Package 'poem'

April 22, 2025

Title POpulation-based Evaluation Metrics

Version 1.0.0

Description
This package provides a comprehensive set of external and internal evaluation metrics. It includes metrics for assessing partitions or fuzzy partitions derived from clustering results, as well as for evaluating
subpopulation identification results within embeddings or graph representations. Additionally, it provides metrics for
comparing spatial domain detection results against ground truth labels, and tools for visualizing spatial errors.

BugReports https://github.com/RoseYuan/poem/issues

License GPL (>= 3)

URL https://roseyuan.github.io/poem/

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.0)

Imports aricode, BiocNeighbors, BiocParallel, bluster, clevr, clue, cluster, elsa, fclust, igraph, Matrix, matrixStats, mclustcomp, methods, pdist, sp, spdep, stats, utils, SpatialExperiment, SummarizedExperiment

Suggests testthat (>= 3.0.0), BiocStyle, knitr, DT, dplyr, kableExtra, scico, cowplot, ggnetwork, ggplot2, tidyr, STexampleData

VignetteBuilder knitr

biocViews DimensionReduction, Clustering, GraphAndNetwork, Spatial, ATACSeq, SingleCell, RNASeq, Software, Visualization

Config/testthat/edition 3

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

git_branch RELEASE_3_21

git_last_commit 4510cc2

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-04-21

Author Siyuan Luo [cre, aut] (ORCID: <https://orcid.org/0009-0007-6404-3244>), Pierre-Luc Germain [aut, ctb] (ORCID: <https://orcid.org/0000-0003-3418-4218>)

Maintainer Siyuan Luo <roseluosy@gmail.com>

Contents

`.check_duplicated_samples` *Check Duplicated Samples*

Description

Checks for duplicated samples in matrix `X`.

Usage

```
.check_duplicated_samples(X, threshold = 1e-09)
```

Arguments

`X`	Numeric matrix of samples.
`threshold`	Numeric, the distance threshold to consider samples as duplicates.

Value

None

`.compute_cluster_core_distance` *Compute Cluster Core Distance*

Description

Computes the core distance for each point in a cluster.

Usage

```
.compute_cluster_core_distance(dists, d)
```

Arguments

`dists`	Numeric matrix of distances.
`d`	Integer, the dimensionality.

Value

Numeric vector of core distances for each point.

`.compute_mutual_reach_dists` *Compute Mutual Reachability Distances*

Description

Computes the mutual reachability distances between points.

Usage

```
.compute_mutual_reach_dists(dists, d)
```

Arguments

`dists`	Numeric matrix of distances.
`d`	Float, the dimensionality.

Value

A list containing core distances and mutual reachability distances.

.compute_pair_to_pair_dists *Compute Pair to Pair Distances*

Description

Compute the pairwise distances between points in matrix `X`.

Usage

```
.compute_pair_to_pair_dists(X, distance = "euclidean")
```

Arguments

`X`	Numeric matrix.
`distance`	String specifying the metric to compute the distances.

Value

Numeric matrix of pairwise distances with self-distances set to `Inf`.

.convert_singleton_clusters_to_noise *Convert Singleton Clusters to Noise*

Description

Converts clusters containing a single instance to noise.

Usage

```
.convert_singleton_clusters_to_noise(labels, noise_id)
```

Arguments

`labels`	Integer vector of cluster IDs.
`noise_id`	Integer, the ID for noise.

Value

Integer vector with singleton clusters converted to noise.

.fn_density_separation

Density Separation of a Pair of Clusters

Description

Computes the density separation between two clusters.

Usage

```
.fn_density_separation(
  cls_i,
  cls_j,
  dists,
  internal_core_dists_i,
  internal_core_dists_j
)
```

Arguments

`cls_i`	Integer, first cluster index.
`cls_j`	Integer, second cluster index.
`dists`	Numeric matrix of distances.
`internal_core_dists_i`	Numeric vector of core distances for cluster i.
`internal_core_dists_j`	Numeric vector of core distances for cluster j.

Value

A list containing the cluster indices and their density separation.

.fn_density_sparseness

Density Sparseness of a Cluster

Description

Computes the density sparseness for a given cluster.

Usage

```
.fn_density_sparseness(cls_inds, dists, d, use_igraph_mst)
```

Arguments

`cls_inds`	Integer vector of cluster indices.
`dists`	Numeric matrix of distances.
`d`	Integer, the dimensionality.
`use_igraph_mst`	Logical flag to use MST implementation in igraph. Currently only mst from igraph is implemented.

Value

A list containing the density sparseness, internal core distances, and internal node indices.

`.get_internal_objects` *Get Internal Objects*

Description

Computes the internal nodes and edges using Minimum Spanning Tree.

Usage

```
.get_internal_objects(mutual_reach_dists, use_igraph_mst = TRUE)
```

Arguments

`mutual_reach_dists`	Numeric matrix representing mutual reachability distances.
`use_igraph_mst`	Logical flag to use MST implementation in igraph. Currently only mst from igraph is implemented.

Value

A list containing the indices of internal nodes and their edge weights.

.get_submatrix *Get Sub matrix*

Description

Extract a sub matrix from a matrix based on optional row and column indices.

Usage

```
.get_submatrix(arr, inds_a = NULL, inds_b = NULL)
```

Arguments

arr	Numeric matrix.
inds_a	Optional integer vector for row indices.
inds_b	Optional integer vector for column indices.

Value

Numeric matrix representing the sub matrix.

CDbw *Calculate CDbw index*

Description

Computes the CDbw-index (Halkidi and Vazirgiannis 2008; Halkidi, Vazirgiannis and Hennig, 2015). This function is directly copied from the fpc CRAN package and was written by Christian Hennig. It is included here to reduce the package dependencies (since fpc has a few not-so-light dependencies that aren't required here).

Usage

```
CDbw(
  x,
  labels,
  r = 10,
  s = seq(0.1, 0.8, by = 0.1),
  clusterstdev = TRUE,
  trace = FALSE
)
```

Arguments

x	Something that can be coerced into a numerical matrix, with elements as rows.
labels	A vector of integers with length =nrow(x) indicating the cluster for each observation.
r	Number of cluster border representatives.
s	Vector of shrinking factors.
clusterstdev	Logical. If TRUE, the neighborhood radius for intra-cluster density is the within-cluster estimated squared distance from the mean of the cluster; otherwise it is the average of these over all clusters.
trace	Logical; whether to print processing info.

Value

A vector with the following values (see refs for details):

cdbw	value of CDbw index (the higher the better).
cohesion	cohesion.
compactness	compactness.
sep	separation.

Author(s)

Christian Hennig

References

Halkidi, M. and Vazirgiannis, M. (2008) A density-based cluster validity approach using multi-representatives. Pattern Recognition Letters 29, 773-786.

Halkidi, M., Vazirgiannis, M. and Hennig, C. (2015) Method-independent indices for cluster validation. In C. Hennig, M. Meila, F. Murtagh, R. Rocci (eds.) Handbook of Cluster Analysis, CRC Press/Taylor & Francis, Boca Raton.

Examples

```
d1 <- mockData()
CDbw(d1[,seq_len(2)], d1[,3])
```

CHAOS *Calculate CHAOS score*

Description

CHAOS score measures the clustering performance by calculating the mean length of the graph edges in the 1-nearest neighbor (1NN) graph for each cluster, averaged across clusters. Lower CHAOS score indicates better spatial domain clustering performance.

Usage

```
CHAOS(labels, location, BNPARAM = NULL)
```

Arguments

labels	Cluster labels.
location	A numeric data matrix containing location information, where rows are points and columns are location dimensions.
BNPARAM	BNPARAM object passed to findKNN specifying the kNN approximation method to use. Defaults to exact for small datasets, and Annoy for larger ones.

Value

A numeric value for CHAOS score.

Examples

```
data(sp_toys)
data <- sp_toys
CHAOS(data$label, data[,c("x", "y")])
CHAOS(data$p1, data[,c("x", "y")])
CHAOS(data$p2, data[,c("x", "y")])
```

dbcv *Calculate DBCV Metric*

Description

Compute the DBCV (Density-Based Clustering Validation) metric.

Usage

```
dbcv(
  X,
  labels,
  distance = "euclidean",
  noise_id = -1,
  check_duplicates = FALSE,
  use_igraph_mst = TRUE,
  BPPARAM = BiocParallel::SerialParam(),
  ...
)
```

Arguments

X	Numeric matrix of samples.
labels	Integer vector of cluster IDs.
distance	String specifying the distance metric. "sqeuclidean", or possible method in stats::dist(). By default "euclidean".
noise_id	Integer, the cluster ID in y for noise (default -1).
check_duplicates	Logical flag to check for duplicate samples.
use_igraph_mst	Logical flag to use igraph's MST implementation. Currently only mst from igraph is implemented.
BPPARAM	BiocParallel params for multithreading (default none)
...	Ignored

Details

This implementation will not fully reproduce the results of other existing implementations (e.g. https://github.com/FelSiq/DBCV) due to the different algorithms used for computing the Minimum Spanning Tree.

Value

A list:

vcs	Numeric vector of validity index for each cluster.
dbcv	Numeric value representing the overall DBCV metric.

References

Davoud Moulavi, et al. 2014; 10.1137/1.9781611973440.96.

Examples

```
data(noisy_moon)
data <- noisy_moon
dbcv(data[, c("x", "y")], data$kmeans_label)
dbcv(data[, c("x", "y")], data$hdbscan_label)
```

ELSA *Calculate ELSA scores*

Description

Calculating the Entropy-based Local indicator of Spatial Association (ELSA) scores, which consist of Ea, Ec and the overall ELSA.

Usage

```
ELSA(labels, location, k = 10)
```

Arguments

labels	Cluster labels.
location	A numerical matrix containing the location information, with rows as samples and columns as location dimensions.
k	Number of nearest neighbors.

Value

A dataframe containing the Ea, Ec and ELSA for all samples in the dataset.

References

Naimi, Babak, et al., 2019; 10.1016/j.spasta.2018.10.001

Examples

```
data(sp_toys)
data <- sp_toys
ELSA(data$label, data[,c("x", "y")], k=6)
ELSA(data$p1, data[,c("x", "y")], k=6)
ELSA(data$p2, data[,c("x", "y")], k=6)
```

emb2knn *Computes k nearest neighbors from embedding*

Description

Computes k nearest neighbors from embedding.

Usage

```
emb2knn(x, k, BNPARAM = NULL)
```

Arguments

x	A numeric matrix (with features as columns and items as rows) from which nearest neighbors will be computed.
k	The number of nearest neighbors.
BNPARAM	A BiocNeighbors parameter object to compute kNNs. Ignored unless the input is a matrix or data.frame. If omitted, the Annoy approximation will be used if there are more than 500 elements.

Value

A knn list.

Examples

```
d1 <- mockData()
emb2knn(as.matrix(d1[,seq_len(2)]),k=5)
```

emb2snn *Computes shared nearest neighbors from embedding*

Description

computes shared nearest neighbors from embedding.

Usage

```
emb2snn(x, k, type = "rank", BNPARAM = NULL)
```

Arguments

x	A numeric matrix (with features as columns and items as rows) from which nearest neighbors will be computed.
k	The number of nearest neighbors.
type	A string specifying the type of weighting scheme to use for shared neighbors. Possible choices include "rank", "number", and "jaccard". See type in bluster::neighborsToSNNGraph for details.
BNPARAM	A BiocNeighbors parameter object to compute kNNs. Ignored unless the input is a matrix or data.frame. If omitted, the Annoy approximation will be used if there are more than 500 elements.

Value

An igraph object.

Examples

```
d1 <- mockData()
emb2snn(as.matrix(d1[,seq_len(2)]),k=5)
```

---

findSpatialKNN *Find the k nearest spatial neighbors*

Description

For a given dataset, find the k nearest neighbors for each object based on their spatial locations, with the option of handling ties.

Usage

```
findSpatialKNN(
  location,
  k,
  keep_ties = TRUE,
  useMedianDist = FALSE,
  BNPARAM = NULL
)
```

Arguments

location	A numeric data matrix containing location information, where rows are points and columns are location dimensions.
k	The number of nearest neighbors to look at.
keep_ties	A Boolean indicating if ties are counted once or not. If TRUE, neighbors of the same distances will be included even if it means returning more than k neighbors.

useMedianDist	Use the median distance of the k nearest neighbor as maximum distance to be included. Ignored if keep_ties=FALSE.
BNPARAM	BNPARAM object passed to findKNN specifying the kNN approximation method to use. Defaults to exact for small datasets, and Annoy for larger ones.

Value

A list of indices.

Examples

```
data(sp_toys)
data <- sp_toys
findSpatialKNN(data[,c("x", "y")], k=6)
```

FMeasure *Calculate F measure*

Description

Compute the F measure between two clustering results. This is directly copied from the package FlowSOM.

Usage

```
FMeasure(true, pred, silent = TRUE)
```

Arguments

true	Array containing real cluster labels for each sample
pred	Array containing predicted cluster labels for each sample
silent	Logical, if FALSE, print some information about precision and recall

Value

F measure score

`fuzzyHardMetrics` *Compute fuzzy-hard versions of pair-sorting partition metrics*

Description

Computes fuzzy-hard versions of pair-sorting partition metrics to compare a hard clustering with both a fuzzy and hard truth. This was especially designed for cases where the fuzzy truth represents an uncertainty of a hard truth. Briefly put, the maximum of the pair concordance between the clustering and either the hard or the fuzzy truth is used, and the hard truth is used to compute completeness. See `fuzzyPartitionMetrics` for the more standard implementation of the metrics.

Usage

```
fuzzyHardMetrics(
  hardTrue,
  fuzzyTrue,
  hardPred,
  nperms = NULL,
  returnElementPairAccuracy = FALSE,
  lowMemory = NULL,
  verbose = TRUE,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

`hardTrue`	An atomic vector coercible to a factor or integer vector containing the true hard labels. Must have the same length as `hardPred`.
`fuzzyTrue`	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns). Must have the same number of rows as the length of `hardTrue`. Also note that the columns of `fuzzyTrue` should be in the order of the levels (or integer values) of `hardTrue`.
`hardPred`	An atomic vector coercible to a factor or integer vector containing the predicted hard labels.
`nperms`	The number of permutations (for correction for chance). If NULL (default), a first set of 10 permutations will be run to estimate whether the variation across permutations is above 0.0025, in which case more (max 1000) permutations will be run.
`returnElementPairAccuracy`	Logical. If TRUE, returns the per-element pair accuracy instead of the various parition-level and dataset-level metrics. Default FALSE.
`lowMemory`	Logical; whether to use the slower, low-memory algorithm. By default this is enabled if the projected memory usage is higher than ~2GB.
`verbose`	Logical; whether to print info and warnings, including the standard error of the mean across permutations (giving an idea of the precision of the adjusted metrics).
`BPPARAM`	BiocParallel params for multithreading (default none)

Value

A list of metrics:

NDC	Hullermeier's NDC (fuzzy rand index)
ACI	Ambrosio's Adjusted Concordance Index (ACI), i.e. a permutation-based fuzzy version of the adjusted Rand index.
fuzzyWH	Fuzzy Wallace Homogeneity index
fuzzyWC	Fuzzy Wallace Completeness index
fuzzyAWH	Adjusted fuzzy Wallace Homogeneity index
fuzzyAWC	Adjusted fuzzy Wallace Completeness index

Author(s)

Pierre-Luc Germain

References

Hullermeier et al. 2012; 10.1109/TFUZZ.2011.2179303;

D'Ambrosio et al. 2021; 10.1007/s00357-020-09367-0

See Also

FuzzyPartitionMetrics().

Examples

```
# generate a fuzzy truth:
fuzzyTrue <- matrix(c(
  0.95, 0.025, 0.025,
  0.98, 0.01, 0.01,
  0.96, 0.02, 0.02,
  0.95, 0.04, 0.01,
  0.95, 0.01, 0.04,
  0.99, 0.005, 0.005,
  0.025, 0.95, 0.025,
  0.97, 0.02, 0.01,
  0.025, 0.025, 0.95),
  ncol = 3, byrow=TRUE)
# a hard truth:
hardTrue <- apply(fuzzyTrue,1,FUN=which.max)
# some predicted labels:
hardPred <- c(1,1,1,1,1,1,2,2,2)
fuzzyHardMetrics(hardTrue, fuzzyTrue, hardPred, nperms=3)
```

`fuzzyHardMetrics2` *Compute fuzzy-hard metrics with lower memory requirement*

Description

This is a slightly slower, but low-memory version of `fuzzyHardMetrics`.

Usage

```
fuzzyHardMetrics2(
  hardTrue,
  fuzzyTrue,
  hardPred,
  nperms = 10,
  returnElementPairAccuracy = FALSE,
  verbose = TRUE,
  BPPARAM = BiocParallel::SerialParam()
)
```

Arguments

`hardTrue`	An atomic vector coercible to a factor or integer vector containing the true hard labels. Must have the same length as `hardPred`.
`fuzzyTrue`	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns). Must have the same number of rows as the length of `hardTrue`. Also note that the columns of `fuzzyTrue` should be in the order of the levels (or integer values) of `hardTrue`.
`hardPred`	An atomic vector coercible to a factor or integer vector containing the predicted hard labels.
`nperms`	The number of permutations (for correction for chance). If NULL (default), a first set of 10 permutations will be run to estimate whether the variation across permutations is above 0.0025, in which case more (max 1000) permutations will be run.
`returnElementPairAccuracy`	Logical. If TRUE, returns the per-element pair accuracy instead of the various parition-level and dataset-level metrics. Default FALSE.
`verbose`	Logical; whether to print info and warnings, including the standard error of the mean across permutations (giving an idea of the precision of the adjusted metrics).
`BPPARAM`	BiocParallel params for multithreading (default none)

Value

A list of metrics:

`NDC`	Hullermeier's NDC (fuzzy rand index)

ACI	Ambrosio's Adjusted Concordance Index (ACI), i.e. a permutation-based fuzzy version of the adjusted Rand index.
fuzzyWH	Fuzzy Wallace Homogeneity index
fuzzyWC	Fuzzy Wallace Completeness index
fuzzyAWH	Adjusted fuzzy Wallace Homogeneity index
fuzzyAWC	Adjusted fuzzy Wallace Completeness index

Author(s)

Pierre-Luc Germain

References

Hullermeier et al. 2012; 10.1109/TFUZZ.2011.2179303;

D'Ambrosio et al. 2021; 10.1007/s00357-020-09367-0

See Also

fuzzyHardMetrics()

Examples

```
# generate a fuzzy truth:
fuzzyTrue <- matrix(c(
0.95, 0.025, 0.025,
0.98, 0.01, 0.01,
0.96, 0.02, 0.02,
0.95, 0.04, 0.01,
0.95, 0.01, 0.04,
0.99, 0.005, 0.005,
0.025, 0.95, 0.025,
0.97, 0.02, 0.01,
0.025, 0.025, 0.95),
ncol = 3, byrow=TRUE)
# a hard truth:
hardTrue <- apply(fuzzyTrue,1,FUN=which.max)
# some predicted labels:
hardPred <- c(1,1,1,1,1,1,2,2,2)
poem:::fuzzyHardMetrics2(hardTrue, fuzzyTrue, hardPred, nperms=3)
```

---

fuzzyHardSpotConcordance

Per-element maximal concordance between a hard and a fuzzy partition

---

Description

Per-element maximal concordance between a hard clustering and hard and fuzzy ground truth labels.

Usage

```
fuzzyHardSpotConcordance(
  hardTrue,
  fuzzyTrue,
  hardPred,
  useNegatives = TRUE,
  verbose = TRUE
)
```

Arguments

hardTrue	A vector of true cluster labels
fuzzyTrue	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns). Must have the same number of rows as the length of hardTrue.
hardPred	A vector of predicted cluster labels
useNegatives	Logical; whether to include negative pairs in the concordance score (tends to result in a larger overall concordance and lower dynamic range of the score). Default TRUE.
verbose	Logical; whether to print expected memory usage for large datasets.

Value

A numeric vector of concordance scores for each element of hardPred

Examples

```
# generate a fuzzy truth:
fuzzyTrue <- matrix(c(
  0.95, 0.025, 0.025,
  0.98, 0.01, 0.01,
  0.96, 0.02, 0.02,
  0.95, 0.04, 0.01,
  0.95, 0.01, 0.04,
  0.99, 0.005, 0.005,
  0.025, 0.95, 0.025,
  0.97, 0.02, 0.01,
  0.025, 0.025, 0.95),
  ncol = 3, byrow=TRUE)
# a hard truth:
hardTrue <- apply(fuzzyTrue,1,FUN=which.max)
# some predicted labels:
hardPred <- c(1,1,1,1,1,2,2,2)
fuzzyHardSpotConcordance(hardTrue, fuzzyTrue, hardPred)
```

`fuzzyPartitionMetrics` *Compute fuzzy-fuzzy versions of pair-sorting partition metrics*

Description

Computes fuzzy versions of pair-sorting partition metrics. This is largely based on the permutation-based implementation by Antonio D'Ambrosio from the `ConsRankClass` package, modified to also compute the fuzzy versions of the adjusted Wallace indices, implement multithreading, and adjust the number of permutations according to their variability.

Usage

```
fuzzyPartitionMetrics(
  P,
  Q,
  computeWallace = TRUE,
  nperms = NULL,
  verbose = TRUE,
  returnElementPairAccuracy = FALSE,
  BPPARAM = BiocParallel::SerialParam(),
  tnorm = c("product", "min", "lukasiewicz")
)
```

Arguments

`P`	A object coercible to a numeric matrix with membership probability of elements (rows) in ground-truth classes (columns).
`Q`	A object coercible to a numeric matrix with membership probability of elements (rows) in predicted clusters (columns). Must have the same number of rows as `P`.
`computeWallace`	Logical; whether to compute the individual fuzzy versions of the Wallace indices (increases running time).
`nperms`	The number of permutations (for correction for chance). If NULL (default), a first set of 10 permutations will be run to estimate whether the variation across permutations is above 0.0025, in which case more (max 1000) permutations will be run.
`verbose`	Logical; whether to print info and warnings, including the standard error of the mean across permutations (giving an idea of the precision of the adjusted metrics).
`returnElementPairAccuracy`	Logical. If TRUE, returns the per-element pair accuracy instead of the various parition-level and dataset-level metrics. Default FALSE.
`BPPARAM`	BiocParallel params for multithreading (default none)
`tnorm`	Which type of t-norm operation to use for class membership of pairs (either product, min, or lukasiewicz) when calculating the Wallace indices. Does not influence the NDC/ACI metrics.

Value

When `returnElementPairAccuracy` is `FALSE`, return a list of metrics:

`NDC`	Hullermeier's NDC (fuzzy rand index)
`ACI`	Ambrosio's Adjusted Concordance Index (ACI), i.e. a permutation-based fuzzy version of the adjusted Rand index.
`fuzzyWH`	Fuzzy Wallace Homogeneity index
`fuzzyWC`	Fuzzy Wallace Completeness index
`fuzzyAWH`	Adjusted fuzzy Wallace Homogeneity index
`fuzzyAWC`	Adjusted fuzzy Wallace Completeness index

Author(s)

Pierre-Luc Germain

References

Hullermeier et al. 2012; 10.1109/TFUZZ.2011.2179303;

D'Ambrosio et al. 2021; 10.1007/s00357-020-09367-0

Examples

```
# generate fuzzy partitions:
```

`fuzzySpotConcordance` *Per-element concordance between two fuzzy partitions*

Description

Per-element concordance between two fuzzy partitions

Usage

```
fuzzySpotConcordance(P, Q)
```

Arguments

`P`	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns)
`Q`	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns). Must have the same number of rows as `P`

Value

A numeric vector of concordance scores for each row of `P`.

Examples

```
# generate fuzzy partitions:
m1 <- matrix(c(0.95, 0.025, 0.025,
               0.98, 0.01, 0.01,
               0.96, 0.02, 0.02,
               0.95, 0.04, 0.01,
               0.95, 0.01, 0.04,
               0.99, 0.005, 0.005,
               0.025, 0.95, 0.025,
               0.97, 0.02, 0.01,
               0.025, 0.025, 0.95),
               ncol = 3, byrow=TRUE)
m2 <- matrix(c(0.95, 0.025, 0.025,
               0.98, 0.01, 0.01,
               0.96, 0.02, 0.02,
               0.025, 0.95, 0.025,
               0.02, 0.96, 0.02,
               0.01, 0.98, 0.01,
               0.05, 0.05, 0.95,
               0.02, 0.02, 0.96,
               0.01, 0.01, 0.98),
               ncol = 3, byrow=TRUE)
colnames(m1) <- colnames(m2) <- LETTERS[seq_len(3)]
fuzzySpotConcordance(m1,m2)
```

`getEmbeddingClassMetrics` *getEmbeddingClassMetrics*

Description

Computes class-level, embedding-based metrics.

Usage

```
getEmbeddingClassMetrics(
  x,
  labels,
  metrics = c("meanSW", "minSW", "pnSW", "dbcv"),
  distance = "euclidean",
  ...
)
```

Arguments

`x`	A data.frame or matrix (with features as columns and items as rows) from which the metrics will be computed.
`labels`	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.

metrics	The metrics to compute.
distance	The distance metric to use (default euclidean).
...	Optional arguments. See details.

Value

A data.frame of metrics for each node/element of x.

---

getEmbeddingElementMetrics
getEmbeddingElementMetrics

Description

Computes element-level, embedding-based metrics.

Usage

```
getEmbeddingElementMetrics(
  x,
  labels,
  metrics = c("SW"),
  distance = "euclidean",
  ...
)
```

Arguments

x	A data.frame or matrix (with features as columns and items as rows) from which the metrics will be computed.
labels	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
metrics	The metrics to compute. Currently, only the silhouette width is supported at the node-level.
distance	The distance metric to use (default euclidean).
...	Optional arguments. See details.

Value

A data.frame of metrics for each node/element of x.

`getEmbeddingGlobalMetrics` *getEmbeddingGlobalMetrics*

Description

Computes dataset-level, embedding-based metrics.

Usage

```
getEmbeddingGlobalMetrics(
  x,
  labels,
  metrics = c("meanSW", "meanClassSW", "pnSW", "minClassSW", "cdbw", "cohesion",
    "compactness", "sep", "dbcv"),
  distance = "euclidean",
  ...
)
```

Arguments

`x`	A data.frame or matrix (with features as columns and items as rows) from which the metrics will be computed.
`labels`	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`metrics`	The metrics to compute.
`distance`	The distance metric to use (default euclidean).
`...`	Optional arguments. See details.

Value

A data.frame (with 1 row) of metrics.

`getEmbeddingMetrics` *Compute embedding-based metrics*

Description

Computes embedding-based metrics for the specified level.

Usage

```
getEmbeddingMetrics(
  x,
  labels,
  metrics = NULL,
  distance = "euclidean",
  level = "class",
  ...
)
```

Arguments

x	A data.frame or matrix (with features as columns and items as rows) from which the metrics will be computed.
labels	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
metrics	The metrics to compute. See details.
distance	The distance metric to use (default euclidean).
level	The level to calculate the metrics. Options include "element", "class" and "dataset".
...	Optional arguments. See details.

Details

The allowed values for metrics depend on the value of level:

- If level = "element", the allowed metrics are: "SW".
- If level = "class", the allowed metrics are: "meanSW", "minSW", "pnSW", "dbcv".
- If level = "dataset", the allowed metrics are: "meanSW", "meanClassSW", "pnSW", "minClassSW", "cdbw", "cohesion", "compactness", "sep", "dbcv".

The function(s) that the optional arguments ... passed to depend on the value of level:

- If level = "element", optional arguments are passed to stats::dist().
- If level = "class", optional arguments are passed to dbcv().
- If level = "dataset", optional arguments are passed to dbcv() or CDbw().

Value

A data.frame of metrics.

Examples

```
d1 <- mockData()
getEmbeddingMetrics(d1[,seq_len(2)], labels=d1$class,
metrics=c("meanSW", "minSW", "pnSW", "dbcv"), level="class")
```

getFuzzyLabel *Get fuzzy representation of labels*

Description

Get fuzzy representation of labels according to the spatial neighborhood label composition.

Usage

```
getFuzzyLabel(labels, location, k = 6, alpha = 0.5, ...)
```

Arguments

labels	An anomic vector of cluster labels
location	A matrix or data.frame of coordinates
k	The wished number of nearest neighbors
alpha	The parameter to control to what extend the spot itself contribute to the class composition calculation. "equal" means it is weighted the same as other neighbors. A numeric value between 0 and 1 means the weight of the frequency contribution for the spot itself, and the frequency contribution for its knn is then 1-alpha. By default 0.5.
...	Passed to findSpatialKNN().

Value

A matrix of fuzzy memberships.

Examples

```
data(sp_toys)
data <- sp_toys
getFuzzyLabel(data$label, data[,c("x", "y")], k=6)
```

getFuzzyPartitionElementMetrics *getFuzzyPartitionElementMetrics*

Description

Computes a selection of external fuzzy clustering evaluation metrics at the element level.

Usage

```
getFuzzyPartitionElementMetrics(
  hardTrue = NULL,
  fuzzyTrue = NULL,
  hardPred = NULL,
  fuzzyPred = NULL,
  fuzzy_true = TRUE,
  fuzzy_pred = FALSE,
  metrics = c("fuzzySPC"),
  useNegatives = TRUE,
  verbose = TRUE,
  usePairs = TRUE
)
```

Arguments

hardTrue	A vector of true cluster labels
fuzzyTrue	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns). Must have the same number of rows as the length of hardTrue.
hardPred	A vector of predicted cluster labels
fuzzyPred	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns).
fuzzy_true	Logical; whether the truth is fuzzy.
fuzzy_pred	Logical; whether the prediction is fuzzy.
metrics	The metrics to compute. Currently only "fuzzySPC" is included at the element level.
useNegatives	Logical; whether to include negative pairs in the concordance score (tends to result in a larger overall concordance and lower dynamic range of the score). Default TRUE.
verbose	Logical; whether to print expected memory usage for large datasets.
usePairs	Logical; whether to compute over pairs instead of elements. Only useful when fuzzy_true=TRUE and fuzzy_pred=FALSE.

Value

A dataframe of metric values.

getFuzzyPartitionMetrics *Compute external metrics for fuzzy clusterings*

Description

Computes a selection of external fuzzy clustering evaluation metrics.

Usage

```
getFuzzyPartitionMetrics(
  hardTrue = NULL,
  fuzzyTrue = NULL,
  hardPred = NULL,
  fuzzyPred = NULL,
  metrics = c("fuzzyWH", "fuzzyAWH", "fuzzyWC", "fuzzyAWC"),
  level = "class",
  nperms = NULL,
  verbose = TRUE,
  returnElementPairAccuracy = FALSE,
  BPPARAM = BiocParallel::SerialParam(),
  useNegatives = TRUE,
  usePairs = NULL,
  ...
)
```

Arguments

hardTrue	An atomic vector coercible to a factor or integer vector containing the true hard labels.
fuzzyTrue	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns).
hardPred	An atomic vector coercible to a factor or integer vector containing the predicted hard labels.
fuzzyPred	A object coercible to a numeric matrix with membership probability of elements (rows) in clusters (columns).
metrics	The metrics to compute. See details.
level	The level to calculate the metrics. Options include "element", "class" and "dataset".
nperms	The number of permutations (for correction for chance). If NULL (default), a first set of 10 permutations will be run to estimate whether the variation across permutations is above 0.0025, in which case more (max 1000) permutations will be run.
verbose	Logical; whether to print info and warnings, including the standard error of the mean across permutations (giving an idea of the precision of the adjusted metrics).

returnElementPairAccuracy
: Logical. If TRUE, returns the per-element pair accuracy instead of the various parition-level and dataset-level metrics. Default FALSE.

BPPARAM
: BiocParallel params for multithreading (default none)

useNegatives
: Logical; whether to include negative pairs in the concordance score (tends to result in a larger overall concordance and lower dynamic range of the score). Default TRUE.

usePairs
: Logical; whether to compute over pairs instead of elements Recommended and TRUE by default.

...
: Optional arguments for fuzzyPartitionMetrics: tnorm. Only useful when fuzzy_true=TRUE and fuzzy_pred=TRUE.

Details

The allowed values for metrics depend on the value of level:

- If level = "element", the allowed metrics are: "fuzzySPC".
- If level = "class", the allowed metrics are: "fuzzyWH", "fuzzyAWH", "fuzzyWC", "fuzzyAWC".
- If level = "dataset", the allowed metrics are: "fuzzyRI", "fuzzyARI", "fuzzyWH", "fuzzyAWH", "fuzzyWC", "fuzzyAWC".

Value

A dataframe of metric results.

Examples

```
# generate fuzzy partitions:
m1 <- matrix(c(0.95, 0.025, 0.025,
               0.98, 0.01, 0.01,
               0.96, 0.02, 0.02,
               0.95, 0.04, 0.01,
               0.95, 0.01, 0.04,
               0.99, 0.005, 0.005,
               0.025, 0.95, 0.025,
               0.97, 0.02, 0.01,
               0.025, 0.025, 0.95),
               ncol = 3, byrow=TRUE)
m2 <- matrix(c(0.95, 0.025, 0.025,
               0.98, 0.01, 0.01,
               0.96, 0.02, 0.02,
               0.025, 0.95, 0.025,
               0.02, 0.96, 0.02,
               0.01, 0.98, 0.01,
               0.05, 0.05, 0.95,
               0.02, 0.02, 0.96,
               0.01, 0.01, 0.98),
               ncol = 3, byrow=TRUE)
colnames(m1) <- colnames(m2) <- LETTERS[seq_len(3)]
getFuzzyPartitionMetrics(fuzzyTrue=m1,fuzzyPred=m2, level="class")
```

```
# generate a fuzzy truth:
fuzzyTrue <- matrix(c(
  0.95, 0.025, 0.025,
  0.98, 0.01, 0.01,
  0.96, 0.02, 0.02,
  0.95, 0.04, 0.01,
  0.95, 0.01, 0.04,
  0.99, 0.005, 0.005,
  0.025, 0.95, 0.025,
  0.97, 0.02, 0.01,
  0.025, 0.025, 0.95),
  ncol = 3, byrow=TRUE)
# a hard truth:
hardTrue <- apply(fuzzyTrue,1,FUN=which.max)
# some predicted labels:
hardPred <- c(1,1,1,1,1,1,2,2,2)
getFuzzyPartitionMetrics(hardPred=hardPred, hardTrue=hardTrue,
fuzzyTrue=fuzzyTrue, nperms=3, level="class")
```

`getGraphClassMetrics` *getGraphClassMetrics*

Description

Computes a selection of supervised graph evaluation metrics using ground truth class labels. The metrics are reported (as average) per class.

Usage

```
getGraphClassMetrics(
  x,
  labels,
  metrics = c("SI", "NP", "AMSP", "PWC", "NCE"),
  directed = NULL,
  ...
)

## S4 method for signature 'list'
getGraphClassMetrics(x, labels, metrics, directed = NULL, k = NULL, ...)

## S4 method for signature 'data.frame'
getGraphClassMetrics(
  x,
  labels,
  metrics,
  directed = NULL,
  k,
```

```
  shared = FALSE,
  ...
)

## S4 method for signature 'matrix'
getGraphClassMetrics(
  x,
  labels,
  metrics,
  directed = NULL,
  k,
  shared = FALSE,
  ...
)

## S4 method for signature 'igraph'
getGraphClassMetrics(
  x,
  labels,
  metrics = c("SI", "NP", "AMSP", "PWC", "NCE"),
  directed = NULL,
  ...
)

## S4 method for signature 'dist'
getGraphClassMetrics(
  x,
  labels,
  metrics = c("SI", "NP", "AMSP", "PWC", "NCE"),
  directed = NULL,
  ...
)
```

Arguments

x	Either an igraph object, a list of nearest neighbors (see details below), or a data.frame or matrix (with features as columns and items as rows) from which nearest neighbors will be computed.
labels	Either a factor or a character vector indicating the true class label of each element (i.e. row or vertex) of x.
metrics	The metrics to compute. See details.
directed	Logical; whether to compute the metrics in a directed fashion. If left to NULL, conventional choices will be made per metric (adhesion, cohesion, PWC AMSP undirected, others directed).
...	Optional arguments for emb2knn() or emb2snn().
k	The number of nearest neighbors to compute and/or use. Can be omitted if x is a graph or list of nearest neighbors.

shared — Logical; whether to use a shared nearest neighbor network instead of a nearest neighbor network. Ignored if x is not an embedding or dist object.

Value

A data.frame of metrics for each class.

`getGraphElementMetrics` *getGraphElementMetrics*

Description

Computes a selection of supervised graph evaluation metrics using ground truth class labels. The metrics are reported (as average) per node/element.

Usage

```
getGraphElementMetrics(x, labels, directed = NULL, ...)

## S4 method for signature 'list'
getGraphElementMetrics(x, labels, metrics, directed = NULL, k = NULL, ...)

## S4 method for signature 'data.frame'
getGraphElementMetrics(
  x,
  labels,
  metrics,
  directed = NULL,
  k,
  shared = FALSE,
  ...
)

## S4 method for signature 'matrix'
getGraphElementMetrics(
  x,
  labels,
  metrics,
  directed = NULL,
  k,
  shared = FALSE,
  ...
)

## S4 method for signature 'igraph'
getGraphElementMetrics(x, labels, directed = NULL, ...)
```

```
## S4 method for signature 'dist'
getGraphElementMetrics(x, labels, directed = NULL, ...)
```

Arguments

x	Either an igraph object, a list of nearest neighbors (see details below), or a data.frame or matrix (with features as columns and items as rows) from which nearest neighbors will be computed.
labels	Either a factor or a character vector indicating the true class label of each element (i.e. row or vertex) of x.
directed	Logical; whether to compute the metrics in a directed fashion. If left to NULL, conventional choices will be made per metric (adhesion, cohesion, PWC AMSP undirected, others directed).
...	Optional arguments for emb2knn() or emb2snn().
metrics	The metrics to compute. See details.
k	The number of nearest neighbors to compute and/or use. Can be omitted if x is a graph or list of nearest neighbors.
shared	Logical; whether to use a shared nearest neighbor network instead of a nearest neighbor network. Ignored if x is not an embedding or dist object.

Value

A data.frame of metrics for each node/element of x.

---

getGraphMetrics *Compute graph-based metrics*

Description

Computes a selection of graph evaluation metrics using class labels.

Usage

```
getGraphMetrics(
  x,
  labels,
  metrics = NULL,
  directed = NULL,
  k = 10,
  shared = FALSE,
  level = "class",
  ...
)
```

Arguments

x	Either an igraph object, a list of nearest neighbors (see details below), or a data.frame or matrix (with features as columns and items as rows) from which nearest neighbors will be computed.
labels	Either a factor or a character vector indicating the true class label of each element (i.e. row or vertex) of x.
metrics	The metrics to compute. See details.
directed	Logical; whether to compute the metrics in a directed fashion. If left to NULL, conventional choices will be made per metric (adhesion, cohesion, PWC AMSP undirected, others directed).
k	The number of nearest neighbors to compute and/or use. Can be omitted if x is a graph or list of nearest neighbors.
shared	Logical; whether to use a shared nearest neighbor network instead of a nearest neighbor network. Ignored if x is not an embedding or dist object.
level	The level to calculate the metrics. Options include "element", "class" and "dataset".
...	Optional arguments for emb2knn() or emb2snn().

Details

The allowed values for metrics depend on the value of level:

- If level = "element", the allowed metrics are: "SI","ISI","NP", "NCE" (see below for details).
- If level = "class", the allowed metrics are:
  - "SI": Simpson's Index.
  - "ISI": Inverse Simpson's Index
  - "NP": Neighborhood Purity
  - "AMSP": Adjusted Mean Shortest Path
  - "PWC": Proportion of Weakly Connected
  - "NCE": Neighborhood Class Enrichment
  - "adhesion": adhesion of a graph, is the minumum number of nodes that must be removed to split a graph.
  - "cohesion": cohesion of a graph, is the minumum number of edges that must be removed to split a graph.
- If level = "dataset", the allowed metrics are: "SI","ISI", "NP","AMSP","PWC","NCE", "adhesion","cohesion".

Value

A data.frame of metrics.

Examples

```
d1 <- mockData()
getGraphMetrics(d1[,seq_len(2)], labels=d1$class, level="class")
```

getNeighboringPairConcordance *Per-element local concordance between a clustering and a ground truth*

Description

Per-element local concordance between a clustering and a ground truth

Usage

```
getNeighboringPairConcordance(
  true,
  pred,
  location,
  k = 20L,
  useNegatives = FALSE,
  distWeights = TRUE,
  BNPARAM = NULL
)
```

Arguments

true	A vector of true class labels
pred	A vector of predicted clusters
location	A matrix or data.frame with spatial dimensions as columns. Alternatively, a nearest neighbor object as produced by findKNN.
k	Approximate number of nearest neighbors to consider
useNegatives	Logical; whether to include the concordance of negative pairs in the score (default FALSE).
distWeights	Logical; whether to weight concordance by distance (default TRUE).
BNPARAM	A BiocNeighbors parameter object to compute kNNs. Ignored unless the input is a matrix or data.frame. If omitted, the Annoy approximation will be used if there are more than 500 elements.

Value

A vector of concordance scores

Examples

```
data(sp_toys)
data <- sp_toys
getNeighboringPairConcordance(data$label, data$p1, data[,c("x", "y")], k=6)
```

`getPairConcordance` *Per-element pair concordance score*

Description

Per-element pair concordance between a clustering and a ground truth. Note that by default, negative pairs (i.e. that are split in both the predicted and true groupings) are not counted. To count it (as in the standard Rand Index), use `useNegatives=TRUE`.

Usage

```
getPairConcordance(
  true,
  pred,
  usePairs = TRUE,
  useNegatives = FALSE,
  adjust = FALSE
)
```

Arguments

`true`	A vector of true class labels
`pred`	A vector of predicted clusters
`usePairs`	Logical; whether to compute over pairs instead of elements Recommended and TRUE by default.
`useNegatives`	Logical; whether to include the consistency of negative pairs in the score (default FALSE).
`adjust`	Logical; whether to adjust for chance. Only implemented for `useNegatives=FALSE` (doesn't make sense on a element-level otherwise).

Value

A vector of concordance scores

`getPartitionClassMetrics`
getPartitionClassMetrics

Description

Computes a selection of external evaluation metrics for partition. The metrics are reported per class.

Usage

```
getPartitionClassMetrics(
  true,
  pred,
  metrics = c("WC", "WH", "AWC", "AWH", "FM")
)
```

Arguments

true	A vector containing the labels of the true classes. Must be a vector of characters, integers, numerics, or a factor, but not a list.
pred	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
metrics	The metrics to compute. If omitted, main metrics will be computed.

Value

A dataframe of metrics.

---

getPartitionElementMetrics

getPartitionElementMetrics

Description

Computes a selection of external evaluation metrics for partition. The metrics are reported per element.

Usage

```
getPartitionElementMetrics(
  true,
  pred,
  metrics = c("SPC"),
  usePairs = TRUE,
  useNegatives = TRUE
)
```

Arguments

true	A vector of true class labels
pred	A vector of predicted clusters
metrics	The metrics to compute.
usePairs	Logical; whether to compute over pairs instead of elements Recommended and TRUE by default.
useNegatives	Logical; whether to include the consistency of negative pairs in the score (default FALSE).

Value

A dataframe of metrics.

`getPartitionGlobalMetrics` *getPartitionGlobalMetrics*

Description

Computes a selection of external evaluation metrics for partition. The metrics are reported per dataset.

Usage

```
getPartitionGlobalMetrics(
  true,
  pred,
 metrics = c("RI", "WC", "WH", "ARI", "NCR", "AWC", "AWH", "MI", "AMI", "VI", "EH",
    "EC", "VM", "FM"),
  ...
)
```

Arguments

`true`	A vector containing the labels of the true classes. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`pred`	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`metrics`	The metrics to compute. If omitted, main metrics will be computed. See below for more details.
`...`	Optional arguments for MI, VI, or VM. See `clevr::mutual_info()`, `clevr::variation_info()` and `clevr::v_measure()` for more details.

Value

A dataframe of metric results. Possible metrics are:

`RI`	Rand Index
`WC`	Wallace Completeness
`WH`	Wallace Homogeneity
`ARI`	Adjusted Rand Index
`AWC`	Adjusted Wallace Completeness
`AWH`	Adjusted Wallace Homogeneity
`NCR`	Normalized class size Rand index

MI	Mutual Information
AMI	Adjusted Mutual Information
VI	Variation of Information
EH	(Entropy-based) Homogeneity
EC	(Entropy-based) Completeness
VM	V-measure
FM	F-measure/weighted average F1 score
VDM	Van Dongen Measure
MHM	Meila-Heckerman Measure
MMM	Maximum-Match Measure
Mirkin	Mirkin Metric
Accuracy	Set Matching Accuracy

`getPartitionMetrics` *Compute partition-based metrics*

Description

Computes a selection of external evaluation metrics for partition.

Usage

```
getPartitionMetrics(true, pred, metrics = NULL, level = "class", ...)
```

Arguments

true	A vector containing the labels of the true classes. Must be a vector of characters, integers, numerics, or a factor, but not a list.
pred	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
metrics	The metrics to compute. If omitted, main metrics will be computed. See details.
level	The level to calculate the metrics. Options include "element", "class" and "dataset".
...	Optional arguments for MI, VI, or VM. See `clevr::mutual_info()`, `clevr::variation_info()` and `clevr::v_measure()` for more details.

Details

The allowed values for `metrics` depend on the value of `level`:

- If `level` = `"element"`, the allowed `metrics` are:
  - `"SPC"`: Spot-wise Pair Concordance.
  - `"ASPC"`: Adjusted Spot-wise Pair Concordance.
- If `level` = `"class"`, the allowed `metrics` are: `"WC"`,`"WH"`,`"AWC"`, `"AWH"`,`"FM"` (see below for details).
- If `level` = `"dataset"`, the allowed `metrics` are:
  - `"RI"`: Rand Index
  - `"WC"`: Wallace Completeness
  - `"WH"`: Wallace Homogeneity
  - `"ARI"`: Adjusted Rand Index
  - `"AWC"`: Adjusted Wallace Completeness
  - `"AWH"`: Adjusted Wallace Homogeneity
  - `"NCR"`: Normalized class size Rand index
  - `"MI"`: Mutual Information
  - `"AMI"`: Adjusted Mutual Information
  - `"VI"`: Variation of Information
  - `"EH"`: (Entropy-based) Homogeneity
  - `"EC"`: (Entropy-based) Completeness
  - `"VM"`: V-measure
  - `"FM"`: F-measure/weighted average F1 score
  - `"VDM"`: Van Dongen Measure
  - `"MHM"`: Meila-Heckerman Measure
  - `"MMM"`: Maximum-Match Measure
  - `"Mirkin"`: Mirkin Metric
  - `"Accuracy"`: Set Matching Accuracy

Value

A data.frame of metrics.

Examples

```
true <- rep(LETTERS[seq_len(3)], each=10)
pred <- c(rep("A", 8), rep("B", 9), rep("C", 3), rep("D", 10))
getPartitionMetrics(true, pred, level="class")
getPartitionMetrics(true, pred, level="dataset")
```

`getSpatialClassExternalMetrics` *Compute class-level external evaluation metrics for spatially-resolved data*

Description

Computes a selection of external clustering evaluation metrics for spatial data at the class/cluster level.

Usage

```
getSpatialClassExternalMetrics(
  true,
  pred,
  location,
  k = 6,
  alpha = 0.5,
  metrics = c("nsWH", "nsAWH", "nsWC", "nsAWC"),
  fuzzy_true = TRUE,
  fuzzy_pred = FALSE,
  ...
)
```

Arguments

`true`	A vector containing the labels of the true classes. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`pred`	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`location`	A matrix or data.frame of coordinates
`k`	The number of neighbors used when calculating the fuzzy class memberships for fuzzy metrics.
`alpha`	The parameter to control to what extend the spot itself contribute to the class composition calculation. `"equal"` means it is weighted the same as other neighbors. A numeric value between `0` and `1` means the weight of the frequency contribution for the spot itself, and the frequency contribution for its knn is then `1-alpha`. By default `0.5`.
`metrics`	a vector of metric names to compute.
`fuzzy_true`	Logical; whether to compute fuzzy class memberships for `true`.
`fuzzy_pred`	Logical; whether to compute fuzzy class memberships for `pred`.
`...`	Optional params for `fuzzyPartitionMetrics` or `findSpatialKNN`.

Value

A data.frame of metrics.

`getSpatialClassInternalMetrics` *Compute class-level internal evaluation metrics for spatially-resolved data*

Description

Computes a selection of internal clustering evaluation metrics for spatial data for each class.

Usage

```
getSpatialClassInternalMetrics(
  labels,
  location,
  k = 6,
  metrics = c("CHAOS", "PAS", "ELSA"),
  ...
)
```

Arguments

`labels`	A vector containing the labels to be evaluated.
`location`	A numerical matrix containing the location information, with rows as samples and columns as location dimensions.
`k`	The size of the spatial neighborhood to look at for each spot. This is used for calculating PAS and ELSA scores.
`metrics`	Possible metrics: "CHAOS", "PAS" and "ELSA".
`...`	Optional params for `PAS()`.

Value

A dataframe of metric values.

`getSpatialElementExternalMetrics` *getSpatialElementExternalMetrics*

Description

Computes a selection of external clustering evaluation metrics for spatial data at the element level.

Usage

```
getSpatialElementExternalMetrics(
  true,
  pred,
  location,
  k = 6,
  alpha = 0.5,
  metrics = c("nsSPC", "NPC", "SpatialSPC"),
  fuzzy_true = TRUE,
  fuzzy_pred = FALSE,
  ...
)
```

Arguments

`true`	A vector containing the labels of the true classes. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`pred`	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`location`	A matrix or data.frame of coordinates
`k`	The number of neighbors used when calculating the fuzzy class memberships for fuzzy metrics, or when calculating the weighted accuracy.
`alpha`	The parameter to control to what extend the spot itself contribute to the class composition calculation. `"equal"` means it is weighted the same as other neighbors. A numeric value between `0` and `1` means the weight of the frequency contribution for the spot itself, and the frequency contribution for its knn is then `1-alpha`. By default `0.5`.
`metrics`	a vector of metric names to compute.
`fuzzy_true`	Logical; whether to compute fuzzy class memberships for `true`.
`fuzzy_pred`	Logical; whether to compute fuzzy class memberships for `pred`.
`...`	Optional params for `getFuzzyPartitionElementMetrics()` or `findSpatialKNN()`.

Value

A data.frame of metrics.

`getSpatialElementInternalMetrics` *Compute spot-level internal evaluation metrics for spatially-resolved data*

Description

Computes a selection of internal clustering evaluation metrics for spatial data at each spot level.

Usage

```
getSpatialElementInternalMetrics(
  labels,
  location,
  k = 6,
  metrics = c("PAS", "ELSA"),
  ...
)
```

Arguments

labels	A vector containing the labels to be evaluated.
location	A numerical matrix containing the location information, with rows as samples and columns as location dimensions.
k	The size of the spatial neighborhood to look at for each spot. This is used for calculating PAS and ELSA scores.
metrics	Possible metrics: "PAS" and "ELSA".
...	Optional params for PAS().

Value

A dataframe containing the metric values for all samples in the dataset. If PAS is calculated, the value is a Boolean about the abnormality of a spot. If ELSA is calculated, Ea, Ec and ELSA for all spots will be returned.

---

getSpatialExternalMetrics

Calculate Spatial External Metrics

Description

A generic function to calculate spatial external metrics. It can be applied to raw components (true, pred, location) or directly to a SpatialExperiment object.

Usage

```
getSpatialExternalMetrics(
  object = NULL,
  true,
  pred,
  location = NULL,
  k = 6,
  alpha = 0.5,
  level = "class",
  metrics = NULL,
```

```
  fuzzy_true = TRUE,
  fuzzy_pred = FALSE,
  ...
)

## S4 method for signature 'missing'
getSpatialExternalMetrics(
  object = NULL,
  true,
  pred,
  location = NULL,
  k = 6,
  alpha = 0.5,
  level = "class",
  metrics = NULL,
  fuzzy_true = TRUE,
  fuzzy_pred = FALSE,
  ...
)

## S4 method for signature 'SpatialExperiment'
getSpatialExternalMetrics(
  object = NULL,
  true,
  pred,
  location = NULL,
  k = 6,
  alpha = 0.5,
  level = "class",
  metrics = NULL,
  fuzzy_true = TRUE,
  fuzzy_pred = FALSE,
  ...
)
```

Arguments

`object`	The main input. Can be a `SpatialExperiment` object or missing (when using `true`, `pred`, and `location` directly).
`true`	When `object` is missing: a vector containing the labels of the true classes. Must be a vector of characters, integers, numerics, or a factor, but not a list. When `object` is a `SpatialExperiment` object: the column name in `colData(object)` containing the true labels.
`pred`	When `object` is missing: a vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list. When `object` is a `SpatialExperiment` object: the column name in `colData(object)` containing the predicted labels.
`location`	A matrix or data.frame of coordinates

k	The number of neighbors used when calculating the fuzzy class memberships for fuzzy metrics, or when calculating the weighted accuracy.
alpha	The parameter to control to what extend the spot itself contribute to the class composition calculation. "equal" means it is weighted the same as other neighbors. A numeric value between 0 and 1 means the weight of the frequency contribution for the spot itself, and the frequency contribution for its knn is then 1-alpha. By default 0.5.
level	The level to calculate the metrics. Options include "element", "class" and "dataset".
metrics	The metrics to compute. See details.
fuzzy_true	Logical; whether to compute fuzzy class memberships for true.
fuzzy_pred	Logical; whether to compute fuzzy class memberships for pred.
...	Additional arguments passed to specific methods.

Details

The allowed values for metrics depend on the value of level:

- If level = "element", the allowed metrics are: "nsSPC", "NPC","SpatialSPC".
- If level = "class", the allowed metrics are: "nsWH", "nsAWH", "nsWC","nsAWC".
- If level = "dataset", the allowed metrics are: "nsRI", "nsARI","nsWH","nsAWH", "nsWC","nsAWC", "SpatialAccuracy","SpatialRI","SpatialARI".

Value

A data.frame of metrics based on the specified input.

Examples

```
# Example with individual components
data(sp_toys)
data <- sp_toys
getSpatialExternalMetrics(true=data$label, pred=data$p1,
location=data[,c("x", "y")], k=6, level="class")

# Example with SpatialExperiment object
se_object <- SpatialExperiment::SpatialExperiment(assays=matrix(NA,
                                            ncol = nrow(data[,c("x", "y")]),
                                            nrow = ncol(data[,c("x", "y")])),
                                spatialCoords=as.matrix(data[,c("x", "y")]))
SummarizedExperiment::colData(se_object) <-
                      cbind(SummarizedExperiment::colData(se_object),
                            data.frame(true=data$label, pred=data$p1))
getSpatialExternalMetrics(object=se_object, true="true", pred="pred", k=6,
                          level="class")
```

`getSpatialGlobalExternalMetrics` *Compute dataset-level external evaluation metrics for spatially-resolved data*

Description

Computes a selection of external clustering evaluation metrics for spatial data at the dataset level. Options include a series of fuzzy pair-counting metrics and set matching-based accuracy.

Usage

```
getSpatialGlobalExternalMetrics(
  true,
  pred,
  location,
  k = 6,
  alpha = 0.5,
 metrics = c("nsRI", "nsARI", "nsWH", "nsAWH", "nsWC", "nsAWC", "SpatialAccuracy",
    "SpatialRI", "SpatialARI"),
  fuzzy_true = TRUE,
  fuzzy_pred = FALSE,
  ...
)
```

Arguments

`true`	A vector containing the labels of the true classes. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`pred`	A vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list.
`location`	A matrix or data.frame of coordinates
`k`	The number of neighbors used when calculating the fuzzy class memberships for fuzzy metrics, or when calculating the weighted accuracy.
`alpha`	The parameter to control to what extend the spot itself contribute to the class composition calculation. `"equal"` means it is weighted the same as other neighbors. A numeric value between `0` and `1` means the weight of the frequency contribution for the spot itself, and the frequency contribution for its knn is then `1-alpha`. By default `0.5`.
`metrics`	a vector of metric names to compute.
`fuzzy_true`	Logical; whether to compute fuzzy class memberships for `true`.
`fuzzy_pred`	Logical; whether to compute fuzzy class memberships for `pred`.
`...`	Optional params for `fuzzyPartitionMetrics` or `findSpatialKNN`.

Value

A data.frame of metrics.

`getSpatialGlobalInternalMetrics` *Compute dataset-level internal evaluation metrics for spatially-resolved data*

Description

Computes a selection of internal clustering evaluation metrics for spatial data at the dataset level. MPC, PC and PE are internal metrics for fuzzy clustering, and their implementations in package `fclust` are used.

Usage

```
getSpatialGlobalInternalMetrics(
  labels,
  location,
  k = 6,
  metrics = c("PAS", "ELSA", "CHAOS"),
  ...
)
```

Arguments

`labels`	A vector containing the labels to be evaluated.
`location`	A numerical matrix containing the location information, with rows as samples and columns as location dimensions.
`k`	The size of the spatial neighborhood to look at for each spot. This is used for calculating PAS and ELSA scores.
`metrics`	The metrics to compute. See below for more details.
`...`	Optional arguments for `PAS()`.

Value

A named vector containing metric values. Possible metrics are:

`PAS`	Proportion of abnormal spots (PAS score).
`ELSA`	Entropy-based Local indicator of Spatial Association (ELSA score).
`CHAOS`	Spatial Chaos Score.
`MPC`	Modified partition coefficient
`PC`	Partition coefficient
`PE`	Partition entropy

References

Yuan, Zhiyuan, et al., 2024; 10.1038/s41592-024-02215-8

Naimi, Babak, et al., 2019; 10.1016/j.spasta.2018.10.001

Wang, et al., 2022; 10.1016/j.ins.2022.11.010

`getSpatialInternalMetrics` *Compute internal metrics for spatial data*

Description

A generic function to compute a selection of internal clustering evaluation metrics for spatial data. It can be applied to raw components (`labels`, `location`) or directly to a `SpatialExperiment` object.

Usage

```
getSpatialInternalMetrics(
  object = NULL,
  labels,
  location = NULL,
  k = 6,
  level = "class",
  metrics = c("CHAOS", "PAS", "ELSA"),
  ...
)

## S4 method for signature 'missing'
getSpatialInternalMetrics(
  object = NULL,
  labels,
  location = NULL,
  k = 6,
  level = "class",
  metrics = c("CHAOS", "PAS", "ELSA"),
  ...
)

## S4 method for signature 'SpatialExperiment'
getSpatialInternalMetrics(
  object = NULL,
  labels,
  location = NULL,
  k = 6,
  level = "class",
  metrics = c("CHAOS", "PAS", "ELSA"),
  ...
)
```

Arguments

object	The main input. Can be a SpatialExperiment object or missing (when using labels, and location directly).
labels	When object is missing: a vector containing the labels of the predicted clusters. Must be a vector of characters, integers, numerics, or a factor, but not a list. When object is a SpatialExperiment object: the column name in colData(object) containing the labels.
location	A numerical matrix containing the location information, with rows as samples and columns as location dimensions.
k	The size of the spatial neighborhood to look at for each spot. This is used for calculating PAS and ELSA scores.
level	The level to calculate the metrics. Options include "element", "class" and "dataset".
metrics	The metrics to compute. See details.
...	Optional params for PAS().

Details

The allowed values for metrics depend on the value of level:

- If level = "element", the allowed metrics are: "PAS", "ELSA".
- If level = "class", the allowed metrics are: "CHAOS", "PAS", "ELSA".
- If level = "dataset", the allowed metrics are:
  - "PAS": Proportion of abnormal spots (PAS score)
  - "ELSA": Entropy-based Local indicator of Spatial Association (ELSA score)
  - "CHAOS": Spatial Chaos Score.
  - "MPC": Modified partition coefficient
  - "PC": Partition coefficient
  - "PE": Partition entropy

Value

A data.frame of metrics.

Examples

```
# Example with individual components
data(sp_toys)
data <- sp_toys
getSpatialInternalMetrics(labels=data$label, location=data[,c("x", "y")],
                          k=6, level="class")

# Example with SpatialExperiment object
se_object <- SpatialExperiment::SpatialExperiment(assays=matrix(NA,
                                            ncol = nrow(data[,c("x", "y")]),
                                            nrow = ncol(data[,c("x", "y")])),
```

```
                                spatialCoords=as.matrix(data[,c("x", "y")]))
SummarizedExperiment::colData(se_object) <-
cbind(SummarizedExperiment::colData(se_object),
      data.frame(label=data$label))
getSpatialInternalMetrics(object=se_object, labels="label", k=6,
                          level="class")
```

---

knnComposition *Compute neighborhood composition*

Description

For a given dataset with locations and labels, compute the label composition of the neighborhood for each sample.

Usage

```
knnComposition(location, k = 6, labels, alpha = 0.5, ...)
```

Arguments

location	A numeric data matrix containing location information, where rows are points and columns are location dimensions.
k	The number of nearest neighbors to look at.
labels	A vector containing the label for the dataset.
alpha	The parameter to control to what extend the spot itself contribute to the class composition calculation. "equal" means it is weighted the same as other neighbors. A numeric value between 0 and 1 means the weight of the frequency contribution for the spot itself, and the frequency contribution for its knn is then 1-alpha. By default 0.5.
...	Optional arguments for findSpatialKNN().

Value

A numerical matrix indicating the composition, where rows correspond to samples and columns correspond to the classes in label.

Examples

```
data(sp_toys)
data <- sp_toys
knnComposition(data[,c("x", "y")], k=6, data$label)
```

matchSets *Match two partitions using Hungarian algorithm*

Description

Match sets from a partitions to a reference partition using the Hungarian algorithm to optimize F1 scores.

Usage

```
matchSets(pred, true, forceMatch = TRUE, returnIndices = is.integer(true))
```

Arguments

pred	An integer or factor of cluster labels
true	An integer or factor of reference labels
forceMatch	Logical; whether to enforce a match for every set of pred
returnIndices	Logical; whether to return indices rather than levels

Value

A vector of matching sets (i.e. level) from true for every set (i.e. level) of pred.

metric_info *Metrics Information*

Description

A dataframe storing the information of all metrics. The code to generate the dataset is at system.file('inst/scripts/', 'metric_info.R', package='poem').

Usage

```
metric_info
```

Format

metric_info:
A data frame.

mockData *Generate mock multidimensional data*

Description

Generates mock multidimensional data of a given number of classes of points, for testing.

Usage

```
mockData(
  Ns = c(25, 15),
  classDiff = 2,
  Sds = 1,
  ndims = 2,
  spread = c(1, 2),
  rndFn = rnorm
)
```

Arguments

Ns	A vector of more than one positive integers specifying the number of elements of each class.
classDiff	The distances between the classes. If there are more than 2 classes, this can be a dist object or a symmetric matrix of length(Ns)-1columns/rows where the lower triangle indicates the desired distances between classes.
Sds	The standard deviation. Can either be a fixed value, a value per class, or a matrix of values for each class (rows) and dimension (column).
ndims	The number of dimensions to generate (default 2).
spread	The spread of the points. Can either be a fixed value, a value per class, or a matrix of values for each class (rows) and dimension (col).
rndFn	The random function, by default rnorm, but should also work for rlnorm and similar.

Value

A data.frame with coordinates and a class column.

Examples

```
d <- mockData()
```

`nnWeightedAccuracy` *nnWeightedAccuracy*

Description

Computes an accuracy score which weighs elements/spots that are misclassified by the proportion of their (spatial) neighborhood that is not of the element/spot's predicted class. This reduces the weight of misclassifications happening at the boundary between domains.

Usage

```
nnWeightedAccuracy(true, pred, location, k = 5, ...)
```

Arguments

`true`	True class labels (vector coercible to factor)
`pred`	Predicted labels (vector coercible to factor)
`location`	The spatial coordinates to compute the nearest neighbors.
`k`	Number of nearest neighbors
`...`	Optional params passed to `findSpatialKNN()`.

Value

A scalar representing the weighted accuracy.

`noisy_moon` *The noisy moon dataset*

Description

A simple toy dataset consists of two interleaving half circles. The code to generate the dataset is at system.file('inst/scripts/', 'noisy_moon.R', package='poem').

Usage

```
noisy_moon
```

Format

`noisy_moon`**:**

A data frame with 100 rows and 5 columns:

x, y Coordinates of each observations.

label Ground truth labels. Either 1 or 2.

kmeans_label Predicted clustering labels using kmeans with 2 centers.

hdbscan_label Predicted clustering labels using hdbscan with `minPts = 5`.

PAS *Calculate PAS score*

Description

PAS score measures the clustering performance by calculating the randomness of the spots that located outside of the spatial region where it was clustered to. Lower PAS score indicates better spatial domian clustering performance.

Usage

```
PAS(labels, location, k = 10, ...)
```

Arguments

labels	Cluster labels.
location	A numerical matrix containing the location information, with rows as samples and columns as location dimensions.
k	Number of nearest neighbors.
...	Optional params for findSpatialKNN().

Value

A numeric value for PAS score, and a boolean vector about the abnormal spots.

Examples

```
data(sp_toys)
data <- sp_toys
PAS(data$label, data[,c("x", "y")], k=6)
PAS(data$p1, data[,c("x", "y")], k=6)
PAS(data$p2, data[,c("x", "y")], k=6)
```

setMatchingAccuracy *The non-spatially-weighted counterpart of nnWeightedAccuracy*

Description

The non-spatially-weighted counterpart of nnWeightedAccuracy

Usage

```
setMatchingAccuracy(true, pred)
```

Arguments

true	True class labels (vector coercible to factor)
pred	Predicted labels (vector coercible to factor)

Value

A scalar representing the weighted accuracy.

---

spatialARI *Spatially aware ARI from Yan, Yinqiao, et al. (2025).*

Description

Computes the spatial Rand Index and spatial ARI (Yan, Feng and Luo, 2025). Note that by default, the decay functions are different from those of the original publication (see details for more information), but the latter can be replicated with `original=TRUE`.

Usage

```
spatialARI(
  true,
  pred,
  location,
  normCoords = TRUE,
  lambda = 0.8,
  fbeta = 4,
  hbeta = 1,
  spotWise = FALSE,
  nChunks = NULL,
  original = FALSE,
  f = function(x) {
     lambda * exp(-x * fbeta)
 },
  h = function(x) {
     lambda * (1 - exp(-x * hbeta))
 }
)
```

Arguments

true	A vector of true class labels
pred	A vector of predicted clusters
location	A matrix of spatial coordinates, with dimensions as columns
normCoords	Logical; whether to normalize the coordinates to 0-1.
lambda	The `alpha` used in the `f` and `h` functions (default 0.8) in Yan, Feng and Luo, 2025.

fbeta, hbeta	Additional factors used in the exponential decay functions (see details). A higher value means a faster decay. These are ignored if original=TRUE.
spotWise	Logical; whether to return the spot-wise spatial concordance (not adjusted for chance).
nChunks	The number of processing chunks. If NULL, this will be determined automatically based on the size of the dataset, so as to remain below 2GB RAM usage.
original	Logical; whether to use the original h/f functions from Yan, Feng and Luo (default FALSE). If set to TRUE, the arguments fbeta, hbeta, f and h are ignored.
f	The f function, which determines the positive contribution of pairs that are in different partitions in the reference, but grouped together in the clustering, based on the distance between mates.
h	The h function, which determines the positive contribution of pairs that are in the same partition in the reference, but different ones in the clustering, based on the distance between mates.

Details

This is a reimplementation of the method from the spARI package, made more scalable (i.e. a bit slower but more memory-efficient) through chunk-based processing, extensible to more than 2 dimensions, and with some additional options. Note that by default, this will not produce the same results as the original method: to do so, set original=TRUE. In our exploration of the method and its behavior, we found the decay to be too slow, and we therefore 1) do not square the distances, and 2) introduced a beta parameter in each function which allows to scale it (a higher beta parameter means a faster decay).

By default, chunking to keep RAM usage roughly below 2GB. Higher speed can be achieved (at higher memory costs) for larger datasets by limiting the number of chunks. The memory usage if done in a single chunk should be roughly 4e-5*nrow(location)^2 Mb, and this scales down linearly with the number of chunks.

Value

A vector containing the spatial Rand Index (spRI) and spatial adjusted Rand Index (spARI). Alternatively, if spotWise=TRUE, a vector of spatial pair concordances for each spot.

Author(s)

Pierre-Luc Germain

References

Yan, Feng and Luo, biorxiv 2025, https://doi.org/10.1101/2025.03.25.645156

Examples

```
data(sp_toys)
spatialARI(true=sp_toys$label, pred=sp_toys$p2, location = sp_toys[,1:2])
```

sp_toys *Toy examples of spatial data*

Description

Toy examples of spatial data. The code to generate the dataset is at system.file('inst/scripts/', 'sp_toys.R', package='poem').

Usage

```
sp_toys
```

Format

`sp_toys`:

A data frame with 240 rows and 11 columns, representing a 16 x 15 array of spots:

x, y Coordinates of each spots.

row, col The row and column index of each spots.

label Ground truth labels. Either 1 or 2.

p1-p6 Hypothetical predicted spatial clustering labels.

toyExamples *Toy embedding examples*

Description

Toy example 2D embeddings of elements of different classes, with varying mixing and spread. Graphs 1-3 all have 20 elements of each of 4 classes, but that are mixed in different fashion in the embedding space. Graphs 4-7 all have 100 elements of class1 and 60 of class2, and the class1 elements vary in their spread. The code to generate the dataset is at system.file('inst/scripts/', 'graph_example.R', package='poem').

Usage

```
toyExamples
```

Format

`toyExamples`:

A data frame.

graph The name of the embedding to which the element belongs.

x, y Coordinates in the 2D embedding.

class The class to which the element belongs.

Index