mip1B", "IFNg", "TNFa")
  return(m)
})
names(data) <- sample(letters, 10)
head( data[[1]] )

## generate counts over all available combinations of markers in data
str(CellCounts(data)) ## 64 columns, as all 2^6 combinations expressed

## generate marginal counts
combos <- list(1, 2, 3, 4, 5, 6) ## marginal cell counts
cc <- CellCounts(data, combos)

## a base R way of doing the same thing
f <- function(data) {
  do.call(rbind, lapply(data, function(x) apply(x, 2, function(x) sum(x > 0))))
}
cc2 <- f(data)

## check that they're identical
stopifnot(identical( unname(cc), unname(cc2) ))

## We can also generate cell counts by expressing various combinations
## of markers (names) in the data.

## count cells expressing IL2 or IL4
CellCounts(data, "IL2|IL4")

## count cells expressing IL2, IL4 or IL6
CellCounts(data, "IL2|IL4|IL6")

## counts for each of IL2, IL4, IL6 (marginally)
CellCounts(data, c("IL2", "IL4", "IL6"))

## counts for cells that are IL2 positive and IL4 negative
CellCounts(data, "IL2 & !IL4")

## expressing the same intent with indices
CellCounts(data, list(c(1, -2)))

## all possible combinations
str(CellCounts(data, Combinations(6)))

## can also call on COMPASSContainers
data(COMPASS)
```

```
CellCounts(CC, "M1&M2")
```

`Combinations` *Generate Combinations*

Description

Given an intenger `n`, generate all binary combinations of `n` elements, transformed to indices. This is primarily for use with the `CellCounts` function, but may be useful for users in some scenarios.

Usage

```
Combinations(n)
```

Arguments

`n`	An integer.

Examples

```
Combinations(3)
```

`COMPASS` *Fit the COMPASS Model*

Description

This function fits the `COMPASS` model.

Usage

```
COMPASS(data, treatment, control, subset = NULL,
  category_filter = function(x) colSums(x > 5) > 2,
  filter_lowest_frequency = 0, filter_specific_markers = NULL,
  model = "discrete", iterations = 40000, replications = 8,
  keep_original_data = FALSE, verbose = TRUE, dropDegreeOne = FALSE, ...)
```

Arguments

`data`	An object of class `COMPASSContainer`.
`treatment`	An R expression, evaluated within the metadata, that returns `TRUE` for those samples that should belong to the treatment group. For example, if the samples that received a positive stimulation were named `"92TH023 Env"` within a variable in `meta` called `Stim`, you could write `Stim == "92TH023 Env"`. The expression should have the name of the stimulation vector on the left hand side.

control	An R expression, evaluated within the metadata, that returns TRUE for those samples that should belong to the control group. See above for details.
subset	An expression used to subset the data. We keep only the samples for which the expression evaluates to TRUE in the metadata.
category_filter	A filter for the categories that are generated. This is a function that will be applied to the *treatment counts* matrix generated from the intensities. Only categories meeting the category_filter criteria will be kept.
filter_lowest_frequency	A number specifying how many of the least expressed markers should be removed.
filter_specific_markers	Similar to filter_lowest_frequency, but lets you explicitly exclude markers.
model	A string denoting which model to fit; currently, only the discrete model ("discrete") is available.
iterations	The number of iterations (per 'replication') to perform.
replications	The number of 'replications' to perform. In order to conserve memory, we only keep the model estimates from the last replication.
keep_original_data	Keep the original COMPASSContainer as part of the COMPASS output? If memory or disk space is an issue, you may set this to FALSE.
verbose	Boolean; if TRUE we output progress information.
dropDegreeOne	Boolean; if TRUE we drop degree one categories and merge them with the negative subset.
...	Other arguments; currently unused.

Value

A COMPASSResult is a list with the following components:

fit	A list of various fitted parameters resulting from the COMPASS model fitting procedure.
data	The data used as input to the COMPASS fitting procedure – in particular, the counts matrices generated for the selected categories, n_s and n_u, can be extracted from here.
orig	If keep_original_data was set to TRUE in the COMPASS fit, then this will be the COMPASSContainer passed in. This is primarily kept for easier running of the Shiny app.

The fit component is a list with the following components:

alpha_s	The hyperparameter shared across all subjects under the stimulated condition. It is updated through the COMPASS model fitting process.
A_alphas	The acceptance rate of alpha_s, as computed through the MCMC sampling process in COMPASS.

`alpha_u`	The hyperparameter shared across all subjects under the unstimulated condition. It is updated through the `COMPASS` model fitting process.
`A_alphau`	The acceptance rate of `alpha_u`, as computed through the MCMC sampling process in `COMPASS`.
`gamma`	An array of dimensions `I x K x T`, where `I` denotes the number of individuals, `K` denotes the number of categories / subsets, and `T` denotes the number of iterations. Each cell in a matrix for a given iteration is either zero or one, reflecting whether individual `i` is responding to the stimulation for subset `k`.
`mean_gamma`	A matrix of mean response rates. Each cell denotes the mean response of individual `i` and subset `k`.
`A_gamma`	The acceptance rate for the gamma. Each element corresponds to the number of times an individual's `gamma` vector was updated.
`categories`	The category matrix, showing which categories entered the model.
`model`	The type of model called.
`posterior`	Posterior measures from the sample fit.
`call`	The matched call used to generate the model fit.

The `data` component is a list with the following components:

`n_s`	The counts matrix for stimulated samples.
`n_u`	The counts matrix for unstimulated samples.
`counts_s`	Total cell counts for stimulated samples.
`counts_u`	Total cell counts for unstimulated samples.
`categories`	The categories matrix used to define which categories will enter the model.
`meta`	The metadata. Note that only **individual-level** metadata will be kept; sample-specific metadata is dropped.
`sample_id`	The name of the vector in the metadata used to identify the samples.
`individual_id`	The name of the vector in the metadata used to identify the individuals.

The `orig` component (included if `keep_original_data` is `TRUE`) is the `COMPASSContainer` object used in the model fit.

Category Filter

The category filter is used to exclude categories (combinations of markers expressed for a particular cell) that are expressed very rarely. It is applied to the `treatment` *counts* matrix, which is a `N` samples by `K` categories matrix. Those categories which are mostly unexpressed can be excluded here. For example, the default criteria,

```
category_filter=function(x) colSums(x > 5) > 2
```

indicates that we should only retain categories for which at least three samples had at least six cells expressing that particular combination of markers.

See Also

- `COMPASSContainer`, for constructing the data object required by `COMPASS`

Examples

```
data(COMPASS) ## loads the COMPASSContainer 'CC'
fit <- COMPASS(CC,
  category_filter=NULL,
  treatment=trt == "Treatment",
  control=trt == "Control",
  verbose=FALSE,
  iterations=100 ## set higher for a real analysis
)
```

`COMPASSContainer` *Generate the Data Object used by COMPASS*

Description

This function generates the data container suitable for use with `COMPASS`.

Usage

```
COMPASSContainer(data, counts, meta, individual_id, sample_id,
  countFilterThreshold = 0)
```

Arguments

`data`	A list of matrices. Each matrix `M_i` is made up of `N_i` cells by `K` markers; for example, it could be the intensity information from an intracellular cytokine experiment. Each element of the list should be named; this name denotes which sample the cell intensities were measured from.
`counts`	A named integer vector of the cell counts(of the parent population) for each sample in `data`.
`meta`	A `data.frame` of metadata, describing the individuals in the experiment. Each row in `meta` should correspond to a row in `data`. There should be one row for each sample; i.e., one row for each element of `data`.
`individual_id`	The name of the vector in `meta` that denotes the individuals from which samples were drawn.
`sample_id`	The name of the vector in `meta` that denotes the samples. This vector should contain all of the names in the `data` input.
`countFilterThreshold`	Numeric; if the number of cells expressing at least one marker of interest is less than this threshold, we remove that file. Default is 0, which means filter is disabled.

Details

The `names` attributes for the `data` and `counts` objects passed should match.

Value

A COMPASSContainer returns a list made up of the same components as input the model, but checks and sanitizes the supplied data to ensure that it conforms to the expectations outlied above.

Examples

```
set.seed(123)
n <- 10 ## number of samples
k <- 3 ## number of markers

## generate some sample data
sid_vec <- paste0("sid_", 1:n) ## sample ids; unique names used to denote samples
iid_vec <- rep_len( paste0("iid_", 1:(n/2) ), n ) ## individual ids

## generate n matrices of 'cell intensities'
data <- replicate(n, {
  nrow <- round(runif(1) * 1E2 + 1000)
  ncol <- k
  vals <- rexp( nrow * ncol, runif(1, 1E-5, 1E-3) )
  vals[ vals < 2000 ] <- 0
  output <- matrix(vals, nrow, ncol)
  output <- output[ apply(output, 1, sum) > 0, ]
  colnames(output) <- paste0("M", 1:k)
  return(output)
})
names(data) <- sid_vec

## make a sample metadata data.frame
meta <- data.frame(
  sid=sid_vec,
  iid=iid_vec,
  trt=rep( c("Control", "Treatment"), each=5 )
)

## generate an example total counts
## recall that cells not expressing any marker are not included
## in the 'data' matrices
counts <- sapply(data, nrow) + round( rnorm(n, 1E3, 1E2) )
counts <- setNames( as.integer(counts), names(counts) )

## insert everything into a COMPASSContainer
CC <- COMPASSContainer(data, counts, meta, "iid", "sid")
```

COMPASSContainer-data *Simulated COMPASSContainer*

Description

This dataset contains simulated data for an intracellular cytokine staining experiment. In this data set, we have paired samples from five individuals, with each pair of samples being subjected to either a 'Control' condition of a 'Treatment' condition.

Details

Please see COMPASSContainer for more information on the components of this object.

The dataset is exported as CC, which is a short alias for **C**OMPASS**C**ontainer.

COMPASSContainerFromGatingSet
Create a COMPASS Container from a GatingSet

Description

This code expects a GatingSet or GatingSetList. It expects a regular expression for the node name (i.e. '/4\+$' would match '/4+' in a node name with the plus sign at the end of the string. Alternatively, you can supply a partial path. The user must supply the individual_id and sample_id, but they have default values suitable for the data we commonly see. Sometimes the child node names don't match the marker names exactly. This function will try to make some guesses about how to match these up. The filter.fun parameter is a function that does some regular expression string substitution to try and clean up the node names by removing various symobls that are often added to gates, {+/-}. The user can provide their own function to do string cleanup. Counts are extracted as well as metadata and single cell data, and these are fed into the COMPASSContainer constructor.

Usage

```
COMPASSContainerFromGatingSet(gs = NULL, node = NULL, filter.fun = NULL,
  individual_id = "PTID", sample_id = "name", mp = NULL,
  matchmethod = c("Levenshtein", "regex"), markers = NA, swap = FALSE,
  countFilterThreshold = 5000)
```

Arguments

gs	a GatingSet or GatingSetList
node	a regular expression to match a single node in the gating tree. If more than one node is matched, an error is thrown.
filter.fun	a function that does string substitution to clean up node names, i.e. turns a 'CD4+' into a 'CD4' to try and match against the parameters slot of the flowFrames in gs
individual_id	a character identifying the subject id column in the gs metadata
sample_id	a character idetifying the sample id column in the gs metadata.
mp	a list mapping node names to markers. This function tries to guess, but may fail. The user can override the guesswork.
matchmethod	a character either 'regex' or 'Levenshtein' for matching nodes to markers.
markers	a character vector of marker names to include.
swap	a logical default FALSE. Set to TRUE if the marker and channel names are swapped.

`countFilterThreshold`
: `numeric` threshold. if the number of cells expressing at least one marker of interest is less than this threshold, we remove that file. Default is 5000.

Details

There is likely not sufficient error checking.

See Also

`COMPASSContainer`

Examples

```
## Not run:
## gs is a GatingSet from flowWorkspace
COMPASSContainerFromGatingSet(gs, "4+")

## End(Not run)
```

`COMPASSDescription` *Get and Set the Description for the Shiny Application*

Description

This is used for setting an informative description used in the Shiny application.

Usage

```
COMPASSDescription(x)

COMPASSDescription(x) <- value
```

Arguments

`x`	A `COMPASS` fit.
`value`	A set of paragraphs describing the experiment, as a character vector.

Details

Information about the `COMPASS` results will be auto-generated.

COMPASSResult-accessors *COMPASSResult Accessors*

Description

These functions can be used for accessing data within a COMPASSResult.

The gamma array associated with a COMPASS model fit.

Usage

```
Gamma(x)

MeanGamma(x)
```

Arguments

x	A COMPASSResult object.

COMPASSResult-data *Simulated COMPASS fit*

Description

This dataset represents the result of fitting the COMPASS model on the accompanying dataset CC, as exported by data(COMPASS). Please see the vignette (vignette("COMPASS")) for more details on how to interact with a COMPASS fit.

Details

The model is fit as follows, using the COMPASSContainer CC.

```
CR <- COMPASS(CC,
  treatment=trt == "Treatment",
  control=trt == "Control",
  iterations=1000
)
```

The dataset is exported as CR, which is a short alias for **C**OMPASS**R**esult.

Please see COMPASS for more information on the output from a COMPASS model fit.

`FunctionalityScore` *Compute the Functionality Score for each subject fit in a COMPASS model*

Description

Computes the functionality score for each observation from the gamma matrix of a COMPASS model fit. The scores are normalized according to the total number of possible subsets that could be observed (`2^M - 1`).

Usage

```
FunctionalityScore(x, n)

## S3 method for class 'COMPASSResult'
FunctionalityScore(x, n)

## Default S3 method:
FunctionalityScore(x, n)
```

Arguments

x	An object of class `COMPASSResult`, as returned by COMPASS. Alternatively, a matrix of functionality scores, used under the assumption that the 'null' category has been dropped.
n	The number of markers included in an experiment. It is inferred from the data when `x` is a `COMPASSResult`.

Value

A numeric vector of functionality scores.

Note

The null category is implicitly dropped when computing the functionality score for a `COMPASS` result; this is not true for the regular matrix method.

Examples

```
FunctionalityScore(CR)
```

GetThresholdedIntensities
Extract Thresholded Intensities from a GatingSet

Description

This function extracts thresholded intensities for children of a node node, as specified through the map argument.

Usage

```
GetThresholdedIntensities(gs, node, map)
```

Arguments

gs	A GatingSet or GatingSetList.
node	The name, or path, of a single node in a GatingSet / GatingSetList.
map	A list, mapping node names to markers.

Details

map should be an R list, mapping node names (as specified in the gating hierarchy of the gating set) to channel names (as specified in either the desc or name columns of the parameters of the associated flowFrames in the GatingSet).

Value

A list with two components:

data	A list of thresholded intensity measures.
counts	A named vector of total cell counts at the node node.

Examples

```
if (require("flowWorkspace")) {

  ## Generate an example GatingSet that could be used with COMPASS
  ## We then pull out the 'data' and 'counts' components that could
  ## be used within a COMPASSContainer

  n <- 10 ## number of samples
  k <- 4 ## number of markers

  sid_vec <- paste0("sid_", 1:n) ## sample ids; unique names used to denote samples
  iid_vec <- rep_len( paste0("iid_", 1:(n/10) ), n ) ## individual ids
  marker_names <- c("TNFa", "IL2", "IL4", "IL6")

  ## Generate n sets of 'flow' data -- a list of matrices, each row
```

```
## is a cell, each column is fluorescence intensities on a particular
## channel / marker
data <- replicate(n, {
  nrow <- round(runif(1) * 1E4 + 1000)
  ncol <- k
  vals <- rexp( nrow * ncol, runif(1, 1E-5, 1E-3) )
  output <- matrix(vals, nrow, ncol)
  colnames(output) <- marker_names
  return(output)
})
names(data) <- sid_vec

## Put it into a GatingSet
fs <- flowSet( lapply(data, flowFrame) )
gs <- GatingSet(fs)

## Add some dummy metadata
meta <- pData(gs)
meta$PTID <- 1:10
pData(gs) <- meta

gate <- rectangleGate( list(TNFa=c(-Inf,Inf)))
add(gs, gate, parent="root", name="dummy")

## Add dummy gate

## Make some gates, and apply them
invisible(lapply(marker_names, function(marker) {
  .gate <- setNames( list( c( rexp(1, runif(1, 1E-5, 1E-3)), Inf) ), marker )
  gate <- rectangleGate(.gate=.gate)
  add(gs, gate, parent="dummy", name=paste0(marker, "+"))
}))

recompute(gs)

## Map node names to channel names
map=list(
  "TNFa+"="TNFa",
  "IL2+"="IL2",
  "IL4+"="IL4",
  "IL6+"="IL6"
)

## Pull out the data as a COMPASS-friendly dataset
node <- "dummy"
map <- map
system.time(
  output <- GetThresholdedIntensities(gs, "dummy", map)
)

system.time(
 output <- COMPASSContainerFromGatingSet(gs, "dummy", individual_id="PTID", sample_id="name")
)
```

```
  str(output)
}
```

markers *Markers*

Description

Returns the markers associated with an experiment.

Usage

```
markers(object)
```

Arguments

object	An R object.

melt_ *Make a 'Wide' data set 'Long'*

Description

Inspired by `reshape2:::melt`, we melt `data.frames` and `matrixs`. This function is built for speed.

Usage

```
melt_(data, ...)

## S3 method for class 'data.frame'
melt_(data, id.vars, measure.vars,
  variable.name = "variable", ..., value.name = "value")

## S3 method for class 'matrix'
melt_(data, ...)
```

Arguments

data	The `data.frame` to melt.
...	Arguments passed to other methods.
id.vars	Vector of id variables. Can be integer (variable position) or string (variable name). If blank, we use all variables not in `measure.vars`.
measure.vars	Vector of measured variables. Can be integer (variable position) or string (variable name). If blank, we use all variables not in `id.vars`.
variable.name	Name of variable used to store measured variable names.
value.name	Name of variable used to store values.

Details

If items to be stacked are not of the same internal type, they will be promoted in the order `logical` > `integer` > `numeric` > `character`.

`merge.COMPASSContainer` *Merge Two COMPASSContainers*

Description

This function merges two `COMPASSContainers`.

Usage

```
## S3 method for class 'COMPASSContainer'
merge(x, y, ...)
```

Arguments

`x`	A `COMPASSContainer`.
`y`	A `COMPASSContainer`.
`...`	other arguments passed to 'COMPASSContainer' call.

Examples

```
## Chop the example COMPASSContainer into two, then merge it back together
CC1 <- subset(CC, trt == "Control")
CC2 <- subset(CC, trt == "Treatment")
merged <- merge(CC1, CC2)
res <- identical(CC, merge(CC1, CC2)) ## should return TRUE in this case
stopifnot( isTRUE(res) )
```

`metadata` *Metadata*

Description

Functions for getting and setting the metadata associated with an object.

Usage

```
metadata(x)

## S3 method for class 'COMPASSContainer'
metadata(x)

## S3 method for class 'COMPASSResult'
metadata(x)

metadata(x) <- value

## S3 replacement method for class 'COMPASSContainer'
metadata(x) <- value
```

Arguments

x	An R object.
value	An R object appropriate for storing metadata in object x; typically a data.frame.

---

pheatmap *A function to draw clustered heatmaps.*

Description

A function to draw clustered heatmaps where one has better control over some graphical parameters such as cell size, etc.

Usage

```
pheatmap(mat, color = colorRampPalette(rev(brewer.pal(n = 7, name =
  "RdYlBu")))(100), kmeans_k = NA, breaks = NA, border_color = "grey60",
  cellwidth = NA, cellheight = NA, scale = "none", cluster_rows = TRUE,
  cluster_cols = TRUE, clustering_distance_rows = "euclidean",
  clustering_distance_cols = "euclidean", clustering_method = "complete",
  treeheight_row = ifelse(cluster_rows, 50, 0),
  treeheight_col = ifelse(cluster_cols, 50, 0), legend = TRUE,
  legend_breaks = NA, legend_labels = NA, annotation = NA,
  annotation_colors = NA, annotation_legend = TRUE, drop_levels = TRUE,
  show_rownames = TRUE, show_colnames = TRUE, main = NA, fontsize = 10,
  fontsize_row = fontsize, fontsize_col = fontsize,
  display_numbers = FALSE, number_format = "%.2f", fontsize_number = 0.8
  * fontsize, filename = NA, width = NA, height = NA,
  row_annotation = NA, row_annotation_legend = TRUE,
  row_annotation_colors = NA, cytokine_annotation = NA, headerplot = NA,
  polar = FALSE, order_by_max_functionality = TRUE, ...)
```

Arguments

`mat` numeric matrix of the values to be plotted.

`color` vector of colors used in heatmap.

`kmeans_k` the number of kmeans clusters to make, if we want to agggregate the rows before drawing heatmap. If NA then the rows are not aggregated.

`breaks` a sequence of numbers that covers the range of values in mat and is one element longer than color vector. Used for mapping values to colors. Useful, if needed to map certain values to certain colors, to certain values. If value is NA then the breaks are calculated automatically.

`border_color` color of cell borders on heatmap, use NA if no border should be drawn.

`cellwidth` individual cell width in points. If left as NA, then the values depend on the size of plotting window.

`cellheight` individual cell height in points. If left as NA, then the values depend on the size of plotting window.

`scale` character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are `"row"`, `"column"` and `"none"`

`cluster_rows` boolean values determining if rows should be clustered,

`cluster_cols` boolean values determining if columns should be clustered.

`clustering_distance_rows`
distance measure used in clustering rows. Possible values are `"correlation"` for Pearson correlation and all the distances supported by `dist`, such as `"euclidean"`, etc. If the value is none of the above it is assumed that a distance matrix is provided.

`clustering_distance_cols`
distance measure used in clustering columns. Possible values the same as for clustering_distance_rows.

`clustering_method`
clustering method used. Accepts the same values as `hclust`.

`treeheight_row` the height of a tree for rows, if these are clustered. Default value 50 points.

`treeheight_col` the height of a tree for columns, if these are clustered. Default value 50 points.

`legend` logical to determine if legend should be drawn or not.

`legend_breaks` vector of breakpoints for the legend.

`legend_labels` vector of labels for the `legend_breaks`.

`annotation` data frame that specifies the annotations shown on top of the columns. Each row defines the features for a specific column. The columns in the data and rows in the annotation are matched using corresponding row and column names. Note that color schemes takes into account if variable is continuous or discrete.

`annotation_colors`
list for specifying annotation track colors manually. It is possible to define the colors for only some of the features. Check examples for details.

`annotation_legend`
boolean value showing if the legend for annotation tracks should be drawn.

`drop_levels`	logical to determine if unused levels are also shown in the legend
`show_rownames`	boolean specifying if column names are be shown.
`show_colnames`	boolean specifying if column names are be shown.
`main`	the title of the plot
`fontsize`	base fontsize for the plot
`fontsize_row`	fontsize for rownames (Default: fontsize)
`fontsize_col`	fontsize for colnames (Default: fontsize)
`display_numbers`	logical determining if the numeric values are also printed to the cells.
`number_format`	format strings (C printf style) of the numbers shown in cells. For example `"%.2f"` shows 2 decimal places and `"%.1e"` shows exponential notation (see more in `gettextf`).
`fontsize_number`	fontsize of the numbers displayed in cells
`filename`	file path where to save the picture. Filetype is decided by the extension in the path. Currently following formats are supported: png, pdf, tiff, bmp, jpeg. Even if the plot does not fit into the plotting window, the file size is calculated so that the plot would fit there, unless specified otherwise.
`width`	manual option for determining the output file width in inches.
`height`	manual option for determining the output file height in inches.
`row_annotation`	data frame that specifies the annotations shown on the rows. Each row defines the features for a specific row. The rows in the data and rows in the annotation are matched using corresponding row names.The category labels are given by the data frame column names.
`row_annotation_legend`	same interpretation as the column parameters.
`row_annotation_colors`	same interpretation as the column parameters
`cytokine_annotation`	A `data.frame` of factors, with either levels `0` = unexpressed, `1` = expressed, or optionally with a third level `-1` = 'left out'. of the categories for each column. They will be colored by their degree of functionality and ordered by degree of functionality and by amount of expression if column clustering is not done.
`headerplot`	is a list with two components, order and data. Order tells how to reorder the columns of the matrix.
`polar`	Boolean; if `TRUE` we draw a polar legend. Primarily for internal use. Data is some summary statistic over the columns which will be plotted in the header where the column cluster tree usually appears. Cytokine ordering is ignored when the headerplot argument is passed.
`order_by_max_functionality`	Boolean; re-order the cytokine labels by maximum functionality?
`...`	graphical parameters for the text used in plot. Parameters passed to `grid.text`, see `gpar`.

Details

The function also allows to aggregate the rows using kmeans clustering. This is advisable if number of rows is so big that R cannot handle their hierarchical clustering anymore, roughly more than 1000. Instead of showing all the rows separately one can cluster the rows in advance and show only the cluster centers. The number of clusters can be tuned with parameter kmeans_k.

Value

Invisibly a list of components

- `tree_row` the clustering of rows as `hclust` object
- `tree_col` the clustering of columns as `hclust` object
- `kmeans` the kmeans clustering of rows if parameter `kmeans_k` was specified

Author(s)

Original version by Raivo Kolde <rkolde@gmail.com>, with modifications by Greg Finak <gfinak@fhcrc.org> and Kevin Ushey <kushey@fhcrc.org>.

Examples

```
# Generate some data
test = matrix(rnorm(200), 20, 10)
test[1:10, seq(1, 10, 2)] = test[1:10, seq(1, 10, 2)] + 3
test[11:20, seq(2, 10, 2)] = test[11:20, seq(2, 10, 2)] + 2
test[15:20, seq(2, 10, 2)] = test[15:20, seq(2, 10, 2)] + 4
colnames(test) = paste("Test", 1:10, sep = "")
rownames(test) = paste("Gene", 1:20, sep = "")

# Draw heatmaps
pheatmap(test)
pheatmap(test, kmeans_k = 2)
pheatmap(test, scale = "row", clustering_distance_rows = "correlation")
pheatmap(test, color = colorRampPalette(c("navy", "white", "firebrick3"))(50))
pheatmap(test, cluster_row = FALSE)
pheatmap(test, legend = FALSE)
pheatmap(test, display_numbers = TRUE)
pheatmap(test, display_numbers = TRUE, number_format = "%.1e")
pheatmap(test, cluster_row = FALSE, legend_breaks = -1:4, legend_labels = c("0",
"1e-4", "1e-3", "1e-2", "1e-1", "1"))
pheatmap(test, cellwidth = 15, cellheight = 12, main = "Example heatmap")
#pheatmap(test, cellwidth = 15, cellheight = 12, fontsize = 8, filename = "test.pdf")


# Generate column annotations
annotation = data.frame(Var1 = factor(1:10 %% 2 == 0,
                              labels = c("Class1", "Class2")), Var2 = 1:10)
annotation$Var1 = factor(annotation$Var1, levels = c("Class1", "Class2", "Class3"))
rownames(annotation) = paste("Test", 1:10, sep = "")

pheatmap(test, annotation = annotation)
```

```
pheatmap(test, annotation = annotation, annotation_legend = FALSE)
pheatmap(test, annotation = annotation, annotation_legend = FALSE, drop_levels = FALSE)

# Specify colors
Var1 = c("navy", "darkgreen")
names(Var1) = c("Class1", "Class2")
Var2 = c("lightgreen", "navy")

ann_colors = list(Var1 = Var1, Var2 = Var2)

#Specify row annotations
row_ann <- data.frame(foo=gl(2,nrow(test)/2),`Bar`=relevel(gl(2,nrow(test)/2),"2"))
rownames(row_ann)<-rownames(test)
pheatmap(test, annotation = annotation, annotation_legend = FALSE, drop_levels = FALSE,row_annotation = row_ann)

#Using cytokine annotations
M<-matrix(rnorm(8*20),ncol=8)
row_annotation<-data.frame(A=gl(4,nrow(M)/4),B=gl(4,nrow(M)/4))
eg<-expand.grid(factor(c(0,1)),factor(c(0,1)),factor(c(0,1)))
colnames(eg)<-c("IFNg","TNFa","IL2")
rownames(eg)<-apply(eg,1,function(x)paste0(x,collapse=""))
rownames(M)<-1:nrow(M)
colnames(M)<-rownames(eg)
cytokine_annotation=eg
pheatmap(M,annotation=annotation,row_annotation=row_annotation,annotation_legend=TRUE,row_annotation_legend=TR

# Specifying clustering from distance matrix
drows = dist(test, method = "minkowski")
dcols = dist(t(test), method = "minkowski")
pheatmap(test, clustering_distance_rows = drows, clustering_distance_cols = dcols)
```

`plot.COMPASSResult` *Plot a COMPASSResult*

Description

This function can be used to visualize the mean probability of response; that is, the probability that there is a difference in response between samples subjected to the 'treatment' condition, and samples subjected to the 'control' condition.

Usage

```
## S3 method for class 'COMPASSResult'
plot(x, y, subset = NULL, threshold = 0.01,
  minimum_dof = 1, maximum_dof = Inf, must_express = NULL, row_annotation,
  palette = colorRampPalette(brewer.pal(10, "Purples"))(20),
  show_rownames = FALSE, show_colnames = FALSE, measure = NULL,
  order_by = FunctionalityScore, ...)
```

Arguments

x	An object of class COMPASSResult.
y	This argument gets passed to row_annotation, if row_annotation is missing. It can be used to group rows (individuals) by different conditions as defined in the metadata.
subset	An R expression, evaluated within the metadata, used to determine which individuals should be kept.
threshold	A numeric threshold for filtering under-expressed categories. Any categories with mean score < threshold are removed.
minimum_dof	The minimum degree of functionality for the categories to be plotted.
maximum_dof	The maximum degree of functionality for the categories to be plotted.
must_express	A character vector of markers that should be included in each subset plotted. For example, must_express=c("TNFa & IFNg") says we include only subsets that are positive for both TNFa or IFNg, while must_express=c("TNFa", "IFNg") says we should keep subsets which are positive for either TNFa or IFNg.
row_annotation	A vector of names, pulled from the metadata, to be used for row annotation.
palette	The colour palette to be used.
show_rownames	Boolean; if TRUE we display row names (ie, the individual ids).
show_colnames	Boolean; if TRUE we display column names (ie, the column name associated with a cytokine; typically not needed)
measure	Optional. By default, we produce a heatmap of the mean gammas produced in a model fit. We can override this by supplying a matrix of suitable dimension as well; these can be generated with the Posterior* functions – see Posterior for examples.
order_by	Order rows within a group. This should be a function; e.g. FunctionalityScore, mean, median, and so on. Set this to NULL to preserve the original ordering of the data.
...	Optional arguments passed to pheatmap.

Value

The plot as a grid object (grob). It can be redrawn with e.g. grid::grid.draw().

Examples

```
## visualize the mean probability of reponse
plot(CR)

## visualize the proportion of cells belonging to a category
plot(CR, measure=PosteriorPs(CR))
```

plot2 *Plot a pair of COMPASSResults*

Description

This function can be used to visualize the mean probability of response – that is, the probability that there is a difference in response between samples subjected to the 'treatment' condition, and samples subjected to the 'control' condition.

Usage

```
plot2(x, y, subset, threshold = 0.01, minimum_dof = 1, maximum_dof = Inf,
  must_express = NULL, row_annotation = NULL, palette = NA,
  show_rownames = FALSE, show_colnames = FALSE, ...)
```

Arguments

`x`	An object of class `COMPASSResult`.
`y`	An object of class `COMPASSResult`.
`subset`	An R expression, evaluated within the metadata, used to determine which individuals should be kept.
`threshold`	A numeric threshold for filtering under-expressed categories. Any categories with mean score < `threshold` are removed.
`minimum_dof`	The minimum degree of functionality for the categories to be plotted.
`maximum_dof`	The maximum degree of functionality for the categories to be plotted.
`must_express`	A character vector of markers that should be included in each subset plotted. For example, `must_express=c("TNFa & IFNg")` says we include only subsets that are positive for both `TNFa` or `IFNg`, while `must_express=c("TNFa", "IFNg")` says we should keep subsets which are positive for either `TNFa` or `IFNg`.
`row_annotation`	A vector of names, pulled from the metadata, to be used for row annotation.
`palette`	The colour palette to be used.
`show_rownames`	Boolean; if `TRUE` we display row names (ie, the individual ids).
`show_colnames`	Boolean; if `TRUE` we display column names (ie, the column name associated with a cytokine; typically not needed)
`...`	Optional arguments passed to `pheatmap`.

Value

The plot as a `grid` object (`grob`). It can be redrawn with e.g. `grid::grid.draw()`.

`PolyfunctionalityScore` *Compute the Polyfunctionality Score for each subject fit in a COMPASS model*

Description

Computes the Polyfunctionality score for each observation from the gamma matrix of a `COMPASS` model fit. The scores are normalized to one.

Usage

```
PolyfunctionalityScore(x, degree, n)

## S3 method for class 'COMPASSResult'
PolyfunctionalityScore(x, degree, n)

## Default S3 method:
PolyfunctionalityScore(x, degree, n)
```

Arguments

`x`	An object of class `COMPASSResult`, as returned by `COMPASS`. Alternatively, a matrix of functionality scores.
`degree`	A vector of weights. If missing, this is the 'degree of functionality', that is, the number of markers expressed in a particular category.
`n`	The total number of markers. This is inferred when `x` is a `COMPASSResult`, and is unused in that case.

Value

A numeric vector of polyfunctionality scores.

Examples

```
PolyfunctionalityScore(CR)
```

`Posterior` *Retrieve Posterior Measures from a COMPASS fit*

Description

These functions can be used to retrieve different posterior measures from a `COMPASS` fit object.

Usage

```
Posterior(x)

PosteriorDiff(x)

PosteriorLogDiff(x)

PosteriorPs(x)

PosteriorPu(x)
```

Arguments

x	An object of class COMPASSResult.

Details

The posterior items retrieved are described as follows::

PosteriorPs: The posterior probability that the samples subjected to the 'treatment', or 'stimulated', condition responded.

PosteriorPu: The posterior probability that the samples subjected to the 'control', or 'unstimulated', condition responded.

PosteriorDiff: The difference in posterior response rates, as described above.

PosteriorLogDiff: The difference in the log response rates, as described above.

Examples

```
Posterior(CR)
PosteriorPs(CR)
PosteriorPu(CR)
PosteriorDiff(CR)
PosteriorLogDiff(CR)
```

---

print.COMPASSContainer

Print a COMPASSContainer Object

Description

This function prints a COMPASSContainer object, giving basic information about the object and the data it encapsulates.

Usage

```
## S3 method for class 'COMPASSContainer'
print(x, ...)
```

Arguments

`x`	An object of class `COMPASSContainer`.
`...`	Optional arguments passed to `cat`.

Examples

```
print(CC)
```

`print.COMPASSResult` *Print a COMPASSResult Object*

Description

This function prints basic information about the model fit by a `COMPASS` call.

Usage

```
## S3 method for class 'COMPASSResult'
print(x, ...)
```

Arguments

`x`	An object of class `COMPASSResult`.
`...`	Optional arguments; currently unused.

Examples

```
print(CR)
```

`shinyCOMPASS` *Start a Shiny Application for Visualizing COMPASS Results*

Description

This function takes a `COMPASSResult` object, and generates a local Shiny application for visualizing the results.

Usage

```
shinyCOMPASS(x, dir = NULL, meta.vars, facet1 = "None", facet2 = "None",
  facet3 = "None",
  main = "Heatmap of Ag-Specificity Posterior Probabilities",
  stimulation = NULL, launch = TRUE, ...)
```

Arguments

x	An object of class COMPASSResult.
dir	A location to write out the .rds files that will be loaded and used by the Shiny application.
meta.vars	A character vector of column names that should be used for potential facetting in the Shiny app. By default, we take all metadata variables; you may want to limit this if you know certain variables are not of interest.
facet1,facet2,facet3	Default values for facets in the Shiny app. Each should be the name of a single vector in the metadata.
main	A title to give to the heatmap and subset histogram plots.
stimulation	The name of the stimulation applied. If this is NULL, the stimulations used are inferred from the data (ie, the COMPASS call used).
launch	Boolean; if TRUE we launch the Shiny application. Otherwise, the user can launch it manually by navigating to the directory dir and running shiny::runApp().
...	Optional arguments passed to shiny::runApp.

See Also

shinyCOMPASSDeps, for identifying packages that you need in order to run the Shiny application.

Examples

```
if (interactive()) {
  oldOpt <- getOption("example.ask")
  options(example.ask=FALSE)
  on.exit( options(example.ask=oldOpt) )
  shinyCOMPASS(CR)
  options(example.ask=TRUE)
}
```

---

shinyCOMPASSDeps *List Shiny Dependencies*

Description

This function can be used to identify the packages still needed in order to launch the Shiny app.

Usage

```
shinyCOMPASSDeps(verbose = TRUE)
```

Arguments

verbose	Boolean; if TRUE we print installation instructions to the screen.

Examples

```
shinyCOMPASSDeps()
```

SimpleCOMPASS *Fit the discrete COMPASS Model*

Description

This function fits the COMPASS model from a user-provided set of stimulated / unstimulated matrices. See the NOTE for important details.

Usage

```
SimpleCOMPASS(n_s, n_u, meta, individual_id, sample_id, iterations = 10000,
  replications = 8, verbose = TRUE)
```

Arguments

n_s	The cell counts for stimulated cells.
n_u	The cell counts for unstimulated cells.
meta	A data.frame of metadata, describing the individuals in the experiment. Each row in meta should correspond to a row in data. There should be one row for each sample; i.e., one row for each element of n_s and n_u.
individual_id	The name of the vector in meta that denotes the individuals from which samples were drawn.
sample_id	The name of the vector in meta that denotes the samples. This vector should contain all of the names in the data input.
iterations	The number of iterations (per 'replication') to perform.
replications	The number of 'replications' to perform. In order to conserve memory, we only keep the model estimates from the last replication.
verbose	Boolean; if TRUE we output progress information.

Value

A list with class COMPASSResult with two components, the fit containing parameter estimates and parameter acceptance rates, and data containing the generated data used as input for the model.

Note

n_s and n_u counts matrices should contain ALL 2^M possible combinations of markers, even if they are 0 for some combinations. The code expects the marker combinations to be named in the following way: "M1&M2&!M3" means the combination represents cells expressing marker "M1" and "M2" and not "M3". For 3 markers, there should be 8 such combinations, such that n_s and n_u have 8 columns.

Examples

```
## Not run:
set.seed(123)
n <- 10 ## number of samples
k <- 3 ## number of markers

## generate some sample data
sid_vec <- paste0("sid_", 1:n) ## sample ids; unique names used to denote samples
iid_vec <- rep_len( paste0("iid_", 1:(n/2) ), n ) ## individual ids
data <- replicate(n, {
  nrow <- round(runif(1) * 1E4 + 1000)
  ncol <- k
  vals <- rexp( nrow * ncol, runif(1, 1E-5, 1E-3) )
  vals[ vals < 2000 ] <- 0
  output <- matrix(vals, nrow, ncol)
  output <- output[ apply(output, 1, sum) > 0, ]
  colnames(output) <- paste0("M", 1:k)
  return(output)
})
meta <- data.frame(
  sid=sid_vec,
  iid=iid_vec,
  trt=rep( c("Control", "Treatment"), each=(n/2) )
)

## generate counts for n_s, n_u
n_s <- CellCounts( data[1:(n/2)], Combinations(k) )
n_u <- CellCounts( data[(n/2+1):n], Combinations(k) )

## A smaller number of iterations is used here for running speed;
## prefer using more iterations for a real fit
SimpleCOMPASS(n_s, n_u, meta, "iid", "sid", iterations=100)

## End(Not run)
```

`subset.COMPASSContainer` *Subset a COMPASSContainer*

Description

Use this function to subset a `COMPASSContainer`.

Usage

```
## S3 method for class 'COMPASSContainer'
subset(x, subset, ...)
```

Arguments

x	A COMPASSContainer.
subset	A logical expression, evaluated within the metadata, indicating which entries to keep.
...	other arguments passed to 'COMPASSContainer' call.

Examples

```
subset(CC, iid == "iid_1")
```

---

summary.COMPASSContainer
Summarize a COMPASSContainer Object

Description

This function prints summary information about a COMPASSContainer object – the number of samples, basic information about the metadata, and so on.

Usage

```
## S3 method for class 'COMPASSContainer'
summary(object, ...)
```

Arguments

object	An object of class COMPASSContainer.
...	Optional arguments; currently ignored.

Examples

```
summary(CC)
```

---

summary.COMPASSResult *Summarize a COMPASSResult Object*

Description

This function prints basic information about the model fit by a COMPASS call.

Usage

```
## S3 method for class 'COMPASSResult'
summary(object, ...)
```

Arguments

`object`	An object of class `COMPASSResult`.
`...`	Optional arguments; currently unused.

Examples

```
print(CR)
```

`TotalCellCounts` *Compute Total Cell Counts*

Description

This function is used to compute total cell counts, per individual, from a `COMPASSContainer`.

Usage

```
TotalCellCounts(data, subset, aggregate = TRUE)
```

Arguments

`data`	A `COMPASSContainer`.
`subset`	An expression, evaluated within the metadata, defining the subset of `data` over which the counts are computed. If left unspecified, the counts are computed over all samples.
`aggregate`	Boolean; if `TRUE` we sum over the individual, to get total counts across samples for each individual.

Examples

```
TotalCellCounts(CC, trt == "Treatment")
TotalCellCounts(CC, trt == "Control")
TotalCellCounts(CC)
```

transpose_list *Transpose a List*

Description

Transpose a matrix-like list.

Usage

```
transpose_list(x)
```

Arguments

x	An R list.

Examples

```
l <- list( 1:3, 4:6, 7:9 )
stopifnot( identical(
  transpose_list( transpose_list(l) ), l
) )
```

UniqueCombinations *Generate Unique Combinations*

Description

Generate all possible unique combinations of `x`. Primarily used as a helper function for `CellCounts`, but may be occasionally useful to the end user.

Usage

```
UniqueCombinations(x, as.matrix)

## S3 method for class 'COMPASSContainer'
UniqueCombinations(x, as.matrix = FALSE)

## Default S3 method:
UniqueCombinations(x, as.matrix = FALSE)
```

Arguments

x	Either a `COMPASSContainer`, or a list of matrices.
as.matrix	Boolean; if `TRUE` we return results as a matrix; otherwise, we return the results as a list.

Examples

```
UniqueCombinations(CC)
```

Index

categories, 3
CC, *12*
CC (COMPASSContainer-data), 9
CellCounts, 3, *5*
Combinations, *3*, 5
COMPASS, *3*, 5, *12*, *13*, *25*, *27*, *31*
COMPASS-package, 2
COMPASSContainer, *3*, *7*, 8, *10*, *11*
COMPASSContainer-data, 9
COMPASSContainerFromGatingSet, 10
COMPASSDescription, 11
COMPASSDescription<- (COMPASSDescription), 11
COMPASSResult-accessors, 12
COMPASSResult-data, 12
CR (COMPASSResult-data), 12

dist, *19*

FunctionalityScore, 13

Gamma (COMPASSResult-accessors), 12
gettextf, *20*
GetThresholdedIntensities, 14
gpar, *20*
grid.text, *20*

hclust, *19*, *21*

markers, 16
MeanGamma (COMPASSResult-accessors), 12
melt_, 16
merge.COMPASSContainer, 17
metadata, 17
metadata<- (metadata), 17

pheatmap, 18
plot (plot.COMPASSResult), 22
plot.COMPASSResult, 22
plot2, 24
PolyfunctionalityScore, 25
Posterior, *23*, 25
PosteriorDiff (Posterior), 25
PosteriorLogDiff (Posterior), 25
PosteriorPs (Posterior), 25
PosteriorPu (Posterior), 25
print.COMPASSContainer, 26
print.COMPASSResult, 27

shinyCOMPASS, 27
shinyCOMPASSDeps, *28*, 28
SimpleCOMPASS, 29
subset.COMPASSContainer, 30
summary.COMPASSContainer, 31
summary.COMPASSResult, 31

TotalCellCounts, 32
transpose_list, 33

UniqueCombinations, 33