Package `conos`

November 8, 2021

Title Clustering on Network of Samples

Version 1.4.4

Description Wires together large collections of single-cell RNA-seq datasets, which allows for both the identification of recurrent cell clusters and the propagation of information between datasets in multi-sample or atlas-scale collections. 'Conos' focuses on the uniform mapping of homologous cell types across heterogeneous sample collections. For instance, users could investigate a collection of dozens of peripheral blood samples from cancer patients combined with dozens of controls, which perhaps includes samples of a related tissue such as lymph nodes. This package interacts with data available through the 'conosPanel' package, which is available in a 'drat' repository. To access this data package, see the instructions at <https://github.com/kharchenkolab/conos>. The size of the 'conosPanel' package is approximately 12 MB.

License GPL-3

Copyright See the file COPYRIGHTS for various conos copyright details

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0), Matrix, igraph

biocViews

Imports abind, cowplot, ComplexHeatmap, dendextend, dplyr, ggplot2, ggrepel, gridExtra,irlba,leidenAlg, magrittr,Matrix.utils, methods,N2R,parallel,R6,reshape2,rlang,Rtsne,scorer (>= 1.0.0),stats,tools,utils

RoxygenNote 7.1.2

Suggests AnnotationDbi, BiocParallel, conosPanel, drat, DESeq2, entropy,ggrastr,GO.db, jsonlite, knitr, org.Hs.eg.db, org.Mm.eg.db, p2data,pagoda2,pbapply,PMA,pROC,plyr,rhdf5, rmarkdown,rnummys,Seurat,shinyessloaders,SummarizedExperiment, testthat,tibble,uwot,zoo

Additional_repositories https://kharchenkolab.github.io/drat/

URL https://github.com/kharchenkolab/conos

BugReports https://github.com/kharchenkolab/conos/issues
R topics documented:

- basicSeuratProc
- bestClusterThresholds
- bestClusterTreeThresholds
- buildWijMatrix
- Conos
- convertToPagoda2
- edgeMat<-
- estimateWeightEntropyPerCell
- findSubcommunities
- getBetweenCellTypeCorrectedDE
- getBetweenCellTypeDE
- getCellNames
- getClustering
- getCountMatrix
- getEmbedding
- getGeneExpression
- getGenes
- getOverdispersedGenes
- getPca
- getPerCellTypeDE
- getRawCountMatrix
- getSampleNamePerCell
- greedyModularityCut
- p2app4conos
- plotClusterBarplots
- plotClusterBoxPlotsByAppType
- plotComponentVariance
- plotDEheatmap
- projectKNNs
- rawMatricesWithCommonGenes
- saveConosForScanPy
- saveDEasCSV
- saveDEasJSON
### basicSeuratProc

Create and preprocess a Seurat object

#### Description

Create and preprocess a Seurat object

#### Usage

```r
basicSeuratProc(
  count.matrix,
  vars.to.regress = NULL,
  verbose = TRUE,
  do.par = TRUE,
  n.pcs = 100,
  cluster = TRUE,
  tsne = TRUE,
  umap = FALSE
)
```

#### Arguments

- `count.matrix` gene count matrix
- `vars.to.regress` variables to regress with Seurat (default=NULL)
- `verbose` boolean Verbose mode (default=TRUE)
- `do.par` boolean Use parallel processing for regressing out variables faster (default=TRUE)
- `n.pcs` numeric Number of principal components (default=100)
- `cluster` boolean Whether to perform clustering (default=TRUE)
- `tsne` boolean Whether to construct tSNE embedding (default=TRUE)
- `umap` boolean Whether to construct UMAP embedding, works only for Seurat v2.3.1 or higher (default=FALSE)

#### Value

Seurat object
**bestClusterThresholds**  
*Find threshold of cluster detectability*

**Description**
For a given clustering, walks the walktrap result tree to find a subtree with max(min(sens,spec)) for each cluster, where sens is sensitivity, spec is specificity.

**Usage**

```r
bestClusterThresholds(res, clusters, clmerges = NULL)
```

**Arguments**
- `res` walktrap result object (igraph)
- `clusters` cluster factor
- `clmerges` integer matrix of cluster merges (default=NULL). If NULL, the function tree-Jaccard() performs calculation without it.

**Value**
- a list of $thresholds - per cluster optimal detectability values, and $node - internal node id (merge row) where the optimum was found.

**bestClusterTreeThresholds**  
*Find threshold of cluster detectability in trees of clusters*

**Description**
For a given clustering, walks the walktrap (of clusters) result tree to find a subtree with max(min(sens,spec)) for each cluster, where sens is sensitivity, spec is specificity.

**Usage**

```r
bestClusterTreeThresholds(res, leaf.factor, clusters, clmerges = NULL)
```

**Arguments**
- `res` walktrap result object (igraph) where the nodes were clusters
- `leaf.factor` a named factor describing cell assignments to the leaf nodes (in the same order as res$names)
- `clusters` cluster factor
- `clmerges` integer matrix of cluster merges (default=NULL). If NULL, the function tree-Jaccard() performs calculation without it.
buildWijMatrix

**Value**

A list of $thresholds - per cluster optimal detectability values, and $node - internal node id (merge row) where the optimum was found.

---

**Description**

Rescale the weights in an edge matrix to match a given perplexity.

**Usage**

```r
buildWijMatrix(x, threads = NULL, perplexity = 50)
```

## S3 method for class `TsparseMatrix`
buildWijMatrix(x, threads = NULL, perplexity = 50)

## S3 method for class `CsparseMatrix`
buildWijMatrix(x, threads = NULL, perplexity = 50)

**Arguments**

- **x** A sparse matrix
- **threads** numeric The maximum number of threads to spawn. Determined automatically if NULL (default=NULL)
- **perplexity** numeric Given perplexity (default=50)

**Value**

A list with the following components:

- **'dist'** An [N,K] matrix of the distances to the nearest neighbors.
- **'id'** An [N,K] matrix of the node indexes of the nearest neighbors. Note that this matrix is 1-indexed, unlike most other matrices in this package.
- **'k'** The number of nearest neighbors.
Description
The class encompasses sample collections, providing methods for calculating and visualizing joint graph and communities.

Public fields
- `samples` list of samples (Pagoda2 or Seurat objects)
- `pairs` pairwise alignment results
- `graph` alignment graph
- `clusters` list of clustering results named by clustering type
- `expression.adj` adjusted expression values
- `embeddings` list of joint embeddings
- `embedding` joint embedding
- `n.cores` number of cores
- `misc` list with unstructured additional info

Methods

**Public methods:**
- `Conos$new()`: initialize Conos class
- `Conos$addSamples()`
- `Conos$buildGraph()`
- `Conos$getDifferentialGenes()`
- `Conos$findCommunities()`
- `Conos$plotPanel()`
- `Conos$embedGraph()`
- `Conos$plotClusterStability()`
- `Conos$plotGraph()`
- `Conos$correctGenes()`
- `Conos$propagateLabels()`
- `Conos$getClusterCountMatrices()`
- `Conos$getDatasetPerCell()`
- `Conos$getJointCountMatrix()`
- `Conos$clone()`
Conos$new(
  x,
  ...,
  n.cores = parallel::detectCores(logical = FALSE),
  verbose = TRUE,
  override.conos.plot.theme = FALSE
)

Arguments:
  x  a named list of pagoda2 or Seurat objects (one per sample)
  ... additional parameters upon initializing Conos
  n.cores  numeric Number of cores to use (default=parallel::detectCores(logical=FALSE))
  verbose  boolean Whether to provide verbose output (default=TRUE)
  override.conos.plot.theme  boolean Whether to reset plot settings to the ggplot2 default
                             (default=FALSE)

Returns: a new 'Conos' object

Examples:
  con <- Conos$new(small_panel.preprocessed, n.cores=1)

Method addSamples(): Initialize or add a set of samples to the conos panel. Note: this will
simply add samples, but will not update graph, clustering, etc.

Usage:
  Conos$addSamples(x, replace = FALSE, verbose = FALSE)

Arguments:
  x  a named list of pagoda2 or Seurat objects (one per sample)
  replace  boolean Whether the existing samples should be purged before adding new ones (de-
            fault=FALSE)
  verbose  boolean Whether to provide verbose output (default=FALSE)

Returns: invisible view of the full sample list

Method buildGraph(): Build the joint graph that encompasses all the samples, establishing
weighted inter-sample cell-to-cell links

Usage:
  Conos$buildGraph(
    k = 15,
    k.self = 10,
    k.self.weight = 0.1,
    alignment.strength = NULL,
    space = "PCA",
    matching.method = "mNN",
    metric = "angular",
    k1 = k,
    data.type = "counts",
    l2.sigma = 1e+05,
  )
var.scale = TRUE,
ncomps = 40,
n.odgenes = 2000,
matching.mask = NULL,
exclude.samples = NULL,
common.centering = TRUE,
verbose = TRUE,
base.groups = NULL,
append.global.axes = TRUE,
append.decoys = TRUE,
decoy.threshold = 1,
n.decoys = k * 2,
score.component.variance = FALSE,
snn = FALSE,
snn.quantile = 0.9,
min.snn.jaccard = 0,
min.snn.weight = 0,
snn.k.self = k.self,
balance.edge.weights = FALSE,
balancing.factor.per.cell = NULL,
same.factor.downweight = 1,
k.same.factor = k,
balancing.factor.per.sample = NULL
)

Arguments:

k  integer integer Size of the inter-sample neighborhood (default=15)
k.self  integer Size of the with-sample neighborhoods (default=10).
k.self.weight numeric Weight multiplier on the intra-sample edges relative to inter-sample edges (default=0.1)
alignment.strength numeric Alignment strength (default=NULL will result in alignment.strength=0)
space character Reduced expression space used to establish putative alignments between pairs of samples (default='PCA'). Currently supported spaces are: — "CPCA" Common principal component analysis — "JNMF" Joint NMF — "genes" Gene expression space (log2 transformed) — "PCA" Principal component analysis — "CCA" Canonical correlation analysis — "PMA" (Penalized Multivariate Analysis <https://cran.r-project.org/web/packages/PMA/index.html>)
matching.method character Matching method (default='mNN'). Currently supported methods are "NN" (nearest neighbors) or "mNN" (mutual nearest neighbors).
metric character Distance metric to measure similarity (default='angular'). Currently supported metrics are "angular" and "L2".
k1 numeric Neighborhood radius for identifying mutually-matching neighbors (default=k). Note that k1 must be greater than or equal to k, i.e. k1>=k. Increasing k1 beyond k will lead to more aggressive alignment of distinct subpopulations (i.e. increased alignment strengths).
data.type character Type of data type in the input pagoda2 objects within r.n (default='counts').
l2.sigma numeric L2 distances get transformed as exp(-d/\sigma) using this value (default=1e5)
var.scale boolean Whether to use common variance scaling (default=TRUE). If TRUE, use geometric means for variance, as we're trying to focus on the common variance components. See scaledMatricesP2() code.
ncomps integer Number of components (default=40)
n.odgenes integer Number of overdispersed genes to be used in each pairwise alignment (default=2000)
matching.mask an optional matrix explicitly specifying which pairs of samples should be compared (a symmetrical matrix of logical values with row and column names corresponding to sample names). (default=NULL). By default, comparisons between all pairs are allowed. The argument can be used to exclude comparisons across certain pairs of samples (e.g. technical replicates, which are expected to show very high similarity).
exclude.samples optional list of sample names that should be excluded from the alignment and the resulting graph (default=NULL)
common.centering boolean When calculating reduced expression space for a given sample pair, whether the expression of genes should be centered using the mean from both samples (TRUE) or using the mean within each sample (FALSE) (default=TRUE)
verbose boolean Whether to provide verbose output (default=TRUE)
base.groups an optional factor on cells specifying previously-obtained cell grouping to be used for adjusting the sample alignment (default: NULL). Specifically, cell clusters specified by the base.groups can be used to i) calculate global expression axes which are appended to the overall set of eigenvectors, ii) adding decoy cells.
append.global.axes boolean Whether to project samples on global expression axes, as defined by pre-defined (typically crude) set of cell subpopulations as specified by the base.groups parameter (default=TRUE, but works only if base.groups is specified)
append.decoys boolean Whether to use pre-defined cell groups (specified by base.groups) to append decoy cells to the samples which are otherwise lacking any of the pre-specified cell groups (default=TRUE, but works only if base.groups is specified). The decoy cells can reduce the number of erroneous matches in highly heterogeneous sample collections, where some of the samples lack entire cell subpopulations which are found in other samples. The approach only works if the base.groups (typically a crude clustering of top-level cell types) can be established with a reasonable confidence.
decoy.threshold integer Minimal number of cells of a given cell type that should exist in a given sample (according to base groups) to avoid addition of decoy cells to that sample for the purposes of alignment (default=1)
n.decoys integer Number of decoy cells that should be added to a sample that had less than decoy.threshold cells of a given cell type (default=k*2)
score.component.variance boolean Whether to score the amount of total variance explained by different components (default=FALSE as it takes extra time to calculate)
append.global.axes boolean Whether to transform the joint graph by computing a shared nearest neighborhood graph (analogous to Seurat 3), further weighting the edges between two matched cells based on the similarity (measured by Jaccard coefficient) of all of their predicted neighbors (across all of the samples) (default: FALSE)
snn.quantile numeric Specifies how the shared neighborhood graph transformation will determine final edge weights. If snn.quantile=NULL, the edge weight will be simply equal to the Jaccard coefficient of the neighborhoods. If snn.quantile is a vector of two numeric values (p1, p2), they will be treated as quantile probabilities, and quantile values (q1,q2) on the set of all Jaccard coefficients (for all edges) will be determined. The edge weights will then be reset, so that edges with Jaccard coefficients below or equal to q1 will be set to 0, and those with coefficients >=q2 will be set to 1. The rest of the weights will be mapped uniformly from [q1,q2]->[0,1] range. If a single numeric value is supplied, it will
be treated as a symmetric quantile probability (i.e. `snn.quantile=0.8` is equivalent to specifying `snn.quantile=c(1-0.8,0.8)`). (default: 0.9)

`min.snn.jaccard` numeric Minimum Jaccard coefficient required for a shared neighborhood graph edge (default: 0). The edges with Jaccard coefficients below this threshold will be removed (i.e. weight set to 0)

`min.snn.weight` numeric Shared nearest neighbor procedure will adjust the weights of the edges, and even eliminate some of the edges (by setting their weight to zero). The `min.snn.weight` parameter allows to set a minimal adjusted edge weight, so that the edge weight is never reduced beyond this level (and hence never deleted) (default: 0 - no adjustments)

`snn.k.self` integer Size of the within-sample neighborhood to be used in shared nearest neighbor calculations (default=k.self)

`balance.edge.weights` boolean Whether to balance edge weights to control for a cell- or sample- specific factor (default=FALSE)

`balancing.factor.per.cell` A per-cell factor (discrete factor, named with cell names) specifying a design difference should be controlled for by adjusting edge weights in the joint graph (default=NULL)

`same.factor.downweight` numeric Optional weighting factor for edges connecting cells with the same cell factor level per cell balancing (default=1.0)

`k.same.factor` integer An neighborhood size that should be used when aligning samples of the same balancing.factor.per.sample level. Setting a value smaller than k will lead to reduction of alignment strength within the sample batches (default=k)

`balancing.factor.per.sample` A covariate factor per sample that should be controlled for by adjusting edge weights in the joint graph (default=NULL)

**Returns:** joint graph to be used for downstream analysis

**Examples:**
```r
con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$buildGraph(k=10, k.self=5, space="PCA", ncomps=10, n.odgenes=20, matching.method="mNN", metric="angular", score.component.variance=TRUE, verbose=TRUE)
```

**Method** `getDifferentialGenes()`: Calculate genes differentially expressed between cell clusters. Estimates base mean, z-score, p-values, specificity, precision, expressionFraction, AUC (if `append.auc=TRUE`)

**Usage:**
```r
Conos$getDifferentialGenes(
  clustering = NULL,
  groups = NULL,
  z.threshold = 3,
  upregulated.only = FALSE,
  verbose = TRUE,
  append.specificity.metrics = TRUE,
  append.auc = TRUE
)
```

**Arguments:**
clustering character Name of the clustering to use (see names(con$clusters)) for the value of the groups factor (default: NULL - if groups are not specified, the first clustering will be used)
groups a cell factor (a factor named with cell names) specifying clusters of cells to be compared (one against all). To compare two cell clusters against each other, simply pass a factor containing only two levels (default: NULL, see clustering)
z.threshold numeric Minimum absolute value of a Z score for which the genes should be reported (default=3.0).
upregulated.only boolean If TRUE, will report only genes significantly upregulated in each cluster; otherwise both up- and down-regulated genes will be reported (default=FALSE)
verbose boolean Whether to provide verbose output (default=TRUE)
append.specificity.metrics boolean Whether to append specificity metrics (default=TRUE)
append.auc boolean Whether to append AUC scores (default=TRUE)

Returns: list of DE results; each is a data frame with rows corresponding to the differentially expressed genes, and columns listing log2 fold change (M), signed Z scores (both raw and adjusted for multiple hypothesis using BH correction), optional specificity/sensitivity and AUC metrics.

Method findCommunities(): Find cell clusters (as communities on the joint graph)

Usage:
Conos$findCommunities(
    method = leiden.community,
    min.group.size = 0,
    name = NULL,
    test.stability = FALSE,
    stability.subsampling.fraction = 0.95,
    stability.subsamples = 100,
    verbose = TRUE,
    cls = NULL,
    sr = NULL,
    ...
)

Arguments:
method community detection method (igraph syntax) (default=leiden.community)
min.group.size numeric Minimal allowed community size (default=0)
name character Optional name of the clustering result (will default to the algorithm name) (default=NULL will try to obtain the name from the community detection method, or will use 'community' as a default)
test.stability boolean Whether to test stability of community detection (default=FALSE)
stability.subsampling.fraction numeric Fraction of clusters to subset (default=0.95). Must be within range [0, 1].
stability.subsamples integer Number of subsampling iterations (default=100)
verbose boolean Whether to provide verbose output (default=TRUE)
cls optional pre-calculated community result (may be useful for stability testing) (default: NULL)
sr optional pre-calculated subsampled community results (useful for stability testing) (default: NULL)

... extra parameters are passed to the specified community detection method

Returns: invisible list containing identified communities (groups) and the full community detection result (result); The results are stored in $clusters$name slot in the conos object. Each such slot contains an object with elements: $results which stores the raw output of the community detection method, and $groups which is a factor on cells describing the resulting clustering. The later can be used, for instance, in plotting: con$plotGraph(groups=con$clusters$leiden$groups).

If test.stability==TRUE, then the result object will also contain a $stability slot.

Examples:

con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$buildGraph(k=10, k.self=5, space='PCA', ncomps=10, n.odgenes=20, matching.method='mNN', metric='angular', score.component.variance=TRUE, verbose=TRUE)
con$findCommunities(method = igraph::walktrap.community, steps=5)

Method plotPanel(): Plot panel of individual embeddings per sample with joint coloring

Usage:
Conos$plotPanel(
  clustering = NULL,
  groups = NULL,
  colors = NULL,
  gene = NULL,
  use.local.clusters = FALSE,
  plot.theme = NULL,
  use.common.embedding = FALSE,
  embedding = NULL,
  adj.list = NULL,
  ...
)

Arguments:

clustering character Name of the clustering to use (see names(con$clusters)) for the value of the groups factor (default=NULL - if groups are not specified, the first clustering will be used)

groups a cell factor (a factor named with cell names) specifying clusters of cells to be compared (one against all). To compare two cell clusters against each other, simply pass a factor containing only two levels (default=NULL, see clustering)

colors a color factor (named with cell names) use for cell coloring

gene show expression of a gene

use.local.clusters boolean Whether clusters should be taken from the individual samples; otherwise joint clusters in the conos object will be used (see clustering) (default=FALSE).

plot.theme string Theme for the plot, passed to plotSamples() (default=NULL)

use.common.embedding boolean Whether a joint embedding in the conos object should be used (or embeddings determined for the individual samples) (default=FALSE)

embedding (default=NULL) If a character value is passed, it is interpreted as an embedding name (a name of a joint embedding in conos when use.common.embedding=TRUE, or a
name of an embedding within the individual objects when use.common.embedding=FALSE). If a matrix is passed, it is interpreted as an actual embedding (then first two columns are interpreted as x/y coordinates, row names must be cell names). If NULL, the default embedding will be used.

adj.list  an optional list of additional ggplot2 directions to apply (default=NULL)

... Additional parameters passed to plotSamples(), plotEmbeddings(), sccore::embeddingPlot().

Returns: cowplot grid object with the panel of plots

Method embedGraph(): Generate an embedding of a joint graph

Usage:

Conos$embedGraph(
  method = "largeVis",
  embedding.name = method,
  M = 1,
  gamma = 1,
  alpha = 0.1,
  perplexity = NA,
  sgd_batches = 1e+08,
  seed = 1,
  verbose = TRUE,
  target.dims = 2,
  ...
)

Arguments:

method  Embedding method (default='largeVis'). Currently 'largeVis' and 'UMAP' are supported.

embedding.name  character Optional name of the name of the embedding set by user to store multiple embeddings (default: method name)

M numeric (largeVis) The number of negative edges to sample for each positive edge to be used (default=1)

gamma numeric (largeVis) The strength of the force pushing non-neighbor nodes apart (default=1)

alpha numeric (largeVis) Hyperparameter used in the default distance function, \( 1/(1+\alpha||y_i - y_j||^2) \) (default=0.1). The function relates the distance between points in the low-dimensional projection to the likelihood that the two points are nearest neighbors. Increasing \( \alpha \) tends to push nodes and their neighbors closer together; decreasing \( \alpha \) produces a broader distribution. Setting \( \alpha \) to zero enables the alternative distance function. \( \alpha \) below zero is meaningless.

perplexity (largeVis) The perplexity passed to largeVis (default=NA)

sgd_batches (largeVis) The number of edges to process during SGD (default=1e8). Defaults to a value set based on the size of the dataset. If the parameter given is between 0 and 1, the default value will be multiplied by the parameter.

seed numeric Random seed for the largeVis algorithm (default=1)

verbose boolean Whether to provide verbose output (default=TRUE)

target.dims numeric Number of dimensions for the reduction (default=2). Higher dimensions can be used to generate embeddings for subsequent reductions by other methods, such as tSNE
... additional arguments, passed to UMAP embedding (run ?conos:::embedGraphUmap for more info)

**Method** `plotClusterStability()`: Plot cluster stability statistics.

**Usage:**

```r
Conos$plotClusterStability(clustering = NULL, what = "all")
```

**Arguments:**

- `clustering` string Name of the clustering result to show (default=NULL)
- `what` string Show a specific plot (ari - adjusted rand index, fjc - flat Jaccard, hjc - hierarchical Jaccard, dend - cluster dendrogram, all - everything except 'dend') (default='all')

**Returns:** cluster stability statistics

**Method** `plotGraph()`: Plot joint graph

**Usage:**

```r
Conos$plotGraph(
  color.by = "cluster",
  clustering = NULL,
  embedding = NULL,
  groups = NULL,
  colors = NULL,
  gene = NULL,
  plot.theme = NULL,
  subset = NULL,
  ...
)
```

**Arguments:**

- `color.by` character A shortcut to color the plot by 'cluster' or by 'sample' (default: 'cluster'). If any other string is input, an error is thrown.
- `clustering` a character name of the clustering to use (see names(con$clusters)) for the value of the groups factor (default: NULL - if groups are not specified, the first clustering will be used)
- `embedding` A character name of an embedding, or a matrix of the actual embedding (rownames should correspond to cells, first to columns to x/y coordinates). If NULL (default: NULL), the latest generated embedding will be used
- `groups` a cell factor (a factor named with cell names) specifying clusters of cells to be compared (one against all). To compare two cell clusters against each other, simply pass a factor containing only two levels (default: NULL, see clustering)
- `colors` a color factor (named with cell names) use for cell coloring (default=NULL)
- `gene` Show expression of a gene (default=NULL)
- `plot.theme` Theme for the plot, passed to sccore::embeddingPlot() (default=NULL)
- `subset` A subset of cells to show (default: NULL - shows all the cells)

**Returns:** ggplot2 plot of joint graph

**Method** `correctGenes()`: Smooth expression of genes to minimize the batch effect between samples Use diffusion of expression on graph with the equation $dv = \exp(-a * (v + b))$
Usage:
Conos$correctGenes(
    genes = NULL,
    n.od.genes = 500,
    fading = 10,
    fading.const = 0.5,
    max.iters = 15,
    tol = 0.005,
    name = "diffusion",
    verbose = TRUE,
    count.matrix = NULL,
    normalize = TRUE
)

Arguments:
genes List of genes to be smooothed smoothing (default=NULL will smooth top n.od.genes overdispersed genes)
n.od.genes numeric If 'genes' is NULL, top n.od.genes of overdispersed genes are taken across all samples (default=500)
fading numeric Level of fading of expression change from distance on the graph (parameter 'a' of the equation) (default=10)
fading.const numeric Minimal penalty for each new edge during diffusion (parameter 'b' of the equation) (default=0.5)
max.iters numeric Maximal number of diffusion iterations (default=15)
tol numeric Tolerance after which the diffusion stops (default=5e-3)
name string Name to save the correction (default='diffusion')
verbose boolean Verbose mode (default=TRUE)
count.matrix Alternative gene count matrix to correct (rows: genes, columns: cells; has to be dense matrix). Default: joint count matrix for all datasets.
normalize boolean Whether to normalize values (default=TRUE)

Returns: smoothed expression of the input genes

Method propagateLabels(): Estimate labeling distribution for each vertex, based on a partial labeling of the cells. There are two methods used for the propagation to calculate the distribution of labels: "solver" and "diffusion". * "diffusion" (default) will estimate the labeling distribution for each vertex, based on provided labels using a random walk. * "solver" will propagate labels using the algorithm described by Zhu, Ghahramani, Lafferty (2003) <http://mlg.eng.cam.ac.uk/zoubin/papers/zgl.pdf>
Confidence values are then calculated by taking the maximum value from this distribution of labels, for each cell.

Usage:
Conos$propagateLabels(labels, method = "diffusion", ...)

Arguments:
labels Input labels
method type of propagation. Either 'diffusion' or 'solver'. 'solver' gives better result but has bad asymptotics, so is inappropriate for datasets > 20k cells. (default='diffusion')
... additional arguments for conos::propagateLabels* functions
Returns: list with three fields: * labels = matrix with distribution of label probabilities for each vertex by rows. * uncertainty = 1 - confidence values * label.distribution = the distribution of labels calculated using either the methods "diffusion" or "solver"

Method getClusterCountMatrices(): Calculate pseudo-bulk expression matrices for clusters (by adding up, for each gene, all of the molecules detected for all cells in a given cluster in a given sample)

Usage:
Conos$getClusterCountMatrices(
    clustering = NULL,
    groups = NULL,
    common.genes = TRUE,
    omit.na.cells = TRUE
)

Arguments:
clustering string Name of the clustering to use
groups a factor on cells to use for coloring
common.genes boolean Whether to bring individual sample matrices to a common gene list (default=TRUE)
omit.na.cells boolean If set to FALSE, the resulting matrices will include a first column named 'NA' that will report total molecule counts for all of the cells that were not covered by the provided factor. (default=TRUE)

Returns: a list of per-sample uniform dense matrices with rows being genes, and columns being clusters

Method getDatasetPerCell(): applies 'getCellNames()' on all samples

Usage:
Conos$getDatasetPerCell()

Returns: list of cellnames for all samples

Examples:
con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$getDatasetPerCell()

Method getJointCountMatrix(): Retrieve joint count matrices

Usage:
Conos$getJointCountMatrix(raw = FALSE)

Arguments:
raw boolean If TRUE, return merged "raw" count matrices, using function getRawCountMatrix(). Otherwise, return the merged count matrices, using getCountMatrix(). (default=FALSE)

Returns: list of merged count matrices

Examples:
con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$getJointCountMatrix()
**Method** `clone()`: The objects of this class are cloneable with this method.

**Usage:**

```r
Conos$clone(deep = FALSE)
```

**Arguments:**

dean  Whether to make a deep clone.

**Examples**

```r
## Method `Conos$new`

con <- Conos$new(small_panel.preprocessed, n.cores=1)

## Method `Conos$buildGraph`

con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$buildGraph(k=10, k.self=5, space="PCA", ncomps=10, n.odgenes=20, matching.method="mNN", metric="angular", score.component.variance=TRUE, verbose=TRUE)

## Method `Conos$findCommunities`

con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$buildGraph(k=10, k.self=5, space="PCA", ncomps=10, n.odgenes=20, matching.method="mNN", metric="angular", score.component.variance=TRUE, verbose=TRUE)
con$findCommunities(method = igraph::walktrap.community, steps=5)

## Method `Conos$getDatasetPerCell`

con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$getDatasetPerCell()

## Method `Conos$getJointCountMatrix`

con <- Conos$new(small_panel.preprocessed, n.cores=1)
con$getJointCountMatrix()
```
convertToPagoda2

Convert Conos object to Pagoda2 object

Usage

certToPagoda2(con, n.pcs = 100, n.odgenes = 2000, verbose = TRUE, ...)

Arguments

con Conos object
n.pcs numeric Number of principal components (default=100)
n.odgenes numeric Number of overdispersed genes (default=2000)
verbose boolean Whether to give verbose output (default=TRUE)
... parameters passed to Pagoda2$new()

Value
	pagoda2 object

edgeMat<-

Set edge matrix edgeMat with certain values on sample

Description

Set edge matrix edgeMat with certain values on sample
Access edgeMat from sample

Usage

eedgeMat(sample) <- value

## S4 replacement method for signature 'Pagoda2'
edgeMat(sample) <- value

## S4 replacement method for signature 'seurat'
edgeMat(sample) <- value

## S4 replacement method for signature 'Seurat'
edgeMat(sample) <- value
**estimateWeightEntropyPerCell**

Estimate entropy of edge weights per cell according to the specified factor. Can be used to visualize alignment quality according to this factor.

### Usage

```r
estimateWeightEntropyPerCell(con, factor.per.cell)
```

### Arguments

- **con**: conos object
- **factor.per.cell**: some factor, which group cells, such as sample or a specific condition

### Value

entropy of edge weights per cell
findSubcommunities  

**Description**

Increase resolution for a specific set of clusters

**Usage**

```r
findSubcommunities(
  con, 
  target.clusters, 
  clustering = NULL, 
  groups = NULL, 
  method = leiden.community, 
  ... 
)
```

**Arguments**

- `con`  
  conos object
- `target.clusters`  
  clusters for which the resolution should be increased
- `clustering`  
  name of clustering in the conos object to use. Either 'clustering' or 'groups' must be provided (default=NULL).
- `groups`  
  set of clusters to use. Ignored if 'clustering' is not NULL (default=NULL).
- `method`  
  function, used to find communities (default=leiden.community).
- `...`  
  additional params passed to the community function

**Value**

set of clusters with increased resolution

---

getBetweenCellTypeCorrectedDE  

**Description**

Compare two cell types across the entire panel
**Usage**

```r
getBetweenCellTypeCorrectedDE(
  con.obj,
  sample.groups = NULL,
  groups = NULL,
  cooks.cutoff = FALSE,
  refgroup = NULL,
  altgroup = NULL,
  min.cell.count = 10,
  independent.filtering = FALSE,
  cluster.sep.chr = "<!>",
  return.details = TRUE,
  only.paired = TRUE,
  correction = NULL,
  ref.level = NULL
)
```

**Arguments**

- `con.obj`: conos object
- `sample.groups`: a named list of two character vectors specifying the app groups to compare
- `groups`: factor describing cell grouping
- `cooks.cutoff`: cooksCutoff parameter for DESeq2
- `refgroup`: cell type to compare to be used as reference
- `altgroup`: cell type to compare to
- `min.cell.count`: minimum number of cells per celltype/sample combination to keep
- `independent.filtering`: independentFiltering parameter for DESeq2
- `cluster.sep.chr`: character string of length 1 specifying a delimiter to separate cluster and app names
- `return.details`: logical, return detailed results
- `only.paired`: only keep samples that that both cell types above the min.cell.count threshold
- `correction`: fold change corrections per genes
- `ref.level`: reference level on the basis of which the correction was calculated

**Value**

Returns either a DESeq2::results() object, or if return.details=TRUE, returns a list of the DESeq2::results(), the samples from the panel to use in this comparison, refgroups, altgroup, and samplegroups
getBetweenCellTypeDE  
*Compare two cell types across the entire panel*

**Description**

Compare two cell types across the entire panel

**Usage**

```r
getBetweenCellTypeDE(
  con.obj,  
  groups = NULL,  
  sample.groups = NULL,  
  cooks.cutoff = FALSE,  
  refgroup = NULL,  
  altgroup = NULL,  
  min.cell.count = 10,  
  independent.filtering = FALSE,  
  cluster.sep.chr = "<!!>",  
  return.details = TRUE,  
  only.paired = TRUE,  
  remove.na = TRUE
)
```

**Arguments**

- `con.obj` conos object
- `groups` factor describing cell grouping (default=NULL)
- `sample.groups` a named list of two character vectors specifying the app groups to compare (default=NULL)
- `cooks.cutoff` boolean cooksCutoff parameter for DESeq2 (default=FALSE)
- `refgroup` cell type to compare to be used as reference (default=NULL)
- `altgroup` cell type to compare to be used as ALT against refgroup (default=NULL)
- `min.cell.count` numeric Minimum number of cells per celltype/sample combination to keep (default=10)
- `independent.filtering` boolean Whether to use independentFiltering parameter for DESeq2 (default=FALSE)
- `cluster.sep.chr` character string of length 1 specifying a delimiter to separate cluster and app names (default='<!!>')
- `return.details` boolean Return detailed results (default=TRUE)
- `only.paired` boolean Only keep samples that that both cell types above the min.cell.count threshold (default=TRUE)
- `remove.na` boolean If TRUE, remove NAs from DESeq calculations (default=TRUE)
**getCellNames**

**Value**

Returns either a DESeq2::results() object, or if return.details=TRUE, returns a list of the DESeq2::results(), the samples from the panel to use in this comparison, refgroups, altgroup, and samplegroups

---

**getCellNames**  
**Access cell names from sample**

**Description**

Access cell names from sample

**Usage**

ggetCellNames(sample)

```r
## S4 method for signature 'Pagoda2'
ggetCellNames(sample)

## S4 method for signature 'seurat'
ggetCellNames(sample)

## S4 method for signature 'Seurat'
ggetCellNames(sample)

## S4 method for signature 'Conos'
ggetCellNames(sample)
```

**Arguments**

- `sample` sample from which to cell names

---

**getClustering**  
**Access clustering from sample**

**Description**

Access clustering from sample
Usage

getClustering(sample, type)

## S4 method for signature 'Pagoda2'
getClustering(sample, type)

## S4 method for signature 'seurat'
getClustering(sample, type)

## S4 method for signature 'Seurat'
getClustering(sample, type)

## S4 method for signature 'Conos'
getClustering(sample, type)

Arguments

sample sample from which to get the clustering
type character Type of clustering to get

getCountMatrix Access count matrix from sample

Description

Access count matrix from sample

Usage

getCountMatrix(sample, transposed = FALSE)

## S4 method for signature 'Pagoda2'
getCountMatrix(sample, transposed = FALSE)

## S4 method for signature 'seurat'
getCountMatrix(sample, transposed = FALSE)

## S4 method for signature 'Seurat'
getCountMatrix(sample, transposed = FALSE)

## S4 method for signature 'Conos'
getCountMatrix(sample, transposed = FALSE)

Arguments

sample sample from which to get the count matrix
transposed boolean Whether the count matrix should be transposed (default=FALSE)
getEmbedding

Access embedding from sample

Description
Access embedding from sample

Usage
getEmbedding(sample, type)

## S4 method for signature 'Pagoda2'
getEmbedding(sample, type)

## S4 method for signature 'seurat'
getEmbedding(sample, type)

## S4 method for signature 'Seurat'
getEmbedding(sample, type)

## S4 method for signature 'Conos'
getEmbedding(sample, type)

Arguments

sample sample from which to get the embedding
type character Type of embedding to get

getGeneExpression

Access gene expression from sample

Description
Access gene expression from sample

Usage
getGeneExpression(sample, gene)

## S4 method for signature 'Pagoda2'
geneExpression(sample, gene)

## S4 method for signature 'Conos'
geneExpression(sample, gene)
getGenes

## S4 method for signature 'Seurat'
getGeneExpression(sample, gene)

## S4 method for signature 'seurat'
getGeneExpression(sample, gene)

Arguments

- **sample**: sample from which to access gene expression
- **gene**: character vector Genes to access

Description

Access genes from sample

Usage

getGenes(sample)

## S4 method for signature 'Pagoda2'
getGenes(sample)

## S4 method for signature 'seurat'
getGenes(sample)

## S4 method for signature 'Seurat'
getGenes(sample)

## S4 method for signature 'Conos'
getGenes(sample)

Arguments

- **sample**: sample from which to get genes
**getOverdispersedGenes**  
*Access overdispersed genes from sample*

**Description**  
Access overdispersed genes from sample

**Usage**

```r
getOverdispersedGenes(sample, n.odgenes = 1000)
```

## S4 method for signature 'Pagoda2'
getOverdispersedGenes(sample, n.odgenes = NULL)

## S4 method for signature 'seurat'
getOverdispersedGenes(sample, n.odgenes = NULL)

## S4 method for signature 'Seurat'
getOverdispersedGenes(sample, n.odgenes = NULL)

## S4 method for signature 'Conos'
getOverdispersedGenes(sample, n.odgenes = NULL)

**Arguments**

- `sample`: sample from which to overdispersed genes
- `n.odgenes`: numeric Number of overdispersed genes to get

---

**getPca**  
*Access PCA from sample*

**Description**

Access PCA from sample

**Usage**

```r
getPca(sample)
```

## S4 method for signature 'Pagoda2'
getPca(sample)

## S4 method for signature 'seurat'
getPca(sample)

## S4 method for signature 'Seurat'
getPca(sample)

## S4 method for signature 'Conos'
getPca(sample)
getPerCellTypeDE

Do differential expression for each cell type in a conos object between the specified subsets of apps

Arguments

sample sample from which to access PCA

groups A factor specifying cell types.
sample.groups A list of two character vector specifying the app groups to compare.
cooks.cutoff Boolean cooksCutoff for DESeq2.
ref.level The reference level of the sample.groups against which the comparison should be made. If NULL, will pick the first one.
min.cell.count Integer Minimal number of cells per cluster for a sample to be taken into account in a comparison.
remove.na Boolean If TRUE, remove NAs from DESeq calculations, which often arise as comparisons not possible.
max.cell.count Maximal number of cells per cluster per sample to include in a comparison.
test Which DESeq2 test to use (options: "LRT" or "Wald").

Usage

getPerCellTypeDE(con.obj, groups = NULL, sample.groups = NULL, cooks.cutoff = FALSE, ref.level = NULL, min.cell.count = 10, remove.na = TRUE, max.cell.count = Inf, test = "LRT", independent.filtering = FALSE, n.cores = 1, cluster.sep.chr = "<!!>", return.details = TRUE)

Arguments

con.obj Conos object.
groups Factor specifying cell types (default=NULL).
sample.groups A list of two character vector specifying the app groups to compare (default=NULL).
cooks.cutoff Boolean cooksCutoff for DESeq2 (default=FALSE).
ref.level The reference level of the sample.groups against which the comparison should be made (default=NULL). If NULL, will pick the first one.
min.cell.count Integer Minimal number of cells per cluster for a sample to be taken into account in a comparison (default=10).
remove.na Boolean If TRUE, remove NAs from DESeq calculations, which often arise as comparisons not possible (default=TRUE).
max.cell.count Maximal number of cells per cluster per sample to include in a comparison (useful for comparing the number of DE genes between cell types) (default=Inf).
test Which DESeq2 test to use (options: "LRT" or "Wald") (default="LRT")
getRawCountMatrix

**independent.filtering**
```
boolean independentFiltering for DESeq2 (default=FALSE)
```

**n.cores**
```
numeric Number of cores (default=1)
```

**cluster.sep.chr**
```
character string of length 1 specifying a delimiter to separate cluster and app names (default='<!!>')</n```

**return.details**
```
boolean Whether to return verbose details (default=TRUE)
```

**Value**
A list of differential expression results for every cell type

---

**getRawCountMatrix**  
*Access raw count matrix from sample*

**Description**
Access raw count matrix from sample

**Usage**
```
getRawCountMatrix(sample, transposed = FALSE)
```
```
# S4 method for signature 'Pagoda2'
getRawCountMatrix(sample, transposed = FALSE)
```
```
# S4 method for signature 'seurat'
getRawCountMatrix(sample, transposed = FALSE)
```
```
# S4 method for signature 'Seurat'
getRawCountMatrix(sample, transposed = FALSE)
```
```
# S4 method for signature 'Conos'
getRawCountMatrix(sample, transposed = FALSE)
```

**Arguments**
```
sample  sample from which to get the raw count matrix
```
```
transposed  boolean Whether the raw count matrix should be transposed (default=FALSE)
```
getSampleNamePerCell  Retrieve sample names per cell

Description
Retrieve sample names per cell

Usage
getSampleNamePerCell(samples)

Arguments
samples  list of samples

Value
list of sample names getSampleNamePerCell(small_panel.preprocessed)

greedyModularityCut  Performs a greedy top-down selective cut to optimize modularity

Description
Performs a greedy top-down selective cut to optimize modularity

Usage
greedyModularityCut(
  wt,
  N,
  leaf.labels = NULL,
  minsize = 0,
  minbreadth = 0,
  flat.cut = TRUE
)

Arguments
wt  walktrap result
N  numeric Number of top greedy splits to take
leaf.labels  leaf sample label factor, for breadth calculations - must be a named factor containing all wt$names, or if wt$names is null, a factor listing cells in the same order as wt leafs (default=NULL)
minsize  numeric Minimum size of the branch (in number of leafs) (default=0)
p2app4conos

Utility function to generate a pagoda2 app from a conos object

Description

Utility function to generate a pagoda2 app from a conos object

Usage

p2app4conos(
    conos,
    cdl = NULL,
    metadata = NULL,
    filename = "conos_app.bin",
    save = TRUE,
    n.cores = 1,
    n.odgenes = 3000,
    nPcs = 100,
    k = 30,
    perplexity = 50,
    log.scale = TRUE,
    trim = 10,
    keep.genes = NULL,
    min.cells.per.gene = 0,
    min.transcripts.per.cell = 100,
    get.largevis = TRUE,
    get.tsne = TRUE,
    make.geneknn = TRUE,
    go.env = NULL,
    cell.subset = NULL,
    max.cells = Inf,
    additional.embeddings = NULL,
    test.pathway.overdispersion = FALSE,
    organism = NULL,
    return.details = FALSE
)
Arguments

conos  Conos object

cdl  list Optional list of raw matrices (so that gene merging doesn’t have to be redone) (default=NULL)

metadata  list Optional list of (named) metadata factors (default=NULL)

filename  string Name of the *.bin file to serialize for the pagoda2 application if save=TRUE (default=’conos_app.bin’)

save  boolean Save serialized *bin file specified in filename (default=TRUE)

n.cores  integer Number of cores (default=1)

n.odgenes  numeric Number of top overdispersed genes to use (dfault=3e3). From pagoda2::basicP2proc().

nPcs  numeric Number of PCs to use (default=100). From pagoda2::basicP2proc().

k  numeric Default number of neighbors to use in kNN graph (default=30). From pagoda2::basicP2proc().

perplexity  numeric Perplexity to use in generating tSNE and largeVis embeddings (default=50). From pagoda2::basicP2proc().

log.scale  boolean Whether to use log scale normalization (default=TRUE). From pagoda2::basicP2proc().

trim  numeric Number of cells to trim in winsorization (default=10). From pagoda2::basicP2proc().

keep.genes  optional set of genes to keep from being filtered out (even at low counts) (default=NULL). From pagoda2::basicP2proc().

min.cells.per.gene  numeric Minimal number of cells required for gene to be kept (unless listed in keep.genes) (default=0). From pagoda2::basicP2proc().

min.transcripts.per.cell  numeric Minimual number of molecules/reads for a cell to be admitted (default=100). From pagoda2::basicP2proc().

get.largevis  boolean Whether to calculate largeVis embedding (default=TRUE). From pagoda2::basicP2proc().

get.tsne  boolean Whether to calculate tSNE embedding (default=TRUE). From pagoda2::basicP2proc().

make.geneknn  boolean Whether pre-calculate gene kNN (for gene search) (default=TRUE). From pagoda2::basicP2proc().

go.env  GO environment for the organism of interest (default=NULL)

cell.subset  string Cells to subset with the conos embedding conos$embedding. If NULL, uses all cells via rownames(conos$embedding) (default=NULL)

max.cells  numeric Limit to the cells that are included in the conos. If Inf, there is no limit (default=Inf)

additional.embeddings  list Additional embeddings to add to conos for the pagoda2 app (default=NULL)

test.pathway.overdispersion  boolean Find all IDs using GO category against either org.Hs.eg.db (‘hs’) or org.Mm.eg.db (‘mm’) (default=FALSE)

organism  string Organism of interest, either ‘hs’ (Homo sapiens) or ‘mm’ (Mus musculus, i.e. mouse) (default=NULL). Only used if test.pathway.overdispersion is TRUE. If NULL and test.pathway.overdispersion=TRUE, then ‘hs’ is used.
plotClusterBarplots

return.details  boolean If TRUE, return list of p2 application, pagoda2 object, list of raw matrices, and cell names. If FALSE, simply return pagoda2 app object. (default=FALSE)

Value

pagoda2 app object

plotClusterBarplots  Plots barplots per sample of composition of each pagoda2 application based on selected clustering

Description

Plots barplots per sample of composition of each pagoda2 application based on selected clustering

Usage

plotClusterBarplots(
  conos.obj = NULL,
  clustering = NULL,
  groups = NULL,
  sample.factor = NULL,
  show.entropy = TRUE,
  show.size = TRUE,
  show.composition = TRUE,
  legend.height = 0.2
)

Arguments

  conos.obj  A conos object (default=NULL)
  clustering  name of clustering in the current object (default=NULL)
  groups  arbitrary grouping of cells (to use instead of the clustering) (default=NULL)
  sample.factor  a factor describing cell membership in the samples (or some other category) (default=NULL). This will default to samples if not provided.
  show.entropy  boolean Whether to include entropy barplot (default=TRUE)
  show.size  boolean Whether to include size barplot (default=TRUE)
  show.composition  boolean Whether to include composition barplot (default=TRUE)
  legend.height  numeric Relative hight of the legend panel (default=0.2)

Value

  a ggplot object
plotClusterBoxPlotsByAppType

Generate boxplot per cluster of the proportion of cells in each celltype

Description

Generate boxplot per cluster of the proportion of cells in each celltype

Usage

plotClusterBoxPlotsByAppType(
  conos.obj,
  clustering = NULL,
  apptypes = NULL,
  return.details = FALSE
)

Arguments

- conos.obj: conos object
- clustering: name of the clustering to use (default=NULL)
- apptypes: a factor specifying how to group the samples (default=NULL)
- return.details: boolean If TRUE return a list with the plot and the summary data.frame (default=FALSE)

Value

Boxplot per cluster of the proportion of cells in each celltype

plotComponentVariance

Plot fraction of variance explained by the successive reduced space components (PCA, CPCA)

Description

Requires buildGraph() or updatePairs() to be ran first with the argument score.component.variance=TRUE.

Usage

plotComponentVariance(
  conos.obj,
  space = "PCA",
  plot.theme = ggplot2::theme_bw()
)
plotDEheatmap

Arguments

conos.obj     conos object
space         character Reduction space to be analyzed (currently, component variance scoring is only supported by PCA and CPCA) (default='PCA')
plot.theme    ggplot theme (default=ggplot2::theme_bw()). Refer to https://ggplot2.tidyverse.org/reference/ggtheme.html for more details.

Value

ggplot

Description

Plot a heatmap of differential genes

Usage

plotDEheatmap(
  con,
  groups,
  de = NULL,
  min.auc = NULL,
  min.specificity = NULL,
  min.precision = NULL,
  n.genes.per.cluster = 10,
  additional.genes = NULL,
  exclude.genes = NULL,
  labeled.gene.subset = NULL,
  expression.quantile = 0.99,
  pal = colorRampPalette(c("dodgerblue1", "grey95", "indianred1"))(1024),
  ordering = "-AUC",
  column.metadata = NULL,
  show.gene.clusters = TRUE,
  remove.duplicates = TRUE,
  column.metadata.colors = NULL,
  show.cluster.legend = TRUE,
  show_heatmap_legend = FALSE,
  border = TRUE,
  return.details = FALSE,
  row.label.font.size = 10,
  order.clusters = FALSE,
  split = FALSE,
  split.gap = 0,
)
cell.order = NULL,
  averaging.window = 0,
  max.cells = Inf,
  ...
)

Arguments

con conos (or p2) object

groups groups in which the DE genes were determined (so that the cells can be ordered correctly)

de differential expression result (list of data frames) (default=NULL)

min.auc optional minimum AUC threshold (default=NULL)

min.speciation optional minimum specificity threshold (default=NULL)

min.precision optional minimum precision threshold (default=NULL)

n.genes.per.cluster numeric Number of genes to show for each cluster (default=10)

additional.genes optional additional genes to include (the genes will be assigned to the closest cluster) (default=NULL)

exclude.genes an optional list of genes to exclude from the heatmap (default=NULL)

labeled.gene.subset a subset of gene names to show (instead of all genes) (default=NULL). Can be a vector of gene names, or a number of top genes (in each cluster) to show the names for.

expression.quantile numeric Expression quantile to show (default=0.99)

pal palette to use for the main heatmap (default=colorRampPalette(c('dodgerblue1','grey95','indianred1'))(1024))

ordering order by which the top DE genes (to be shown) are determined (default "-AUC")

column.metadata additional column metadata, passed either as a data.frame with rows named as cells, or as a list of named cell factors (default=NULL).

show.gene.clusters whether to show gene cluster color codes

remove.duplicates remove duplicated genes (leaving them in just one of the clusters)

column.metadata.colors a list of color specifications for additional column metadata, specified according to the HeatmapMetadata format. Use "clusters" slot to specify cluster colors.

show.cluster.legend boolean Whether to show the cluster legend (default=TRUE)

show_heatmap_legend boolean Whether to show the expression heatmap legend (default=FALSE)
**projectKNNs**

Project a distance matrix into a lower-dimensional space.

**Description**

Takes as input a sparse matrix of the edge weights connecting each node to its nearest neighbors, and outputs a matrix of coordinates embedding the inputs in a lower-dimensional space.

**Usage**

```r
projectKNNs(
  wij,
  dim = 2,
  sgd_batches = NULL,
  M = 5,
  gamma = 7,
  alpha = 1,
  rho = 1,
  coords = NULL,
```
Arguments

wij A symmetric sparse matrix of edge weights, in C-compressed format, as created with the Matrix package.
dim numeric Number of dimensions for the projection space (default=2).
sgd_batches The number of edges to process during SGD (default=NULL). Defaults to a value set based on the size of the dataset. If the parameter given is between 0 and 1, the default value will be multiplied by the parameter.
M numeric Number of negative edges to sample for each positive edge (default=5).
gamma numeric Strength of the force pushing non-neighbor nodes apart (default=7).
alpha numeric Hyperparameter used in the default distance function, \(1 / (1 + \alpha ||y_i - y_j||^2)\) (default=1). The function relates the distance between points in the low-dimensional projection to the likelihood that the two points are nearest neighbors. Increasing \(\alpha\) tends to push nodes and their neighbors closer together; decreasing \(\alpha\) produces a broader distribution. Setting \(\alpha\) to zero enables the alternative distance function. \(\alpha\) below zero is meaningless.
rho numeric Initial learning rate (default=1)
coords An initialized coordinate matrix (default=NULL).
useDegree boolean Whether to use vertex degree to determine weights (default=FALSE). If TRUE, weights determined in negative sampling; if FALSE, weights determined by the sum of the vertex’s edges. See Notes.
momentum If not NULL (the default), SGD with momentum is used, with this multiplier, which must be between 0 and 1. Note that momentum can drastically speed-up training time, at the cost of additional memory consumed.
seed numeric Random seed to be passed to the C++ functions (default=NULL). If NULL, sampled from hardware entropy pool. Note that if the seed is not NULL (the default), the maximum number of threads will be set to 1 in phases of the algorithm that would otherwise be non-deterministic.
threads numeric The maximum number of threads to spawn (default=NULL). Determined automatically if NULL.
verbose boolean Verbosity (default=getOption("verbose", TRUE))

Details

The algorithm attempts to estimate a \(\text{dim}\)-dimensional embedding using stochastic gradient descent and negative sampling.
The objective function is:

\[
O = \sum_{(i,j) \in E} w_{ij} (\log f(||p(e_{ij} = 1||)) + \sum_{k=1}^{M} E_{jk} p_{s(j)} \gamma \log (1 - f(||p(e_{ij k} - 1||)))
\]

where \(f()\) is a probabilistic function relating the distance between two points in the low-dimensional projection space, and the probability that they are nearest neighbors.

The default probabilistic function is \(1/(1 + \alpha \|x\|^2)\). If \(\alpha\) is set to zero, an alternative probabilistic function, \(1/(1 + \exp(x^2))\) will be used instead.

Note that the input matrix should be symmetric. If any columns in the matrix are empty, the function will fail.

**Value**

A dense \([N,D]\) matrix of the coordinates projecting the \(w_{ij}\) matrix into the lower-dimensional space.

**Note**

If specified, \(seed\) is passed to the C++ and used to initialize the random number generator. This will not, however, be sufficient to ensure reproducible results, because the initial coordinate matrix is generated using the \(R\) random number generator. To ensure reproducibility, call \texttt{set.seed}\ before calling this function, or pass it a pre-allocated coordinate matrix.

The original paper called for weights in negative sampling to be calculated according to the degree of each vertex, the number of edges connecting to the vertex. The reference implementation, however, uses the sum of the weights of the edges to each vertex. In experiments, the difference was imperceptible with small (MNIST-size) datasets, but the results seems aesthetically preferable using degree. The default is to use the edge weights, consistent with the reference implementation.

**Examples**

```r
## Not run:
data(CO2)
CO2$Plant <- as.integer(CO2$Plant)
CO2$Type <- as.integer(CO2$Type)
CO2$Treatment <- as.integer(CO2$Treatment)
co <- scale(as.matrix(CO2))
# Very small datasets often produce a warning regarding the alias table. This is safely ignored.
suppressWarnings(vis <- largeVis(t(co), K = 20, sgd_batches = 1, threads = 2))
suppressWarnings(coords <- projectKNNs(vis$wij, threads = 2))
plot(t(coords))
## End(Not run)
```
rawMatricesWithCommonGenes

Get raw matrices with common genes

Description
Get raw matrices with common genes

Usage
rawMatricesWithCommonGenes(con.obj, sample.groups = NULL)

Arguments
- con.obj: Conos object
- sample.groups: list of samples to select from Conos object, con.obj$samples (default=NULL)

Value
raw matrices subset with common genes

saveConosForScanPy

Save Conos object on disk to read it from ScanPy

Description
Save Conos object on disk to read it from ScanPy

Usage
saveConosForScanPy(
    con,
    output.path,
    hdf5_filename,
    metadata.df = NULL,
    cm.norm = FALSE,
    pseudo.pca = FALSE,
    pca = FALSE,
    n.dims = 100,
    embedding = TRUE,
    alignment.graph = TRUE,
    verbose = FALSE
 )
saveDEasCSV

Arguments

- **con**: conos object
- **output.path**: path to a folder, where intermediate files will be saved
- **hdf5_filename**: name of HDF5 written with ScanPy files. Note: the rhdf5 package is required
- **metadata.df**: data.frame with additional metadata with rownames corresponding to cell ids, which should be passed to ScanPy (default=NULL) If NULL, only information about cell ids and origin dataset will be saved.
- **cm.norm**: boolean Whether to include the matrix of normalised counts (default=FALSE).
- **pseudo.pca**: boolean Whether to produce an emulated PCA by embedding the graph to a space with ‘n.dims’ dimensions and save it as a pseudoPCA (default=FALSE).
- **pca**: boolean Whether to include PCA of all the samples (not batch corrected) (default=FALSE).
- **n.dims**: numeric Number of dimensions for calculating PCA and/or pseudoPCA (default=100).
- **alignment.graph**: boolean Whether to include graph of connectivities and distances (default=TRUE).
- **verbose**: boolean Whether to use verbose mode (default=FALSE)

Value

AnnData object for ScanPy, saved to disk

See Also


---

saveDEasCSV **Save differential expression as table in *csv format**

Description

Save differential expression as table in *csv format

Usage

```r
saveDEasCSV(de.results, saveprefix, gene.metadata = NULL)
```

Arguments

- **de.results**: output of differential expression results, corrected or uncorrected
- **saveprefix**: character prefix for output file
- **gene.metadata**: gene metadta to include (default=NULL)
saveDEasJSON

*Save differential expression results as JSON*

**Description**

Save differential expression results as JSON

**Usage**

```r
saveDEasJSON(
  de.results = NULL,
  saveprefix = NULL,
  gene.metadata = NULL,
  cluster.sep.chr = "<!!>"
)
```

**Arguments**

- **de.results**
  - differential expression results (default=NULL)
- **saveprefix**
  - prefix for the differential expression output (default=NULL)
- **gene.metadata**
  - data.frame with gene metadata (default=NULL)
- **cluster.sep.chr**
  - character string of length 1 specifying a delimiter to separate cluster and app names (default='<!>')</n
**Value**

JSON with DE results

**scanKModularity**

*Scan joint graph modularity for a range of k (or k.self) values Builds graph with different values of k (or k.self if scan.k.self=TRUE), evaluating modularity of the resulting multilevel clustering NOTE: will run evaluations in parallel using con$n.cores (temporarily setting con$n.cores to 1 in the process)*

**Description**

Scan joint graph modularity for a range of k (or k.self) values Builds graph with different values of k (or k.self if scan.k.self=TRUE), evaluating modularity of the resulting multilevel clustering NOTE: will run evaluations in parallel using con$n.cores (temporarily setting con$n.cores to 1 in the process)
Usage

```r
scanKModularity(
    con,
    min = 3,
    max = 50,
    by = 1,
    scan.k.self = FALSE,
    omit.internal.edges = TRUE,
    verbose = TRUE,
    plot = TRUE,
    ...
)
```

Arguments

- `con` Conos object to test
- `min` numeric Minimal value of k to test (default=3)
- `max` numeric Value of k to test (default=50)
- `by` numeric Scan step (default=1)
- `scan.k.self` boolean Whether to test dependency on scan.k.self (default=FALSE)
- `omit.internal.edges` boolean Whether to omit internal edges of the graph (default=TRUE)
- `verbose` boolean Whether to provide verbose output (default=TRUE)
- `plot` boolean Whether to plot the output (default=TRUE)
- `...` other parameters will be passed to con$buildGraph()

Value

a data frame with $k$ $m$ columns giving k and the corresponding modularity

---

**sgdBatches**

`sgdBatches(N, E = 150 * N/2)`

Calculate the default number of batches for a given number of vertices and edges. The formula used is the one used by the 'largeVis' reference implementation. This is substantially less than the recommendation $E * 10000$ in the original paper.

Description

Calculate the default number of batches for a given number of vertices and edges. The formula used is the one used by the 'largeVis' reference implementation. This is substantially less than the recommendation $E * 10000$ in the original paper.

Usage

`sgdBatches(N, E = 150 * N/2)`
Arguments

N  Number of vertices
E  Number of edges (default = 150*N/2)

Value

The recommended number of sgd batches.

Examples

# Observe that increasing K has no effect on processing time
N <- 70000 # MNIST
K <- 10:250
plot(K, sgdBatches(rep(N, length(K)), N * K / 2))

# Observe that processing time scales linearly with N
N <- c(seq(from = 1, to = 10000, by = 100), seq(from = 10000, to = 10000000, by = 1000))
plot(N, sgdBatches(N))
stableTreeClusters

Determine number of detectable clusters given a reference walktrap and a bunch of permuted walktraps

Description

Determine number of detectable clusters given a reference walktrap and a bunch of permuted walktraps

Usage

```r
stableTreeClusters(
  refwt,
  tests,
  min.threshold = 0.8,
  min.size = 10,
  n.cores = 30,
  average.thresholds = FALSE
)
```

Arguments

- `refwt` reference walktrap result
- `tests` a list of permuted walktrap results
- `min.threshold` numeric Min detectability threshold (default=0.8)
- `min.size` numeric Minimum cluster size (number of leaves) (default=10)
- `n.cores` numeric Number of cores (default=30)
- `average.thresholds` boolean Report a single number of detectable clusters for averaged detected thresholds (default=FALSE) (a list of detected clusters for each element of the tests list is returned by default)

Value

number of detectable stable clusters
Index

* datasets
  - small_panel.preprocessed, 44

basicSeuratProc, 3
bestClusterThresholds, 4
bestClusterTreeThresholds, 4
buildWijMatrix, 5
Conos, 6
convertToPagoda2, 18

dgeMat (edgeMat<-), 18
dgeMat,Pagoda2-method (edgeMat<-), 18
dgeMat,Seurat-method (edgeMat<-), 18
dgeMat,seurat-method (edgeMat<-), 18
dgeMat<-, 18
dgeMat<-,Pagoda2-method (edgeMat<-), 18
dgeMat<-,Seurat-method (edgeMat<-), 18
dgeMat<-,seurat-method (edgeMat<-), 18
estimateWeightEntropyPerCell, 19

findSubcommunities, 20
getBetweenCellTypeCorrectedDE, 20
getBetweenCellTypeDE, 22
getCellNames, 23
getCellNames,Conos-method (getCellNames), 23
getCellNames,Pagoda2-method (getCellNames), 23
getCellNames,Seurat-method (getCellNames), 23
getCellNames,seurat-method (getCellNames), 23
getClustering, 23
getClustering,Conos-method (getClustering), 23
getClustering,Pagoda2-method (getClustering), 23
getClustering,Seurat-method (getClustering), 23
getClustering,seurat-method (getClustering), 23

getCountMatrix, 24
getCountMatrix,Pagoda2-method (getCountMatrix), 24
getCountMatrix,Seurat-method (getCountMatrix), 24
getCountMatrix,seurat-method (getCountMatrix), 24

getEmbedding, 25
getEmbedding,Conos-method (getEmbedding), 25
getEmbedding,Pagoda2-method (getEmbedding), 25
getEmbedding,Seurat-method (getEmbedding), 25
getEmbedding,seurat-method (getEmbedding), 25

getGeneExpression, 25
getGeneExpression,Conos-method (getGeneExpression), 25
getGeneExpression,Pagoda2-method (getGeneExpression), 25
getGeneExpression,Seurat-method (getGeneExpression), 25
getGeneExpression,seurat-method (getGeneExpression), 25

getGenes, 26
getGenes,Conos-method (getGenes), 26
getGenes,Pagoda2-method (getGenes), 26
getGenes,Seurat-method (getGenes), 26
getGenes,seurat-method (getGenes), 26
getOverdispersedGenes, 27
getOverdispersedGenes,Conos-method (getOverdispersedGenes), 27
getOverdispersedGenes,Pagoda2-method (getOverdispersedGenes), 27
getOverdispersedGenes,Seurat-method (getOverdispersedGenes), 27
getOverdispersedGenes,seurat-method (getOverdispersedGenes), 27

46
getOverdispersedGenes, seurat-method
  (getOverdispersedGenes), 27
getPca, 27
getPca, Pagoda2-method (getPca), 27
getcPca, Seurat-method (getPca), 27
getcPca, seurat-method (getPca), 27
getPerCellTypeDE, 28
getRawCountMatrix, 29
getcRawCountMatrix, Conos-method
  (getRawCountMatrix), 29
getcRawCountMatrix, Pagoda2-method
  (getRawCountMatrix), 29
getcRawCountMatrix, Seurat-method
  (getRawCountMatrix), 29
getcRawCountMatrix, seurat-method
  (getRawCountMatrix), 29
getsSampleNamePerCell, 30
greedyModularityCut, 30
p2app4conos, 31
plotClusterBarplots, 33
plotClusterBoxPlotsByAppType, 34
plotComponentVariance, 34
plotDEheatmap, 35
projectKNNs, 37
rawMatricesWithCommonGenes, 40
saveConosForScanPy, 40
saveDEasCSV, 41
saveDEasJSON, 42
scanKModularity, 42
set.seed, 39
sgdBatches, 43
small_panel.preprocessed, 44
stableTreeClusters, 45