Package 'adjoin'

April 21, 2026

Title Constructing Adjacency Matrices Based on Spatial and Feature Similarity

Version 0.1.0

Description Constructs sparse adjacency matrices from spatial coordinates, feature measurements, class labels, and temporal indices. Supports nearest-neighbor graphs, heat-kernel weights, graph Laplacians, diffusion operators, and bilateral smoothers for graph-based data analysis, following spectral graph methods in von Luxburg (2007) <doi:10.1007/s11222-007-9033-z>, diffusion maps in Coifman and Lafon (2006) <doi:10.1016/j.acha.2006.04.006>, and bilateral filtering in Tomasi and Manduchi (1998) <doi:10.1109/ICCV.1998.710815>.

Depends R (>= 4.0.0)

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/bbuchsbaum/adjoin

BugReports https://github.com/bbuchsbaum/adjoin/issues

RoxygenNote 7.3.3

Imports Rcpp (>= 0.11.3), Matrix, assertthat, Rnanoflann, chk, RSpectra, igraph, parallel, stats, methods, utils, mgcv, proxy, corpcor

Suggests testthat (>= 3.0.0), RcppHNSW (>= 0.6.0), knitr, rmarkdown, furrr, crayon, ggplot2

Config/testthat/edition 3

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

Config/Needs/website albersdown

NeedsCompilation yes

Author Bradley R. Buchsbaum [aut, cre] (ORCID: <https://orcid.org/0000-0002-1108-4866>)

Maintainer Bradley R. Buchsbaum <brad.buchsbaum@gmail.com>

Repository CRAN

Date/Publication 2026-04-21 18:12:30 UTC

Contents

`adjacency` *Extract Adjacency Matrix from Graph Objects*

Description

Extract the adjacency matrix from graph objects such as neighbor_graph or nnsearch objects.

Usage

```
adjacency(x, ...)
```

Arguments

`x`	A graph object (neighbor_graph, nnsearch, etc.).
`...`	Additional arguments passed to specific methods.

Value

A sparse Matrix object representing the adjacency matrix.

Examples

```
adj_matrix <- matrix(c(0,1,0,
                       1,0,1,
                       0,1,0), nrow=3, byrow=TRUE)
ng <- neighbor_graph(adj_matrix)
adjacency(ng)
```

`adjacency.neighbor_graph`
Extract adjacency matrix from neighbor_graph object

Description

Extract the adjacency matrix from a neighbor_graph object.

Usage

```
## S3 method for class 'neighbor_graph'
adjacency(x, attr = "weight", ...)
```

Arguments

x	A neighbor_graph object.
attr	A character string specifying the edge attribute to use for weights (default: "weight").
...	Additional arguments (currently ignored).

Value

A sparse Matrix object representing the adjacency matrix.

Examples

```
adj_matrix <- Matrix::Matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0),
                             nrow = 3, byrow = TRUE, sparse = TRUE)
ng <- neighbor_graph(adj_matrix)
adj <- adjacency(ng)
```

---

adjacency.nn_search *Create Adjacency Matrix from nnsearch Object*

Description

Convert a nearest neighbor search result to a sparse adjacency matrix.

Usage

```
## S3 method for class 'nn_search'
adjacency(
  x,
  idim = nrow(x$indices),
  jdim = max(x$indices),
  return_triplet = FALSE,
  ...
)
```

Arguments

x	An object of class "nnsearch".
idim	The number of rows in the resulting matrix.
jdim	The number of columns in the resulting matrix.
return_triplet	Logical; whether to return triplet format.
...	Additional arguments (currently ignored).

Value

A sparse Matrix representing the adjacency matrix.

Examples

```
res <- list(indices = matrix(c(2L,1L), nrow=2),
            distances = matrix(c(0.1,0.2), nrow=2))
class(res) <- "nn_search"
attr(res,"len") <- 2; attr(res,"metric") <- "l2"
adjacency(res, idim=2, jdim=2)
```

`between_class_neighbors` *Between-Class Neighbors*

Description

A generic function to compute the between-class neighbors of a graph.

Usage

```
between_class_neighbors(x, ng, ...)
```

Arguments

`x`	An object.
`ng`	A neighbor graph object.
`...`	Additional arguments passed to specific methods.

Value

An object representing the between-class neighbors of the input graph, the structure of which depends on the input object's class.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
ng <- neighbor_graph(matrix(c(0,1,1,1,0,1,1,1,0),3))
between_class_neighbors(cg, ng)
```

between_class_neighbors.class_graph *Between-Class Neighbors for class_graph Objects*

Description

Compute the between-class neighbors of a class_graph object.

Usage

```
## S3 method for class 'class_graph'
between_class_neighbors(x, ng, ...)
```

Arguments

x	A class_graph object.
ng	A neighbor_graph object.
...	Additional arguments (currently ignored).

Value

A neighbor_graph object representing the between-class neighbors.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
ng <- neighbor_graph(matrix(c(0,1,1,1,0,1,1,1,0),3))
between_class_neighbors(cg, ng)
```

bilateral_smoother *Bilateral Spatial Smoother*

Description

This function computes a bilateral smoothing of the input data, which combines spatial and feature information to provide a smoothed representation of the data.

Usage

```
bilateral_smoother(
  coord_mat,
  feature_mat,
  nnk = 27,
  s_sigma = 2.5,
  f_sigma = 0.7,
  stochastic = FALSE
)
```

Arguments

coord_mat	A matrix with the spatial coordinates of the data points, where each row represents a point and each column represents a coordinate dimension.
feature_mat	A matrix with the feature vectors of the data points, where each row represents a point and each column represents a feature dimension.
nnk	The number of nearest neighbors to consider for smoothing (default: 27). Must be >= 4.
s_sigma	The spatial bandwidth in standard deviations (default: 2.5).
f_sigma	The normalized feature bandwidth in standard deviations (default: 0.7).
stochastic	A logical value indicating whether to make the resulting adjacency matrix doubly stochastic (default: FALSE).

Value

A sparse adjacency matrix representing the smoothed data.

Examples

```
set.seed(123)
coord_mat <- as.matrix(expand.grid(1:10, 1:10))
feature_mat <- matrix(rnorm(100*10), 100, 10)
S <- bilateral_smoother(coord_mat, feature_mat, nnk=8)
```

---

binary_label_matrix *Create a Binary Label Adjacency Matrix (All Pairs)*

Description

Constructs a binary adjacency matrix based on two sets of labels ‘a‘ and ‘b‘, creating edges for ALL pairs (i, j) where labels match (type="s") or differ (type="d"). This computes the full cross-product comparison between the two label vectors.

Usage

```
binary_label_matrix(a, b = NULL, type = c("s", "d"))
```

Arguments

a	A vector of labels for the first set of data points.
b	A vector of labels for the second set of data points (default: NULL). If NULL, ‘b‘ will be set to ‘a‘.
type	A character specifying the type of adjacency matrix to create, either "s" for same labels or "d" for different labels (default: "s").

Details

For type="s", the result is a block-diagonal structure when a==b, with blocks corresponding to each class. For type="d", the result is the complement.

This function uses efficient sparse matrix multiplication via indicator matrices, avoiding O(n^2) memory usage from expanding all pairs.

Value

A sparse binary adjacency matrix of dimensions (length(a) x length(b)) with 1s where the label relationship holds.

See Also

diagonal_label_matrix for element-wise (positional) comparison

Examples

```
data(iris)
a <- iris[,5]
bl <- binary_label_matrix(a, type="d")
```

---

class_graph *Construct a Class Graph*

Description

A graph in which members of the same class have edges.

Usage

```
class_graph(labels, sparse = TRUE)
```

Arguments

labels	A vector of class labels.
sparse	A logical value, indicating whether to use sparse matrices in the computation. Default is TRUE.

Value

A class_graph object, which is a list containing the following components:

adjacency A matrix representing the adjacency of the graph.

params A list of parameters used in the construction of the graph.

labels A vector of class labels.

class_indices A list of vectors, each containing the indices of elements belonging to a specific class.

class_freq A table of frequencies for each class.

levels A vector of unique class labels.

classes A character string indicating the type of graph ("class_graph").

Examples

```
data(iris)
labels <- iris[,5]
cg <- class_graph(labels)
```

---

class_means *Class Means*

Description

A generic function to compute the mean of each class.

Usage

```
class_means(x, ...)
```

Arguments

x	An object (e.g., class_graph).
...	Additional arguments passed to specific methods.

Value

A matrix or data frame representing the means of each class, the structure of which depends on the input object's class.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
class_means(cg, matrix(1:9, nrow=3))
```

class_means.class_graph *Class Means for class_graph Objects*

Description

Compute the mean of each class for a class_graph object.

Usage

```
## S3 method for class 'class_graph'
class_means(x, X, ...)
```

Arguments

x	A class_graph object.
X	The data matrix corresponding to the graph nodes.
...	Additional arguments (currently ignored).

Value

A matrix where each row represents the mean values for each class.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
class_means(cg, matrix(1:9, nrow=3))
```

commute_time_distance *Compute the commute-time distance between nodes in a graph*

Description

This function computes the commute-time distance between nodes in a graph using either eigenvalue or pseudoinverse methods.

Usage

```
commute_time_distance(A, ncomp = nrow(A) - 1)
```

Arguments

A	A symmetric, non-negative matrix representing the adjacency matrix of the graph
ncomp	Integer, number of components to use in the computation, default is (nrow(A) - 1)

Value

A list with the following components:

`eigenvectors`	Matrix, eigenvectors of the matrix M
`eigenvalues`	Vector, eigenvalues of the matrix M
`cds`	Matrix, the computed commute-time distances
`gap`	Numeric, the gap between the two largest eigenvalues

The returned object has class "commute_time" and "list".

Examples

```
A <- matrix(c(0, 1, 1, 0,
              1, 0, 1, 1,
              1, 1, 0, 1,
              0, 1, 1, 0), nrow = 4, byrow = TRUE)

result <- commute_time_distance(A)
```

---

compute_diffusion_kernel

Compute Markov diffusion kernel via eigen decomposition

Description

Efficient computation of the Markov diffusion kernel for a graph represented by a sparse adjacency matrix. For large graphs, uses RSpectra to compute only the leading k eigenpairs of the normalized transition matrix.

Usage

```
compute_diffusion_kernel(A, t, k = NULL, symmetric = TRUE)
```

Arguments

`A`	Square sparse adjacency matrix (dgCMatrix) of an undirected, weighted graph with non-negative entries.
`t`	Diffusion time parameter (positive scalar).
`k`	Number of leading eigenpairs to compute. If NULL, performs full eigendecomposition.
`symmetric`	If TRUE (default), uses symmetric normalization to guarantee real eigenvalues.

Value

dgCMatrix representing the diffusion kernel matrix.

Examples

```
library(Matrix)
A <- sparseMatrix(i = c(1, 2, 3, 4), j = c(2, 3, 4, 5),
                  x = c(1, 1, 1, 1), dims = c(5, 5))
A <- A + t(A)  # Make symmetric

K <- compute_diffusion_kernel(A, t = 0.5)

K_approx <- compute_diffusion_kernel(A, t = 0.5, k = 3)
```

---

compute_diffusion_map *Diffusion map embedding and distance*

Description

Computes the diffusion map embedding of a graph and the pairwise diffusion distances based on the leading eigenvectors of the normalized transition matrix.

Usage

```
compute_diffusion_map(A, t, k = 10)
```

Arguments

A	Square sparse adjacency matrix (dgCMatrix) of an undirected, weighted graph.
t	Diffusion time parameter (positive scalar).
k	Number of diffusion coordinates to compute, excluding the trivial first coordinate.

Value

A list with two components:

embedding	n×k matrix of diffusion coordinates where n is the number of nodes.
distances	n×n matrix of squared diffusion distances between all node pairs.

Examples

```
library(Matrix)
A <- sparseMatrix(i = c(1, 2, 3), j = c(2, 3, 4), x = c(1, 1, 1), dims = c(4, 4))
A <- A + t(A)  # Make symmetric

result <- compute_diffusion_map(A, t = 1.0, k = 2)

print(result$embedding)

print(result$distances[1, ])  # distances from node 1
```

convolve_matrix *Convolve a Data Matrix with a Kernel Matrix*

Description

Performs right-multiplication of a data matrix ‘X‘ by a kernel matrix ‘Kern‘, optionally with symmetric normalization.

Usage

```
convolve_matrix(X, Kern, normalize = FALSE)
```

Arguments

X	A data matrix to be transformed (n x p).
Kern	A square kernel matrix (p x p) used for the transformation.
normalize	A logical flag indicating whether to apply symmetric normalization D^(-1/2) Kern D^(-1/2) before multiplication (default: FALSE).

Value

A matrix resulting from X %*% Kern (or normalized version).

A matrix resulting from `X %*% Kern` (or the normalized version when `normalize=TRUE`).

Examples

```
X <- matrix(1:6, nrow=2)
K <- diag(3)
convolve_matrix(X, K)
```

cross_adjacency *Cross Adjacency*

Description

This function computes the cross adjacency matrix or graph between two sets of points based on their k-nearest neighbors and a kernel function applied to their distances.

Usage

```
cross_adjacency(
  X,
  Y,
  k = 5,
  FUN = heat_kernel,
  type = c("normal", "mutual", "asym"),
  as = c("igraph", "sparse", "index_sim"),
  backend = c("nanoflann", "hnsw"),
  M = 16,
  ef = 200
)
```

Arguments

X	A matrix of size nXk, where n is the number of data points and k is the dimensionality of the feature space.
Y	A matrix of size pXk, where p is the number of query points and k is the dimensionality of the feature space.
k	An integer indicating the number of nearest neighbors to consider (default: 5).
FUN	A kernel function to apply to the Euclidean distances between data points (default: heat_kernel).
type	A character string indicating the type of adjacency to compute. One of "normal", "mutual", or "asym" (default: "normal").
as	A character string indicating the format of the output. One of "igraph", "sparse", or "index_sim" (default: "igraph").
backend	Nearest-neighbor backend. ‘"nanoflann"‘ uses exact Euclidean search; ‘"hnsw"‘ uses approximate search via ‘RcppHNSW‘.
M, ef	HNSW tuning parameters used only when ‘backend = "hnsw"‘. Larger ‘ef‘ usually improves recall at the cost of runtime.

Details

Distances passed to ‘FUN‘ are Euclidean distances. The default ‘backend = "nanoflann"‘ uses exact Euclidean search. Set ‘backend = "hnsw"‘ to opt in to approximate search via ‘RcppHNSW‘; squared L2 distances from ‘RcppHNSW‘ are converted back to Euclidean distances before weighting.

Value

If ’as’ is "index_sim", a two-column matrix where the first column contains the indices of nearest neighbors and the second column contains the corresponding kernel values. If ’as’ is "igraph", an igraph object representing the cross adjacency graph. If ’as’ is "sparse", a sparse adjacency matrix.

Examples

```
X <- matrix(rnorm(6), ncol=2)
Y <- matrix(rnorm(8), ncol=2)
cross_adjacency(X, Y, k=1, as="sparse")
```

`cross_spatial_adjacency` *Cross Spatial Adjacency*

Description

Constructs a cross spatial adjacency matrix between two sets of points, where weights are determined by spatial relationships.

Usage

```
cross_spatial_adjacency(
  coord_mat1,
  coord_mat2,
  dthresh = sigma * 3,
  nnk = 27,
  weight_mode = c("binary", "heat"),
  sigma = 5,
  normalized = TRUE
)
```

Arguments

`coord_mat1`	A matrix with the spatial coordinates of the first set of data points, where each row represents a point and each column represents a coordinate dimension.
`coord_mat2`	A matrix with the spatial coordinates of the second set of data points, where each row represents a point and each column represents a coordinate dimension.
`dthresh`	The distance threshold for nearest neighbors (default: sigma * 3).
`nnk`	The number of nearest neighbors to consider (default: 27).
`weight_mode`	The mode to use for weighting the adjacency matrix, either "binary" or "heat" (default: "binary").
`sigma`	The bandwidth for heat kernel weights (default: 5).
`normalized`	A logical value indicating whether to normalize the adjacency matrix (default: TRUE).

Value

A sparse cross adjacency matrix with weighted spatial relationships between the two sets of points.

Examples

```
coord_mat1 <- as.matrix(expand.grid(x=1:5, y=1:5, z=1:5))
coord_mat2 <- as.matrix(expand.grid(x=6:10, y=6:10, z=6:10))
csa <- cross_spatial_adjacency(coord_mat1, coord_mat2, nnk=3,
                               weight_mode="binary", sigma=5,
                               normalized=TRUE)
```

---

cross_weighted_spatial_adjacency

Cross-adjacency matrix with feature weighting

Description

Cross-adjacency matrix with feature weighting

Usage

```
cross_weighted_spatial_adjacency(
  coord_mat1,
  coord_mat2,
  feature_mat1,
  feature_mat2,
  wsigma = 0.73,
  alpha = 0.5,
  nnk = 27,
  maxk = nnk,
  weight_mode = c("binary", "heat"),
  sigma = 1,
  dthresh = sigma * 2.5,
  normalized = TRUE
)
```

Arguments

coord_mat1	the first coordinate matrix (the query)
coord_mat2	the second coordinate matrix (the reference)
feature_mat1	the first feature matrix
feature_mat2	the second feature matrix
wsigma	the sigma for the feature heat kernel
alpha	the mixing weight for the spatial distance (1=all spatial weighting, 0=all feature weighting)
nnk	the maximum number of spatial nearest neighbors to include
maxk	the maximum number of neighbors to include within spatial window

weight_mode	the type of weighting to use: "binary" or "heat" (default: "binary")
sigma	the spatial sigma for the heat kernel weighting (default: 1)
dthresh	the threshold for the spatial distance
normalized	whether to normalize the rows to sum to 1

Value

A sparse cross-graph adjacency matrix of feature-weighted spatial similarities.

Examples

```
set.seed(123)
coords <- as.matrix(expand.grid(1:5, 1:5))
fmat1 <- matrix(rnorm(5*25), 25, 5)
fmat2 <- matrix(rnorm(5*25), 25, 5)

adj <- cross_weighted_spatial_adjacency(coords, coords, fmat1, fmat2)
```

---

design_kernel *Build a flexible design-similarity kernel*

Description

Constructs a positive semi-definite (PSD) kernel K over design regressors that encodes factorial similarity: same level similarity, optional smoothness across ordinal levels, and interaction structure via Kronecker composition. Works either directly in cell space (one regressor per cell) or in effect-coded space (main effects, interactions), by pulling K back through user-specified contrast matrices.

Usage

```
design_kernel(
  factors,
  terms = NULL,
  rho = NULL,
  include_intercept = TRUE,
  rho0 = 1e-08,
  basis = c("cell", "effect"),
  contrasts = NULL,
  block_structure = NULL,
  normalize = c("none", "unit_trace", "unit_fro", "max_diag"),
  jitter = 1e-08
)
```

Arguments

factors	A named list, one entry per factor, e.g. list(A=list(L=5,type="nominal"), B=list(L=5,type="ordinal", l=1.5)). For ordinal, supply length-scale l (>0). Supported types: "nominal", "ordinal", "circular" (wrap-around distances).
terms	A list of character vectors; each character vector lists factor names that participate in that term. Examples: list("A", "B", c("A","B")) for main A, main B, A:B. If NULL, defaults to all singletons (main effects) and the full interaction.
rho	A named numeric vector of nonnegative weights for each term (names like "A", "B", "A:B"). If NULL, defaults to 1 for each term. Use rho0 for the identity term (see below).
include_intercept	Logical; if TRUE, adds rho0 * I to K (small ridge / identity term).
rho0	Nonnegative scalar weight for the identity term; default 1e-8.
basis	Either "cell" (default) or "effect". If "effect", you must supply ‘contrasts‘.
contrasts	A named list of contrast matrices for each factor, with dimensions L_i x d_i. For example: list(A = contr.sum(5), B = contr.sum(5)). You can also pass orthonormal Helmert.
block_structure	If basis="effect", a character vector naming the sequence of effect blocks to include, e.g., c("A","B","A:B"). If NULL, inferred from ‘terms‘ and ‘contrasts‘.
normalize	One of c("none","unit_trace","unit_fro","max_diag").
jitter	Small diagonal added to ensure SPD; default 1e-8.

Details

The kernel is constructed using the following principles:

- For each factor i with L_i levels, define a per-factor kernel K_i (nominal or ordinal)
- For any term S (subset of factors), construct a term kernel by Kronecker product: K_S = kronecker(K_i, K_j, ...) for i,j in S and kronecker with J matrices for factors not in S
- Combine term kernels with nonnegative weights rho[S] and optionally add a small ridge
- If using effect coding, map the cell kernel to effect-space: K_effect = T’ K_cell T

Value

A list with elements:

K	PSD kernel matrix in the requested basis (cell or effect)
K_cell	The cell-space kernel matrix (always returned)
info	A list containing levels, factor_names, term_names, basis, map (T matrix for effect basis), blocks

Examples

```
factors <- list(
  A = list(L=2, type="nominal"),
  B = list(L=3, type="nominal")
)
K1 <- design_kernel(factors)
print(dim(K1$K))  # 6x6 cell-space kernel

factors <- list(
  dose = list(L=3, type="ordinal", l=1.0),
  treat = list(L=2, type="nominal")
)
K2 <- design_kernel(factors)
print(dim(K2$K))  # 6x6 cell-space kernel
```

---

diagonal_label_matrix *Create a Diagonal Label Comparison Matrix (Element-wise)*

Description

Compares labels at corresponding positions (element-wise) between two equal-length vectors ‘a‘ and ‘b‘. Creates a sparse matrix with entries only on the diagonal, where position (i, i) is 1 if ‘a[i]‘ and ‘b[i]‘ satisfy the comparison.

Usage

```
diagonal_label_matrix(
  a,
  b,
  type = c("s", "d"),
  dim1 = length(a),
  dim2 = length(b)
)
```

Arguments

a	A vector of labels for the first set of data points.
b	A vector of labels for the second set of data points. Must have same length as ‘a‘.
type	A character specifying the comparison type: "s" for same labels (a[i] == b[i]) or "d" for different labels (a[i] != b[i]). Default is "s".
dim1	The row dimension of the output matrix (default: length(a)).
dim2	The column dimension of the output matrix (default: length(b)).

Details

This function performs element-wise comparison, NOT all-pairs comparison. For all-pairs comparison (block structure), use binary_label_matrix.

The vectors ‘a‘ and ‘b‘ must have the same length. If they differ, recycling will occur which is likely unintended.

Value

A sparse diagonal matrix where entry (i, i) is 1 if the labels at position i satisfy the comparison, 0 otherwise.

See Also

binary_label_matrix for all-pairs comparison

Examples

```
a <- factor(c("x","y","x"))
b <- factor(c("x","x","y"))
diagonal_label_matrix(a, b, type="d")
```

---

diagonal_label_matrix_na

Diagonal Label Comparison with NA Handling

Description

Compares labels at corresponding positions (element-wise) between two equal-length vectors ‘a‘ and ‘b‘, with explicit NA handling. Creates a sparse matrix with entries only on the diagonal.

Usage

```
diagonal_label_matrix_na(
  a,
  b,
  type = c("s", "d"),
  return_matrix = TRUE,
  dim1 = length(a),
  dim2 = length(b)
)
```

Arguments

a	The first categorical label vector.
b	The second categorical label vector. Must have same length as ‘a‘.
type	The type of comparison: "s" for same labels (a[i] == b[i]) or "d" for different labels (a[i] != b[i]). Default is "s".
return_matrix	A logical flag indicating whether to return the result as a sparse matrix (default: TRUE) or a triplet matrix with columns (i, j, x).
dim1	The row dimension of the output matrix (default: length(a)).
dim2	The column dimension of the output matrix (default: length(b)).

Details

This function performs element-wise (positional) comparison, NOT all-pairs comparison. Positions where either label is NA are excluded from the result.

For all-pairs comparison (block structure), use binary_label_matrix. For diagonal comparison without NA handling, use diagonal_label_matrix.

Value

If return_matrix is TRUE, a sparse diagonal matrix where entry (i, i) is 1 if the labels at position i satisfy the comparison (and neither is NA). If return_matrix is FALSE, a 3-column matrix of (row, col, value) triplets.

See Also

binary_label_matrix, diagonal_label_matrix

Examples

```
a <- c("x","y", NA)
b <- c("x","y","y")
diagonal_label_matrix_na(a, b, type="s", return_matrix=TRUE)
```

---

difference_of_gauss *Compute the Difference of Gaussians for a coordinate matrix*

Description

This function computes the Difference of Gaussians for a given coordinate matrix using specified sigma values and the number of nearest neighbors.

Usage

```
difference_of_gauss(
  coord_mat,
  sigma1 = 2,
  sigma2 = sigma1 * 1.6,
  nnk = min(nrow(coord_mat), max(27, ncol(coord_mat)^2))
)
```

Arguments

coord_mat	A numeric matrix representing coordinates
sigma1	Numeric, the first sigma parameter for the Gaussian smoother (default is 2)
sigma2	Numeric, the second sigma parameter for the Gaussian smoother (default is sigma1 * 1.6)
nnk	Integer, the number of nearest neighbors for adjacency (default is 3^(ncol(coord_mat)))

Value

A sparse symmetric matrix representing the computed Difference of Gaussians

Examples

```
coord_mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, ncol = 2)

result <- difference_of_gauss(coord_mat, sigma1 = 2, sigma2 = 3.2)
```

---

discriminating_distance

Compute Discriminating Distance for Similarity Graph

Description

This function computes a discriminating distance matrix for the similarity graph based on the class labels. It adjusts the similarity graph by modifying the weights within and between classes, making it more suitable for tasks like classification and clustering.

Usage

```
discriminating_distance(X, labels, k = NULL, sigma = NULL)
```

Arguments

X	A numeric matrix or data frame containing the data points.
labels	A factor or numeric vector containing the class labels for each data point.
k	An integer representing the number of nearest neighbors to consider. Default is half the number of samples.
sigma	A numeric value representing the scaling factor for the heat kernel. If not provided, it will be estimated.

Value

A discriminating distance matrix in the form of a sparse matrix.

Examples

```
X <- matrix(rnorm(100*100), 100, 100)
labels <- factor(rep(1:5, each=20))
sigma <- 0.7
D <- discriminating_distance(X, labels, k=length(labels)/2, sigma=sigma)
```

---

discriminating_similarity

Compute Similarity Graph Weighted by Class Structure

Description

This function computes a similarity graph that is weighted by the class structure of the data. It is useful for preserving the local similarity and diversity within the data, making it suitable for tasks like face and handwriting digits recognition.

Usage

```
discriminating_similarity(X, k, sigma, cg, threshold = 0.01)
```

Arguments

X	A numeric matrix or data frame containing the data points.
k	An integer representing the number of nearest neighbors to consider.
sigma	A numeric value representing the scaling factor for the heat kernel.
cg	A class_graph object computed from the labels.
threshold	A numeric value representing the threshold for the class graph. Default is 0.01.

Value

A weighted similarity graph in the form of a sparse matrix.

References

Local similarity and diversity preserving discriminant projection for face and handwriting digits recognition

Examples

```
X <- matrix(rnorm(100*100), 100, 100)
labels <- factor(rep(1:5, each=20))
cg <- class_graph(labels)
sigma <- 0.7
W <- discriminating_similarity(X, k=length(labels)/2, sigma, cg)
```

---

dist_to_sim *Convert Distance to Similarity*

Description

Convert distance values to similarity values using various transformation methods.

Usage

```
dist_to_sim(x, ...)
```

Arguments

x	An object representing distances (Matrix, nn_search, etc.).
...	Additional arguments passed to specific methods.

Value

A similarity matrix or object with distances converted to similarities.

Examples

```
d <- Matrix::Matrix(as.matrix(dist(matrix(rnorm(6), ncol=2))), sparse=TRUE)
dist_to_sim(d, method="heat", sigma=1)
```

dist_to_sim.Matrix *Convert Distance to Similarity for Matrix Objects*

Description

Convert distance values in a sparse Matrix to similarity values.

Usage

```
## S3 method for class 'Matrix'
dist_to_sim(
  x,
  method = c("heat", "binary", "normalized", "cosine", "correlation"),
  sigma = 1,
  len = 1,
  ...
)
```

Arguments

x	A Matrix object containing distances.
method	The transformation method for converting distances to similarities.
sigma	The bandwidth parameter for the heat kernel method.
len	The length parameter used in transformation calculations.
...	Additional arguments (currently ignored).

Value

The Matrix object with distances converted to similarities.

Examples

```
m <- Matrix::Matrix(c(0,1,2,0), nrow=2, sparse=TRUE)
dist_to_sim(m, method="heat", sigma=1)
```

dist_to_sim.nn_search *Convert Distance to Similarity for nn_search Objects*

Description

Convert distance values in a nearest neighbor search result to similarity values.

Usage

```
## S3 method for class 'nn_search'
dist_to_sim(
  x,
  method = c("heat", "binary", "normalized", "cosine", "correlation"),
  sigma = 1,
  ...
)
```

Arguments

x	An object of class "nn_search".
method	The transformation method for converting distances to similarities.
sigma	The bandwidth parameter for the heat kernel method.
...	Additional arguments (currently ignored).

Value

The modified nn_search object with distances converted to similarities.

Examples

```
res <- list(indices = matrix(c(1L,2L), nrow=1),
            distances = matrix(c(0.5, 1.0), nrow=1))
class(res) <- "nn_search"; attr(res,"len") <- 2; attr(res,"metric") <- "l2"
dist_to_sim(res, method="heat", sigma=1)
```

---

edges *Edges for Graph-Like Objects*

Description

Retrieve the edges of a graph-like object.

Usage

```
edges(x, ...)
```

Arguments

x	A graph-like object.
...	Further arguments passed to or from other methods.

Value

A matrix containing the edges of the graph-like object.

Examples

```
adj_matrix <- matrix(c(0,1,0,
                       1,0,1,
                       0,1,0), nrow=3, byrow=TRUE)
ng <- neighbor_graph(adj_matrix)
edges(ng)
```

`edges.neighbor_graph` *Extract edges from neighbor_graph object*

Description

Retrieve the edges of a neighbor_graph object.

Usage

```
## S3 method for class 'neighbor_graph'
edges(x, ...)
```

Arguments

x	A neighbor_graph object.
...	Additional arguments (currently ignored).

Value

A two-column matrix containing the edges, where each row represents an edge between two nodes.

Examples

```
adj_matrix <- Matrix::Matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0),
                             nrow = 3, byrow = TRUE, sparse = TRUE)
ng <- neighbor_graph(adj_matrix)
edge_list <- edges(ng)
```

estimate_sigma *Estimate Bandwidth Parameter (Sigma) for the Heat Kernel*

Description

Estimate a reasonable bandwidth parameter (sigma) for the heat kernel based on a data matrix and the specified quantile of the frequency distribution of distances.

Usage

```
estimate_sigma(X, prop = 0.25, nsamples = 500, normalized = FALSE)
```

Arguments

X	A data matrix where samples are rows and features are columns.
prop	A numeric value representing the quantile of the frequency distribution of distances used to determine the bandwidth parameter. Default is 0.25.
nsamples	An integer representing the number of samples to draw from the data matrix. Default is 500.
normalized	A logical value indicating whether to normalize the data. Default is FALSE.

Value

A numeric value representing the estimated bandwidth parameter (sigma) for the heat kernel.

Examples

```
X <- matrix(rnorm(1000), nrow=100, ncol=10)

sigma_default <- estimate_sigma(X)

sigma_custom <- estimate_sigma(X, prop=0.3, nsamples=300)
```

example_kernel_5x5 *Example 5x5 factorial kernel*

Description

Creates a simple 5x5 factorial design kernel with two nominal factors, useful for testing and demonstration purposes.

Usage

```
example_kernel_5x5(rho0 = 1e-08, rhoA = 1, rhoB = 1, rhoAB = 0)
```

Arguments

rho0	Weight for identity/ridge term (default 1e-8)
rhoA	Weight for main effect of factor A (default 1)
rhoB	Weight for main effect of factor B (default 1)
rhoAB	Weight for A:B interaction (default 0)

Value

A list with kernel matrices and metadata (see design_kernel)

Examples

```
K1 <- example_kernel_5x5(rhoA=1, rhoB=1, rhoAB=0)

K2 <- example_kernel_5x5(rhoA=1, rhoB=1, rhoAB=1)
```

---

expand_label_similarity

Expand Similarity Between Labels Based on a Precomputed Similarity Matrix

---

Description

Expands the similarity between labels based on a precomputed similarity matrix, ‘sim_mat‘, with either above-threshold or below-threshold values depending on the value of the ‘above‘ parameter.

Usage

```
expand_label_similarity(labels, sim_mat, threshold = 0, above = TRUE)
```

Arguments

labels	A vector of labels for which the similarities will be expanded.
sim_mat	A precomputed similarity matrix containing similarities between the unique labels.
threshold	A threshold value used to filter the expanded similarity values (default: 0).
above	A boolean flag indicating whether to include the values above the threshold (default: TRUE) or below the threshold (FALSE).

Value

A sparse symmetric similarity matrix with the expanded similarity values.

Examples

```
labels <- c("a","b","a")
smat <- matrix(c(1,.2,.2, 0.2,1,0.5, 0.2,0.5,1), nrow=3,
               dimnames=list(c("a","b","c"), c("a","b","c")))
expand_label_similarity(labels, smat, threshold=0.1)
```

`factor_sim` *Compute Similarity Matrix for Factors in a Data Frame*

Description

Calculate the similarity matrix for a set of factors in a data frame using various similarity methods.

Usage

```
factor_sim(des, method = c("Jaccard", "Rogers", "simple matching", "Dice"))
```

Arguments

`des`	A data frame containing factors for which the similarity matrix will be computed.
`method`	A character vector specifying the method used for computing the similarity. The available methods are: • "Jaccard" - Jaccard similarity coefficient • "Rogers" - Rogers and Tanimoto similarity coefficient • "simple matching" - Simple matching coefficient • "Dice" - Dice similarity coefficient

Details

The `factor_sim` function computes the similarity matrix for a set of factors in a data frame using the chosen method. The function first converts the data frame into a model matrix, then calculates the similarity matrix using the `proxy::simil` function from the `proxy` package.

The function supports four similarity methods: Jaccard, Rogers, simple matching, and Dice. The choice of method depends on the specific use case and the desired properties of the similarity measure.

Value

A similarity matrix computed using the specified method for the factors in the data frame.

Examples

```
des <- data.frame(
  var1 = factor(c("a", "b", "a", "b", "a")),
  var2 = factor(c("c", "c", "d", "d", "d"))
)

sim_jaccard <- factor_sim(des, method = "Jaccard")

sim_dice <- factor_sim(des, method = "Dice")
```

`feature_weighted_spatial_constraints` *Construct Feature-Weighted Spatial Constraints for Data Blocks*

Description

This function creates a sparse matrix of feature-weighted spatial constraints for a set of data blocks. The feature-weighted spatial constraints matrix is useful in applications like image segmentation and analysis, where both spatial and feature information are crucial for identifying different regions in the image.

Usage

```
feature_weighted_spatial_constraints(
  coords,
  feature_mats,
  sigma_within = 5,
  sigma_between = 3,
  wsigma_within = 0.73,
  wsigma_between = 0.73,
  alpha_within = 0.5,
  alpha_between = 0.5,
  shrinkage_factor = 0.1,
  nnk_within = 27,
  nnk_between = 27,
  maxk_within = nnk_within,
  maxk_between = nnk_between,
  weight_mode_within = "heat",
  weight_mode_between = "binary",
  variable_weights = rep(1, ncol(coords) * length(feature_mats)),
  verbose = FALSE
)
```

Arguments

`coords`	The spatial coordinates as a matrix with rows as objects and columns as dimensions.

feature_mats	A list of feature matrices, one for each data block.
sigma_within	The bandwidth of the within-block smoother. Default is 5.
sigma_between	The bandwidth of the between-block smoother. Default is 3.
wsigma_within	The bandwidth of the within-block feature weights. Default is 0.73.
wsigma_between	The bandwidth of the between-block feature weights. Default is 0.73.
alpha_within	The scaling factor for within-block feature weights. Default is 0.5.
alpha_between	The scaling factor for between-block feature weights. Default is 0.5.
shrinkage_factor	The amount of shrinkage towards the spatial block average. Default is 0.1.
nnk_within	The maximum number of nearest neighbors for within-block smoother. Default is 27.
nnk_between	The maximum number of nearest neighbors for between-block smoother. Default is 27.
maxk_within	The maximum number of nearest neighbors for within-block computation. Default is ‘nnk_within‘.
maxk_between	The maximum number of nearest neighbors for between-block computation. Default is ‘nnk_between‘.
weight_mode_within	The within-block nearest neighbor weight mode ("heat" or "binary"). Default is "heat".
weight_mode_between	The between-block nearest neighbor weight mode ("heat" or "binary"). Default is "binary".
variable_weights	A vector of per-variable weights. Default is a vector of ones with length equal to the product of the number of columns in the ‘coords‘ matrix and the length of ‘feature_mats‘.
verbose	A boolean indicating whether to print progress messages. Default is FALSE.

Value

A sparse matrix representing the feature-weighted spatial constraints for the provided data blocks.

Details

The function computes within-block and between-block constraints based on the provided coordinates, feature matrices, and other input parameters. It balances the within-block and between-block constraints using a shrinkage factor, and normalizes the resulting matrix by the first eigenvalue. The function also takes into account the weights of the variables in the provided feature matrices.

Examples

```
set.seed(123)
coords <- as.matrix(expand.grid(1:4, 1:4))
fmats <- replicate(3, matrix(rnorm(16 * 4), 4, 16), simplify = FALSE)
```

```
conmat <- feature_weighted_spatial_constraints(
  coords, fmats,
  sigma_within = 1.5, sigma_between = 1.5,
  nnk_within = 4, nnk_between = 4,
  maxk_within = 3, maxk_between = 2
)

conmat <- feature_weighted_spatial_constraints(
  coords, fmats,
  alpha_within = 0.3, alpha_between = 0.7,
  maxk_between = 2, maxk_within = 2,
  sigma_between = 2, nnk_between = 4
)
```

`find_nn` *Find nearest neighbors*

Description

Find nearest neighbors

Usage

```
find_nn(x, ...)
```

Arguments

`x`	An object of class "nnsearcher".
`...`	Further arguments passed to or from other methods.

Value

A nearest neighbors result object.

Examples

```
X <- matrix(rnorm(20), nrow=5)
nn <- nnsearcher(X)
find_nn(nn, k=2)
```

find_nn.nnsearcher *Find Nearest Neighbors Using nnsearcher*

Description

Search for the k nearest neighbors using a pre-built nnsearcher object.

Usage

```
## S3 method for class 'nnsearcher'
find_nn(x, query = NULL, k = 5, ...)
```

Arguments

x	An object of class "nnsearcher".
query	A matrix of query points. If NULL, searches within the original data.
k	The number of nearest neighbors to find.
...	Additional arguments (currently unused).

Value

An object of class "nn_search" containing indices, distances, and labels.

Examples

```
X <- matrix(rnorm(100), nrow=10, ncol=10)
searcher <- nnsearcher(X)
result <- find_nn(searcher, k=3)
```

find_nn_among *Find nearest neighbors among a subset*

Description

Find nearest neighbors among a subset

Usage

```
find_nn_among(x, ...)
```

Arguments

x	An object of class "nnsearcher" or "class_graph".
...	Further arguments passed to or from other methods.

Value

A nearest neighbors result object.

Examples

```
X <- matrix(rnorm(20), nrow=5)
nn <- nnsearcher(X)
find_nn_among(nn, k=2, idx=1:3)
```

`find_nn_among.class_graph` *Find Nearest Neighbors Among Classes*

Description

Find the nearest neighbors within each class for a class_graph object.

Usage

```
## S3 method for class 'class_graph'
find_nn_among(x, X, k = 5, ...)
```

Arguments

`x`	A class_graph object.
`X`	The data matrix corresponding to the graph nodes.
`k`	The number of nearest neighbors to find.
`...`	Additional arguments (currently unused).

Value

A search result object containing indices, distances, and labels.

Examples

```
labs <- factor(c("a","a","b","b"))
cg <- class_graph(labs)
X <- matrix(rnorm(12), nrow=4)
find_nn_among(cg, X, k=1)
```

find_nn_among.nnsearcher *Find Nearest Neighbors Among Subset Using nnsearcher*

Description

Search for the k nearest neighbors within a specified subset of points.

Usage

```
## S3 method for class 'nnsearcher'
find_nn_among(x, k = 5, idx, ...)
```

Arguments

x	An object of class "nnsearcher".
k	The number of nearest neighbors to find.
idx	A numeric vector specifying the subset of point indices to search among.
...	Additional arguments (currently unused).

Value

An object of class "nn_search" containing indices, distances, and labels.

Examples

```
X <- matrix(rnorm(20), nrow=5)
searcher <- nnsearcher(X)
find_nn_among(searcher, k=2, idx=1:3)
```

find_nn_between *Find nearest neighbors between two sets of data points*

Description

Find nearest neighbors between two sets of data points

Usage

```
find_nn_between(x, ...)
```

Arguments

x	An object of class "nnsearcher".
...	Further arguments passed to or from other methods.

Value

A nearest neighbors result object.

Examples

```
X <- matrix(rnorm(20), nrow=5)
nn <- nnsearcher(X)
find_nn_between(nn, k=1, idx1=1:2, idx2=3:5)
```

---

find_nn_between.nnsearcher

Find Nearest Neighbors Between Two Sets Using nnsearcher

Description

Search for the k nearest neighbors from one set of points to another set.

Usage

```
## S3 method for class 'nnsearcher'
find_nn_between(x, k = 5, idx1, idx2, restricted = FALSE, ...)
```

Arguments

x	An object of class "nnsearcher".
k	The number of nearest neighbors to find.
idx1	A numeric vector specifying indices of the first set of points.
idx2	A numeric vector specifying indices of the second set of points.
restricted	Logical; if TRUE, use restricted search mode.
...	Additional arguments (currently unused).

Value

An object of class "nn_search" containing indices, distances, and labels.

Examples

```
X <- matrix(rnorm(40), nrow=10)
searcher <- nnsearcher(X)
find_nn_between(searcher, k=2, idx1=1:5, idx2=6:10)
```

graph_weights *Convert a Data Matrix to an Adjacency Graph*

Description

Convert a data matrix with n instances and p features to an n-by-n adjacency graph using specified neighbor and weight modes.

Usage

```
graph_weights(
  X,
  k = 5,
  neighbor_mode = c("knn"),
 weight_mode = c("heat", "normalized", "binary", "euclidean", "cosine", "correlation"),
  type = c("normal", "mutual", "asym"),
  sigma = NULL,
  eps = NULL,
  labels = NULL,
  ...
)
```

Arguments

X	A data matrix where each row represents an instance and each column represents a variable. Similarity is computed over instances.
k	An integer representing the number of neighbors (ignored when neighbor_mode is not 'epsilon').
neighbor_mode	A character string specifying the method for assigning weights to neighbors, either "supervised", "knn", "knearest_misses", or "epsilon".
weight_mode	A character string specifying the weight mode: binary (1 if neighbor, 0 otherwise), 'heat', 'normalized', 'euclidean', 'cosine', or 'correlation'.
type	A character string specifying the nearest neighbor policy, one of: normal, mutual, asym.
sigma	A numeric parameter for the heat kernel (exp(-dist/(2*sigma^2))).
eps	A numeric value representing the neighborhood radius when neighbor_mode is 'epsilon' (not implemented).
labels	A factor vector representing the class of the categories when weight_mode is 'supervised' with nrow(labels) equal to nrow(X).
...	Additional parameters passed to the internal functions.

Details

This function converts a data matrix with n instances and p features into an adjacency graph. The adjacency graph is created based on the specified neighbor and weight modes. The neighbor mode determines how neighbors are assigned weights, and the weight mode defines the method used to compute weights.

Distances passed to ‘weight_mode‘ kernels are Euclidean (square root already applied to Rnanoflann outputs). Custom kernels should be written accordingly; if a kernel expects squared distances, wrap it to square its input.

Value

An adjacency graph based on the specified neighbor and weight modes.

See Also

graph_weights_fast for additional backends and self-tuning options

Examples

```
X <- matrix(rnorm(100*100), 100, 100)
sm <- graph_weights(X, neighbor_mode="knn", weight_mode="normalized", k=3)

labels <- factor(rep(letters[1:4], 5))
sm3 <- graph_weights(X, neighbor_mode="knn", k=3, labels=labels, weight_mode="cosine")
sm4 <- graph_weights(X, neighbor_mode="knn", k=100, labels=labels, weight_mode="cosine")
```

---

graph_weights_fast *Fast kNN Graph Weights*

Description

Construct a sparse k-nearest-neighbor (kNN) graph quickly. The default backend is ‘Rnanoflann‘, which performs exact Euclidean search. Set ‘backend = "hnsw"‘ to opt in to approximate HNSW search via ‘RcppHNSW‘.

Usage

```
graph_weights_fast(
  X,
  k = 15,
 weight_mode = c("self_tuned", "heat", "normalized", "binary", "euclidean", "cosine",
    "correlation"),
  type = c("normal", "mutual", "asym"),
  backend = c("nanoflann", "hnsw", "auto"),
  sigma = NULL,
  local_k = min(7L, k),
  M = 16,
  ef = 200
)
```

Arguments

X	A numeric matrix with rows as observations and columns as features.
k	Number of nearest neighbors (per row).
weight_mode	Weighting to convert distances into edge weights. One of ‘"self_tuned"‘, ‘"heat"‘, ‘"normalized"‘, ‘"binary"‘, ‘"euclidean"‘, ‘"cosine"‘, or ‘"correlation"‘.
type	Symmetrization policy. ‘"normal"‘ returns a union graph with ‘max(w_ij, w_ji)‘; ‘"mutual"‘ returns an intersection graph with ‘min(w_ij, w_ji)‘; ‘"asym"‘ returns the directed kNN graph.
backend	Neighbor search backend: ‘"nanoflann"‘ (default, exact Euclidean search), ‘"hnsw"‘ (approximate search via ‘RcppHNSW‘), or ‘"auto"‘ (currently resolves to ‘"nanoflann"‘ to avoid implicit approximate results).
sigma	Bandwidth for ‘"heat"‘/‘"normalized"‘ modes. If ‘NULL‘, a robust value is estimated from kNN distances.
local_k	Local neighborhood size used by ‘"self_tuned"‘; defaults to ‘min(7, k)‘.
M, ef	HNSW parameters (only used when ‘backend = "hnsw"‘). Larger ‘ef‘ usually improves recall at the cost of runtime.

Details

The default weighting mode (‘"self_tuned"‘) uses a self-tuning heat kernel based on per-point local scale, which tends to work well across varying densities.

Provides additional neighbor search backends (HNSW, nanoflann) and self-tuning sigma estimation not available in graph_weights. Approximate search is never selected unless ‘backend = "hnsw"‘ is requested.

Value

A sparse ‘dgCMatrix‘ adjacency matrix.

See Also

graph_weights for the standard interface

Examples

```
X <- matrix(rnorm(200), 20, 10)
W <- graph_weights_fast(X, k = 5)
```

heat_kernel *Compute the Heat Kernel*

Description

This function computes the heat kernel, which is a radial basis function that can be used for smoothing, interpolation, and approximation tasks. The heat kernel is defined as exp(-x^2/(2*sigma^2)), where x is the distance and sigma is the bandwidth. It acts as a similarity measure for points in a space, assigning high values for close points and low values for distant points.

Usage

```
heat_kernel(x, sigma = 1)
```

Arguments

x	A numeric vector or matrix representing the distances between data points.
sigma	The bandwidth of the heat kernel, a positive scalar value. Default is 1.

Value

A numeric vector or matrix with the same dimensions as the input ‘x‘, containing the computed heat kernel values.

Details

The heat kernel is widely used in various applications, including machine learning, computer graphics, and image processing. It can be employed in kernel methods, such as kernel PCA, Gaussian process regression, and support vector machines, to capture the local structure of the data. The heat kernel's behavior is controlled by the bandwidth parameter sigma, which determines the smoothness of the resulting function.

Examples

```
x <- seq(-3, 3, length.out = 100)
y <- heat_kernel(x, sigma = 1)
plot(x, y, type = "l", main = "Heat Kernel")
```

helmert_contrasts *Build Helmert orthonormal contrasts*

Description

Creates orthonormal Helmert contrast matrices for a set of factors, suitable for use with effect-coded kernels. The resulting contrasts are orthonormal, which can improve numerical stability.

Usage

```
helmert_contrasts(Ls)
```

Arguments

Ls	Named integer vector specifying the number of levels per factor

Value

Named list of orthonormal contrast matrices, each of dimension L x (L-1) where L is the number of levels for that factor. Each matrix has orthonormal columns.

Examples

```
contrasts <- helmert_contrasts(c(A=3, B=4))
print(dim(contrasts$A))  # 3 x 2
print(dim(contrasts$B))  # 4 x 3
```

heterogeneous_neighbors *Heterogeneous Neighbors for class_graph Objects*

Description

Compute the neighbors between different classes for a class_graph object.

Usage

```
heterogeneous_neighbors(x, X, k, weight_mode = "heat", sigma = 1, ...)
```

Arguments

x	A class_graph object.
X	The data matrix corresponding to the graph nodes.
k	The number of nearest neighbors to find.
weight_mode	Method for weighting edges (e.g., "heat", "binary", "euclidean").
sigma	Scaling factor for heat kernel if ‘weight_mode="heat"‘.
...	Additional arguments passed to weight function.

Value

A neighbor_graph object representing the between-class neighbors.

Examples

```
labs <- factor(c("a","a","b","b"))
cg <- class_graph(labs)
X <- matrix(rnorm(8), ncol=2)
heterogeneous_neighbors(cg, X, k=1)
```

---

homogeneous_neighbors *Homogeneous Neighbors for class_graph Objects*

Description

Compute the neighbors within the same class for a class_graph object.

Usage

```
homogeneous_neighbors(x, X, k, weight_mode = "heat", sigma = 1, ...)
```

Arguments

x	A class_graph object.
X	The data matrix corresponding to the graph nodes.
k	The number of nearest neighbors to find.
weight_mode	Method for weighting edges (e.g., "heat", "binary", "euclidean").
sigma	Scaling factor for heat kernel if ‘weight_mode="heat"‘.
...	Additional arguments passed to weight function.

Value

A neighbor_graph object representing the within-class neighbors.

Examples

```
labs <- factor(c("a","a","b","b"))
cg <- class_graph(labs)
X <- matrix(rnorm(8), ncol=2)
homogeneous_neighbors(cg, X, k=1)
```

inverse_heat_kernel *inverse_heat_kernel*

Description

inverse_heat_kernel

Usage

```
inverse_heat_kernel(x, sigma = 1)
```

Arguments

x	the distances
sigma	the bandwidth

Value

Numeric vector/matrix of inverse heat kernel values.

Examples

```
inverse_heat_kernel(c(1, 2), sigma = 1)
```

kernel_alignment *Kernel alignment score*

Description

Computes the normalized Frobenius inner product between two symmetric matrices, which measures the similarity between two kernels. Useful for model selection and comparing different kernel parameterizations.

Usage

```
kernel_alignment(A, B)
```

Arguments

A	First symmetric matrix
B	Second symmetric matrix (must have same dimensions as A)

Details

The kernel alignment is computed as: align(A,B) = <A,B>_F / (||A||_F * ||B||_F), where <A,B>_F = sum(A * B) is the Frobenius inner product and ||A||_F is the Frobenius norm. This provides a normalized measure of kernel similarity.

Value

Scalar alignment score in [-1,1], where 1 indicates perfect positive alignment, 0 indicates orthogonality, and -1 indicates perfect negative alignment

Examples

```
A <- matrix(c(2, 1, 0, 1, 2, 1, 0, 1, 2), 3, 3)
B <- matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1), 3, 3)
align_score <- kernel_alignment(A, B)
print(align_score)  # alignment between A and identity matrix
```

---

kernel_roots *Square root and inverse square root of a PSD kernel*

Description

Computes the matrix square root and inverse square root of a positive semi-definite kernel using eigendecomposition with optional regularization.

Usage

```
kernel_roots(K, jitter = 1e-10)
```

Arguments

K	Symmetric positive semi-definite matrix
jitter	Small ridge parameter to ensure numerical stability (default 1e-10)

Details

For a positive semi-definite matrix K, this function computes K^(1/2) and K^(-1/2) using the eigendecomposition K = V * diag(lambda) * V', where K^(1/2) = V * diag(sqrt(lambda)) * V' and K^(-1/2) = V * diag(1/sqrt(lambda)) * V'. The jitter parameter adds a small ridge to eigenvalues to prevent numerical issues with near-zero eigenvalues.

Value

A list containing:

Khalf	The matrix square root K^(1/2) with same dimensions as K
Kihalf	The matrix inverse square root K^(-1/2) with same dimensions as K
evals	Eigenvalues (after jitter adjustment)
evecs	Eigenvectors matrix

Examples

```
K <- matrix(c(2, 1, 1, 2), 2, 2)
result <- kernel_roots(K)
print(dim(result$Khalf))  # 2x2
```

laplacian *Compute Graph Laplacian of a Weight Matrix*

Description

This function computes the graph Laplacian of a given weight matrix. The graph Laplacian is defined as the difference between the degree matrix and the adjacency matrix.

Usage

```
laplacian(x, ...)
```

Arguments

x	The weight matrix representing the graph structure.
...	Additional arguments to be passed to specific implementations of the laplacian method.

Value

The graph Laplacian matrix of the given weight matrix.

Examples

```
W <- Matrix::Matrix(c(0,1,0,
              1,0,1,
              0,1,0), nrow=3, byrow=TRUE, sparse=TRUE)
ng <- neighbor_graph(W)
laplacian(ng)
```

laplacian.neighbor_graph *Compute Laplacian matrix for neighbor_graph object*

Description

Compute the Laplacian matrix of a neighbor_graph object.

Usage

```
## S3 method for class 'neighbor_graph'
laplacian(x, normalized = FALSE, ...)
```

Arguments

x	A neighbor_graph object.
normalized	A logical value indicating whether the normalized Laplacian should be computed (default: FALSE).
...	Additional arguments (currently ignored).

Details

The unnormalized Laplacian is computed as L = D - A where D is the degree matrix and A is the adjacency matrix. The normalized Laplacian is computed as L_sym = I - D^(-1/2) A D^(-1/2).

Value

A sparse Matrix object representing the Laplacian matrix.

Examples

```
adj_matrix <- Matrix::Matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0),
                             nrow = 3, byrow = TRUE, sparse = TRUE)
ng <- neighbor_graph(adj_matrix)
L <- laplacian(ng)
L_norm <- laplacian(ng, normalized = TRUE)
```

---

local_global_adjacency

Local + Global KNN Adjacency

Description

Build an adjacency matrix that mixes L neighbors inside a local radius r with K neighbors outside that radius. Far neighbors receive a mild penalty so they can contribute without dominating.

Usage

```
local_global_adjacency(
  coord_mat,
  L = 5,
  K = 5,
  r,
  weight_mode = c("heat", "binary"),
  sigma = r/2,
  far_penalty = c("lambda", "exp"),
  lambda = 0.6,
  tau = r,
  nnk_buffer = 10,
  include_diagonal = FALSE,
```

```
  symmetric = TRUE,
  normalized = FALSE
)
```

Arguments

coord_mat	Numeric matrix of coordinates (rows = points).
L	Number of local neighbors (within r) to keep for each point.
K	Number of far neighbors (outside r) to keep for each point.
r	Radius defining the local ball.
weight_mode	Weighting scheme name (e.g., "heat"); forwarded to internal helper get_neighbor_fun.
sigma	Bandwidth for the heat/normalized kernels; default r/2.
far_penalty	Either "lambda" (constant multiplier) or "exp" (decay with distance beyond r).
lambda	Constant multiplier for far neighbors when far_penalty = "lambda".
tau	Scale of exponential decay when far_penalty = "exp".
nnk_buffer	Extra candidates requested from the NN search to ensure enough far neighbors are available.
include_diagonal	Logical; keep self-loops.
symmetric	Logical; if TRUE, symmetrize by averaging A and t(A).
normalized	Logical; if TRUE, row-normalize the matrix (stochastic).

Value

A sparse adjacency matrix mixing local and far neighbors.

Examples

```
set.seed(1)
coords <- matrix(runif(200), ncol = 2)
A <- local_global_adjacency(coords, L = 4, K = 3, r = 0.15,
                            weight_mode = "heat", lambda = 0.7)
Matrix::rowSums(A)[1:5]
```

---

make_doubly_stochastic

Compute the doubly stochastic matrix from a given matrix

Description

This function iteratively computes the doubly stochastic matrix from a given input matrix. A doubly stochastic matrix is a matrix in which both row and column elements sum to 1.

Usage

```
make_doubly_stochastic(A, iter = 30, tol = 1e-06)
```

Arguments

A	A numeric matrix for which to compute the doubly stochastic matrix
iter	Integer, the number of iterations to perform (default is 30)
tol	Numeric convergence tolerance; iteration stops when max row-sum deviation from 1 is below this value (default 1e-6)

Value

A numeric matrix representing the computed doubly stochastic matrix

Examples

```
A <- matrix(c(2, 4, 6, 8, 10, 12), nrow = 3, ncol = 2)

result <- make_doubly_stochastic(A, iter = 30)
```

---

nclasses *Number of Classes*

Description

A generic function to compute the number of classes in a graph.

Usage

```
nclasses(x)
```

Arguments

x	An object.

Value

The number of classes in the input object.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
nclasses(cg)
```

nclasses.class_graph *Number of Classes for class_graph Objects*

Description

Compute the number of classes in a class_graph object.

Usage

```
## S3 method for class 'class_graph'
nclasses(x)
```

Arguments

x	A class_graph object.

Value

The number of classes in the class_graph.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
nclasses(cg)
```

neighbors *Neighbors of a Set of Nodes*

Description

This function retrieves the indices of neighbors of one or more vertices in a given graph or graph-like object.

Usage

```
neighbors(x, i, ...)
```

Arguments

x	The graph or graph-like object in which to find the neighbors.
i	The vertex or vertices for which to find the neighbors. Can be a single vertex index or a vector of vertex indices.
...	Additional arguments to be passed to specific implementations of the neighbors method.

Value

A list of vertex indices representing the neighbors of the specified vertices. The length of the list is equal to the number of input vertices, and each element in the list contains the neighbor indices for the corresponding input vertex.

Examples

```
g <- neighbor_graph(igraph::make_ring(5))

n <- adjoin::neighbors(g, 1)
```

`neighbors.neighbor_graph`

Get neighbors for neighbor_graph object

Description

Retrieve the neighbors of a specific node or all nodes in a neighbor_graph object.

Usage

```
## S3 method for class 'neighbor_graph'
neighbors(x, i, ...)
```

Arguments

`x`	A neighbor_graph object.
`i`	An integer specifying the index of the node for which neighbors should be retrieved. If missing, returns neighbors for all nodes.
`...`	Additional arguments (currently ignored).

Value

If i is provided, a list containing the neighbors of node i. If i is missing, a list with neighbors for all nodes.

Examples

```
adj_matrix <- Matrix::Matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0),
                             nrow = 3, byrow = TRUE, sparse = TRUE)
ng <- neighbor_graph(adj_matrix)
neighbors(ng, 1)  # Neighbors of node 1
neighbors(ng)     # All neighbors
```

neighbor_graph *Neighbor Graph*

Description

A generic function to create a neighbor_graph object.

Create a neighbor_graph object from an igraph object or Matrix object.

Usage

```
neighbor_graph(x, ...)

## S3 method for class 'igraph'
neighbor_graph(x, params = list(), type = NULL, classes = NULL, ...)

## S3 method for class 'Matrix'
neighbor_graph(x, params = list(), type = NULL, classes = NULL, ...)

## S3 method for class 'matrix'
neighbor_graph(x, params = list(), type = NULL, classes = NULL, ...)
```

Arguments

x	An igraph or Matrix object.
...	Additional arguments.
params	A list of parameters (default: empty list).
type	A character string specifying the type of graph (currently unused).
classes	A character vector specifying additional classes for the object.

Value

A neighbor_graph object, the structure of which depends on the input object's class.

A neighbor_graph object wrapping the input graph structure.

Examples

```
library(igraph)
g <- make_ring(5)
ng1 <- neighbor_graph(g)

adj_matrix <- Matrix::Matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0),
                             nrow = 3, byrow = TRUE, sparse = TRUE)
ng2 <- neighbor_graph(adj_matrix)
```

`neighbor_graph.nnsearcher` *Create Neighbor Graph from nnsearcher Object*

Description

Construct a neighbor graph from nearest neighbor search results.

Usage

```
## S3 method for class 'nnsearcher'
neighbor_graph(
  x,
  query = NULL,
  k = 5,
  type = c("normal", "asym", "mutual"),
 transform = c("heat", "binary", "euclidean", "normalized", "cosine", "correlation"),
  sigma = 1,
  ...
)
```

Arguments

`x`	An object of class "nnsearcher".
`query`	A matrix of query points. If NULL, uses original data.
`k`	The number of nearest neighbors to find.
`type`	The type of graph construction method.
`transform`	The transformation method for converting distances to weights.
`sigma`	The bandwidth parameter for the transformation.
`...`	Additional arguments (currently unused).

Value

A neighbor_graph object representing the constructed graph.

Examples

```
X <- matrix(rnorm(20), nrow=5)
searcher <- nnsearcher(X)
neighbor_graph(searcher, k=2, type="normal", transform="heat", sigma=1)
```

`nnsearcher` *Nearest Neighbor Searcher*

Description

Create a nearest neighbor searcher object for efficient nearest neighbor search. Uses ‘Rnanoflann‘ for exact Euclidean search by default. Uses ‘RcppHNSW‘ for approximate search only when cosine or inner-product distances are requested.

Usage

```
nnsearcher(
  X,
  labels = 1:nrow(X),
  ...,
  distance = c("l2", "euclidean", "cosine", "ip"),
  M = 16,
  ef = 200
)
```

Arguments

`X`	A numeric matrix where each row represents a data point.
`labels`	A vector of labels corresponding to each row in X. Defaults to row indices.
`...`	Additional arguments (currently unused).
`distance`	The distance metric to use. One of "l2", "euclidean", "cosine", or "ip". Note: "cosine" and "ip" require the RcppHNSW package.
`M`	The maximum number of connections for HNSW (only used with cosine/ip).
`ef`	The size of the dynamic candidate list for HNSW (only used with cosine/ip). Larger values usually improve recall at the cost of runtime.

Value

An object of class "nnsearcher" containing the data matrix, labels, search index, and search parameters.

Examples

```
X <- matrix(rnorm(100), nrow=10, ncol=10)
searcher <- nnsearcher(X)
```

node_density *Node Density*

Description

Compute the local density around each node in a graph.

Usage

```
node_density(x, ...)
```

Arguments

x	A graph-like object.
...	Additional arguments passed to specific methods.

Value

A numeric vector of node densities.

Examples

```
adj_matrix <- matrix(c(0,1,1,
                       1,0,0,
                       1,0,0), nrow=3, byrow=TRUE)
ng <- neighbor_graph(adj_matrix)
node_density(ng, matrix(rnorm(9), nrow=3))
```

node_density.neighbor_graph *Compute node density for neighbor_graph object*

Description

Compute the node density for a neighbor_graph object based on local neighborhoods.

Usage

```
## S3 method for class 'neighbor_graph'
node_density(x, X, ...)
```

Arguments

x	A neighbor_graph object.
X	A data matrix containing the data points, with rows as observations.
...	Additional arguments (currently ignored).

Details

Node density is computed as the average squared distance from each node to its neighbors, normalized by the square of the neighborhood size.

Value

A numeric vector containing the node densities.

Examples

```
X <- matrix(rnorm(30), nrow = 10, ncol = 3)
adj_matrix <- Matrix::Matrix(diag(10), sparse = TRUE)
ng <- neighbor_graph(adj_matrix)
densities <- node_density(ng, X)
```

---

non_neighbors *Get Indices of Non-neighbors of a Node*

Description

Retrieve the indices of nodes that are not neighbors of a specified node.

Usage

```
non_neighbors(x, ...)
```

Arguments

x	A neighbor graph object.
...	Additional arguments passed to specific methods.

Value

A numeric vector of node indices that are not neighbors of the given node.

Examples

```
adj_matrix <- matrix(c(0,1,0,
                       1,0,0,
                       0,0,0), nrow=3, byrow=TRUE)
ng <- neighbor_graph(adj_matrix)
non_neighbors(ng, 1)
```

non_neighbors.neighbor_graph

Get non-neighbors for neighbor_graph object

Description

Retrieve the non-neighboring nodes of a given node in a neighbor_graph object.

Usage

```
## S3 method for class 'neighbor_graph'
non_neighbors(x, i, ...)
```

Arguments

x	A neighbor_graph object.
i	The index of the node for which non-neighboring nodes will be returned.
...	Additional arguments (currently ignored).

Value

A numeric vector of node indices that are not neighbors of the given node (excluding the node itself).

Examples

```
adj_matrix <- Matrix::Matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0),
                             nrow = 3, byrow = TRUE, sparse = TRUE)
ng <- neighbor_graph(adj_matrix)
```

normalized_heat_kernel

normalized_heat_kernel

Description

normalized_heat_kernel

Usage

```
normalized_heat_kernel(x, sigma = 0.68, len)
```

Arguments

x	the distances
sigma	the bandwidth
len	the normalization factor (e.g. the length of the feature vectors)

Value

Numeric vector/matrix of normalized heat kernel values.

Examples

```
normalized_heat_kernel(c(1,2), sigma = .5, len = 4)
```

---

normalize_adjacency *Normalize Adjacency Matrix*

Description

This function normalizes an adjacency matrix by dividing each element by the product of the square root of the corresponding row and column sums. Optionally, it can also symmetrize the normalized matrix by averaging it with its transpose.

Usage

```
normalize_adjacency(
  sm,
  symmetric = TRUE,
  handle_isolates = c("self_loop", "keep_zero", "drop")
)
```

Arguments

sm	A sparse adjacency matrix representing the graph.
symmetric	A logical value indicating whether to symmetrize the matrix after normalization (default: TRUE).
handle_isolates	How to treat zero-degree nodes when normalizing: "self_loop" adds a self-loop (default), "keep_zero" leaves them as zero, or "drop" removes them from the matrix.

Value

A normalized and, if requested, symmetrized adjacency matrix.

Examples

```
set.seed(123)
A <- matrix(runif(100), 10, 10)
A_normalized <- normalize_adjacency(A)
```

`nvertices` *Number of Vertices in Graph-like Objects*

Description

Retrieve the number of vertices in a neighbor_graph object.

Usage

```
nvertices(x, ...)
```

Arguments

`x`	an object with a neighborhood
`...`	Additional arguments (currently ignored).

Value

The number of vertices in the neighbor_graph object.

Examples

```
adj_matrix <- matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0), nrow = 3, byrow = TRUE)
ng <- neighbor_graph(adj_matrix)
nvertices(ng) # Should return 3
```

`nvertices.neighbor_graph`
Get number of vertices in neighbor_graph object

Description

Get the number of vertices in a neighbor_graph object.

Usage

```
## S3 method for class 'neighbor_graph'
nvertices(x, ...)
```

Arguments

x	A neighbor_graph object.
...	Additional arguments (currently ignored).

Value

An integer representing the number of vertices in the graph.

Examples

```
adj_matrix <- Matrix::Matrix(c(0, 1, 1, 0, 1, 0, 1, 0, 0),
                             nrow = 3, byrow = TRUE, sparse = TRUE)
ng <- neighbor_graph(adj_matrix)
nvertices(ng)  # Should return 3
```

---

pairwise_adjacency *Compute a pairwise adjacency matrix for multiple graphs*

Description

This function computes a pairwise adjacency matrix for multiple graphs with a given set of spatial coordinates and feature vectors. The function takes two user-defined functions to compute within-graph and between-graph similarity measures.

Usage

```
pairwise_adjacency(Xcoords, Xfeats, fself, fbetween)
```

Arguments

Xcoords	A list of numeric matrices or data.frames containing the spatial coordinates of the nodes of each graph.
Xfeats	A list of numeric matrices or data.frames containing the feature vectors for the nodes of each graph.
fself	A function that computes similarity for nodes within the same graph (e.g., Xi_1, Xi_2).
fbetween	A function that computes similarity for nodes across graphs (e.g., Xi_1, Xj_1).

Value

A sparse matrix representing the pairwise adjacency matrix for the input graphs.

Examples

```
coords <- list(matrix(c(0,0,1,0), ncol=2, byrow=TRUE),
               matrix(c(0,1,1,1), ncol=2, byrow=TRUE))
feats  <- list(matrix(rnorm(2), ncol=1),
               matrix(rnorm(2), ncol=1))
fself <- function(c,f) spatial_adjacency(c, normalized=FALSE, include_diagonal=FALSE)
fbetween <- function(c1,c2,f1,f2) cross_spatial_adjacency(c1,c2, normalized=FALSE)
M <- pairwise_adjacency(coords, feats, fself, fbetween)
dim(M)
```

---

print.repulsion_graph *Print method for repulsion_graph objects*

Description

Print method for repulsion_graph objects

Usage

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

Arguments

x	A repulsion_graph object
...	Additional arguments passed to print

Value

The input x, invisibly.

Examples

```
coords <- matrix(c(0,0,1,0), ncol=2, byrow=TRUE)
W <- neighbor_graph(spatial_adjacency(coords, nnk=2, sigma=1))
cg <- class_graph(factor(c(1,2)))
rg <- repulsion_graph(W, cg)
print(rg)
```

psparse *Apply a Function to Non-Zero Elements in a Sparse Matrix*

Description

This function applies a specified function (e.g., max) to each pair of non-zero elements in a sparse matrix. It can return the result as a triplet representation or a sparse matrix.

Usage

```
psparse(M, FUN, return_triplet = FALSE)
```

Arguments

M	A sparse matrix object from the Matrix package.
FUN	A function to apply to each pair of non-zero elements in the sparse matrix.
return_triplet	A logical value indicating whether to return the result as a triplet representation. Default is FALSE.

Value

If return_triplet is TRUE, a matrix containing the i, j, and x values in the triplet format; otherwise, a sparse matrix with the updated values.

Examples

```
library(Matrix)
M <- sparseMatrix(i = c(1, 3, 1), j = c(2, 3, 3), x = c(1, 2, 3))
psparse_max <- psparse(M, FUN = max)
psparse_sum_triplet <- psparse(M, FUN = `+`, return_triplet = TRUE)
```

repulsion_graph *Create a Repulsion Graph*

Description

Constructs a "repulsion graph" derived from an input graph ‘W‘ and a class structure graph ‘cg‘. The resulting graph retains only the edges from ‘W‘ that connect nodes belonging to *different* classes according to ‘cg‘. Edges connecting nodes within the same class are removed (or "repulsed").

Usage

```
repulsion_graph(
  W,
  cg,
  method = c("weighted", "binary"),
  threshold = 0,
  norm_fac = 1
)
```

Arguments

W	An input graph object. Can be a 'Matrix' object (e.g., 'dgCMatrix') representing the adjacency matrix, or a 'neighbor_graph' object. Edge weights are used if 'method="weighted"'.
cg	A 'class_graph' object defining the class membership of the nodes in 'W'. Must have the same dimensions as 'W'.
method	'character'. Specifies how to handle the weights of the remaining (between-class) edges: - '"weighted"' (default): Retains the original weights from 'W'. - '"binary"': Sets the weight of all remaining edges to 1.
threshold	'numeric'. A threshold applied to the input graph 'W' *before* filtering by class. Edges in 'W' with weights strictly below this value are discarded. Default is 0.
norm_fac	'numeric'. A normalization factor applied *only* if 'method = "weighted"'. The weights of the retained edges are divided by this factor. Default is 1 (no normalization).

Details

This function takes an existing graph 'W' (representing similarities, connections, etc.) and filters it based on class labels. The 'class_graph' object 'cg' provides a binary adjacency matrix where '1' indicates nodes belong to the *same* class. By taking the complement ('!adjacency(cg)'), we get a mask where '1' indicates nodes belong to *different* classes. Element-wise multiplication ('W * !adjacency(cg)') effectively removes within-class edges from 'W'.

The 'method' argument controls whether the remaining edge weights are kept as is ('"weighted"') or converted to binary indicators ('"binary"').

This type of graph is useful when focusing on interactions *between* distinct groups or classes within a larger network or dataset.

Value

A 'repulsion_graph' object (inheriting from 'neighbor_graph'), representing the filtered graph containing only between-class edges.

Examples

```
library(Matrix)
```

```
set.seed(123)
N <- 50
X <- matrix(rnorm(N * 5), N, 5)
W_adj <- Matrix(rsparsematrix(N, N, 0.1, symmetric = TRUE)) # Base adjacency
diag(W_adj) <- 0
W_ng <- neighbor_graph(W_adj) # Convert to neighbor_graph

labels <- factor(sample(1:3, N, replace = TRUE))
cg <- class_graph(labels)

R_weighted <- repulsion_graph(W_ng, cg, method = "weighted")
print(R_weighted)
plot(R_weighted$G, vertex.color = labels, vertex.size=8, vertex.label=NA)
title("Weighted Repulsion Graph (Edges only between classes)")

R_binary <- repulsion_graph(W_ng, cg, method = "binary")
print(R_binary)

data(iris)
X_iris <- as.matrix(iris[, 1:4])
labels_iris <- iris[, 5]
cg_iris <- class_graph(labels_iris)

W_iris_knn <- graph_weights(X_iris, k = 5, weight_mode = "heat", sigma = 0.7)

R_iris <- repulsion_graph(W_iris_knn, cg_iris, method = "weighted")
print(R_iris)
plot(R_iris$G, vertex.color = as.numeric(labels_iris), vertex.size=5, vertex.label=NA)
title("Iris Repulsion Graph (k=5, heat weights)")
```

`search_result` *Search result for nearest neighbor search*

Description

Search result for nearest neighbor search

Usage

```
search_result(x, result)
```

Arguments

`x`	An object of class "nnsearcher".
`result`	The result from the nearest neighbor search.

Value

An object with the class "nn_search".

Examples

```
res <- list(idx = matrix(c(1L,2L), nrow=1),
            dist = matrix(c(0.1,0.2), nrow=1))
dummy <- nnsearcher(matrix(rnorm(4), nrow=2))
search_result(dummy, res)
```

`search_result.nnsearcher` *Convert Search Results for nnsearcher Objects*

Description

Convert raw search results to a standardized format for nnsearcher objects.

Usage

```
## S3 method for class 'nnsearcher'
search_result(x, result)
```

Arguments

`x`	An object of class "nnsearcher".
`result`	A raw result object from nearest neighbor search.

Value

An object of class "nn_search" with standardized field names.

Examples

```
res <- list(idx = matrix(c(1L,2L), nrow=1),
            dist = matrix(c(0.1,0.2), nrow=1))
dummy <- nnsearcher(matrix(rnorm(4), nrow=2))
search_result(dummy, res)
```

`spatial_adjacency` *Compute the spatial adjacency matrix for a coordinate matrix*

Description

This function computes the spatial adjacency matrix for a given coordinate matrix using specified parameters. Adjacency is determined by distance threshold and the maximum number of neighbors.

Usage

```
spatial_adjacency(
  coord_mat,
  dthresh = sigma * 3,
  nnk = 27,
  weight_mode = c("binary", "heat"),
  sigma = 5,
  include_diagonal = TRUE,
  normalized = TRUE,
  stochastic = FALSE,
  handle_isolates = c("self_loop", "keep_zero", "drop")
)
```

Arguments

coord_mat	A numeric matrix representing the spatial coordinates
dthresh	Numeric, the distance threshold defining the radius of the neighborhood (default is sigma*3)
nnk	Integer, the maximum number of neighbors to include in each spatial neighborhood (default is 27)
weight_mode	Character, the mode for computing weights, either "binary" or "heat" (default is "binary")
sigma	Numeric, the bandwidth of the heat kernel if weight_mode == "heat" (default is 5)
include_diagonal	Logical, whether to assign 1 to diagonal elements (default is TRUE)
normalized	Logical, whether to make row elements sum to 1 (default is TRUE)
stochastic	Logical, whether to make column elements also sum to 1 (only relevant if normalized == TRUE) (default is FALSE)
handle_isolates	How to treat zero-degree nodes when normalizing: "self_loop" adds a self-loop (default), "keep_zero" leaves them as zero, or "drop" removes them from the matrix.

Value

A sparse symmetric matrix representing the computed spatial adjacency

Examples

```
coord_mat = as.matrix(expand.grid(x=1:6, y=1:6))
sa <- spatial_adjacency(coord_mat)
```

`spatial_autocor` *Compute a spatial autocorrelation matrix*

Description

This function computes a spatial autocorrelation matrix using a radius-based nearest neighbor search. The function leverages the mgcv package to fit a generalized additive model (GAM) to the data and constructs the autocorrelation matrix using the fitted model.

Usage

```
spatial_autocor(X, cds, radius = 8, nsamples = 1000, maxk = 64)
```

Arguments

`X`	A numeric matrix or data.frame, where each column represents a variable and each row represents an observation.
`cds`	A numeric matrix or data.frame of spatial coordinates (x, y, or more dimensions) with the same number of rows as X.
`radius`	A positive numeric value representing the search radius for the radius-based nearest neighbor search. Default is 8.
`nsamples`	A positive integer indicating the number of samples to be taken for fitting the GAM. Default is 1000.
`maxk`	Maximum number of neighbors to request from the NN search before radius filtering (prevents O(n^2) memory). Default 64.

Value

A sparse matrix representing the spatial autocorrelation matrix for the input data.

Examples

```
set.seed(1)
cds <- as.matrix(expand.grid(1:10, 1:10))
X <- matrix(rnorm(5*nrow(cds)), nrow=5, ncol=nrow(cds))
S <- spatial_autocor(X, cds, radius=5, nsamples=100, maxk=50)
dim(S)
```

spatial_constraints *Construct a Sparse Matrix of Spatial Constraints for Data Blocks*

Description

This function creates a sparse matrix of spatial constraints for a set of data blocks. The spatial constraints matrix is useful in applications like image segmentation, where spatial information is crucial for identifying different regions in the image.

Usage

```
spatial_constraints(
  coords,
  nblocks = 1,
  sigma_within = 5,
  sigma_between = 1,
  shrinkage_factor = 0.1,
  nnk_within = 27,
  nnk_between = 1,
  weight_mode_within = "heat",
  weight_mode_between = "binary",
  variable_weights = 1,
  verbose = FALSE
)
```

Arguments

coords	The spatial coordinates as a matrix with rows as objects and columns as dimensions; or as a list of matrices where each element of the list contains the coordinates for a block.
nblocks	The number of coordinate blocks. Default is 1. If ‘coords‘ is a list and ‘nblocks‘ is omitted, it is inferred from ‘length(coords)‘.
sigma_within	The bandwidth of the within-block smoother. Default is 5.
sigma_between	The bandwidth of the between-block smoother. Default is 1.
shrinkage_factor	The amount of shrinkage towards the spatial block average. Default is 0.1.
nnk_within	The maximum number of nearest neighbors for within-block smoother. Default is 27.
nnk_between	The maximum number of nearest neighbors for between-block smoother. Default is 1.
weight_mode_within	The within-block nearest neighbor weight mode ("heat" or "binary"). Default is "heat".
weight_mode_between	The between-block nearest neighbor weight mode ("heat" or "binary"). Default is "binary".

variable_weights
: A vector of per-variable weights. Default is 1.

verbose
: A boolean indicating whether to print progress messages. Default is FALSE.

Value

A sparse matrix representing the spatial constraints for the provided data blocks.

Details

The function computes within-block and between-block constraints based on the provided coordinates, bandwidths, and other input parameters. It then balances the within-block and between-block constraints using a shrinkage factor, and normalizes the resulting matrix by the first eigenvalue.

Examples

```
coords <- as.matrix(expand.grid(1:2, 1:2))
S <- spatial_constraints(coords, nblocks=1, sigma_within=1, nnk_within=3)
dim(S)
```

---

spatial_laplacian *Compute the spatial Laplacian matrix of a coordinate matrix*

Description

This function computes the spatial Laplacian matrix of a given coordinate matrix using specified parameters.

Usage

```
spatial_laplacian(
  coord_mat,
  dthresh = 1.42,
  nnk = 27,
  weight_mode = c("binary", "heat"),
  sigma = dthresh/2,
  normalized = TRUE,
  stochastic = FALSE,
  handle_isolates = c("self_loop", "keep_zero", "drop")
)
```

Arguments

coord_mat
: A numeric matrix representing coordinates

dthresh
: Numeric, the distance threshold for adjacency (default is 1.42)

nnk
: Integer, the number of nearest neighbors for adjacency (default is 27)

weight_mode	Character, the mode for computing weights, either "binary" or "heat" (default is "binary")
sigma	Numeric, the sigma parameter for the heat kernel (default is dthresh/2)
normalized	Logical, whether the adjacency matrix should be normalized (default is TRUE)
stochastic	Logical, whether the adjacency matrix should be stochastic (default is FALSE)
handle_isolates	How to treat zero-degree nodes when normalizing: "self_loop" (default), "keep_zero", or "drop".

Value

A sparse symmetric matrix representing the computed spatial Laplacian

Examples

```
coord_mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, ncol = 2)

result <- spatial_laplacian(coord_mat, dthresh = 1.42, nnk = 27, weight_mode = "binary")
```

---

spatial_lap_of_gauss *Spatial Laplacian of Gaussian for coordinates*

Description

This function computes the spatial Laplacian of Gaussian for a given coordinate matrix using a specified sigma value.

Usage

```
spatial_lap_of_gauss(coord_mat, sigma = 2)
```

Arguments

coord_mat	A numeric matrix representing coordinates
sigma	Numeric, the sigma parameter for the Gaussian smoother (default is 2)

Value

A sparse symmetric matrix representing the computed spatial Laplacian of Gaussian

Examples

```
coord_mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, ncol = 2)

result <- spatial_lap_of_gauss(coord_mat, sigma = 2)
```

spatial_smoother *Compute the spatial smoother matrix for a coordinate matrix*

Description

This function computes the spatial smoother matrix for a given coordinate matrix using specified parameters.

Usage

```
spatial_smoother(
  coord_mat,
  sigma = 5,
  nnk = 3^(ncol(coord_mat)),
  stochastic = TRUE,
  handle_isolates = c("self_loop", "keep_zero", "drop")
)
```

Arguments

coord_mat	A numeric matrix representing coordinates
sigma	Numeric, the sigma parameter for the Gaussian smoother (default is 5)
nnk	Integer, the number of nearest neighbors for adjacency (default is 3^(ncol(coord_mat)))
stochastic	Logical, whether the adjacency matrix should be doubly stochastic (default is TRUE)
handle_isolates	How to treat zero-degree nodes when normalizing: "self_loop" adds a self-loop (default), "keep_zero" leaves them as zero, or "drop" removes them from the matrix.

Value

A sparse matrix representing the computed spatial smoother. If stochastic=TRUE, the matrix is row-stochastic (rows sum to 1) but not symmetric. If stochastic=FALSE, the matrix is symmetric but not row-stochastic.

Examples

```
coord_mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 3, ncol = 2)

result <- spatial_smoother(coord_mat, sigma = 5, nnk = 3^(ncol(coord_mat)), stochastic = TRUE)
```

sum_contrasts *Build sum-to-zero contrasts*

Description

Creates sum-to-zero contrast matrices for a set of factors, suitable for use with effect-coded kernels.

Usage

```
sum_contrasts(Ls)
```

Arguments

Ls	Named integer vector specifying the number of levels per factor

Value

Named list of contrast matrices, each of dimension L x (L-1) where L is the number of levels for that factor. Each matrix has row sums of zero.

Examples

```
contrasts <- sum_contrasts(c(A=3, B=4))
print(dim(contrasts$A))  # 3 x 2
print(dim(contrasts$B))  # 4 x 3
```

temporal_adjacency *Compute the temporal adjacency matrix of a time series*

Description

This function computes the temporal adjacency matrix of a given time series using a specified weight mode, sigma, and window size.

Usage

```
temporal_adjacency(
  time,
  weight_mode = c("heat", "binary"),
  sigma = 1,
  window = 2
)
```

Arguments

time	A numeric vector representing a time series
weight_mode	Character, the mode for computing weights, either "heat" or "binary" (default is "heat")
sigma	Numeric, the sigma parameter for the heat kernel (default is 1)
window	Integer, the window size for computing adjacency (default is 2)

Value

A sparse symmetric matrix representing the computed temporal adjacency

Examples

```
time <- 1:10

result <- temporal_adjacency(time, weight_mode = "heat", sigma = 1, window = 2)
```

---

temporal_autocor *Compute the temporal autocorrelation of a matrix*

Description

This function computes the temporal autocorrelation of a given matrix using a specified window size and optionally inverts the correlation matrix.

Usage

```
temporal_autocor(X, window = 3, inverse = FALSE)
```

Arguments

X	A numeric matrix for which to compute the temporal autocorrelation
window	integer, the window size for computing the autocorrelation, must be between 1 and ncol(X) (default is 3)
inverse	logical, whether to compute the inverse of the correlation matrix (default is FALSE)

Value

A sparse symmetric matrix representing the computed temporal autocorrelation

Examples

```
X <- matrix(rnorm(50), nrow = 10, ncol = 5)

result <- temporal_autocor(X, window = 2)
```

temporal_laplacian *Compute the temporal Laplacian matrix of a time series*

Description

This function computes the temporal Laplacian matrix of a given time series using a specified weight mode, sigma, and window size.

Usage

```
temporal_laplacian(
  time,
  weight_mode = c("heat", "binary"),
  sigma = 1,
  window = 2
)
```

Arguments

time	A numeric vector representing a time series
weight_mode	Character, the mode for computing weights, either "heat" or "binary" (default is "heat")
sigma	Numeric, the sigma parameter for the heat kernel (default is 1)
window	Integer, the window size for computing adjacency (default is 2)

Value

A sparse symmetric matrix representing the computed temporal Laplacian

Examples

```
time <- 1:10

result <- temporal_laplacian(time, weight_mode = "heat", sigma = 1, window = 2)
```

threshold_adjacency *Threshold Adjacency*

Description

This function extracts the k-nearest neighbors from an existing adjacency matrix. It returns a new adjacency matrix containing only the specified number of nearest neighbors.

Usage

```
threshold_adjacency(A, k = 5, type = c("normal", "mutual"), ncores = 1)
```

Arguments

A	An adjacency matrix representing the graph.
k	An integer specifying the number of neighbors to consider (default: 5).
type	A character string indicating the type of k-nearest neighbors graph to compute. One of "normal" or "mutual" (default: "normal").
ncores	An integer specifying the number of cores to use for parallel computation (default: 1).

Value

A sparse adjacency matrix containing only the specified number of nearest neighbors.

Examples

```
A <- matrix(runif(100), 10, 10)
A_thresholded <- threshold_adjacency(A, k = 5)
```

---

weighted_factor_sim *Compute Weighted Similarity Matrix for Factors in a Data Frame*

Description

Calculate the weighted similarity matrix for a set of factors in a data frame.

Usage

```
weighted_factor_sim(des, wts = rep(1, ncol(des))/ncol(des))
```

Arguments

des	A data frame containing factors for which the weighted similarity matrix will be computed.
wts	A numeric vector of weights corresponding to the factors in the data frame. The default is equal weights for all factors.

Value

A weighted similarity matrix computed for the factors in the data frame.

Examples

```
des <- data.frame(
  var1 = factor(c("a", "b", "a", "b", "a")),
  var2 = factor(c("c", "c", "d", "d", "d"))
)

sim_default_weights <- weighted_factor_sim(des)

sim_custom_weights <- weighted_factor_sim(des, wts = c(0.7, 0.3))
```

---

weighted_knn *Weighted k-Nearest Neighbors*

Description

This function computes a weighted k-nearest neighbors graph or adjacency matrix from a data matrix. The function takes into account the Euclidean distance between instances and applies a kernel function to convert the distances into similarities.

Usage

```
weighted_knn(
  X,
  k = 5,
  FUN = heat_kernel,
  type = c("normal", "mutual", "asym"),
  as = c("igraph", "sparse"),
  backend = c("nanoflann", "hnsw"),
  M = 16,
  ef = 200,
  ...
)
```

Arguments

X	A data matrix where rows are instances and columns are features.
k	An integer specifying the number of nearest neighbors to consider (default: 5).
FUN	A kernel function used to convert Euclidean distances into similarities (default: heat_kernel).
type	A character string indicating the type of k-nearest neighbors graph to compute. One of "normal", "mutual", or "asym" (default: "normal").
as	A character string specifying the format of the output. One of "igraph" or "sparse" (default: "igraph").
backend	Nearest-neighbor backend. ‘"nanoflann"‘ uses exact Euclidean search; ‘"hnsw"‘ uses approximate search via ‘RcppHNSW‘.

M, ef	HNSW tuning parameters used only when ‘backend = "hnsw"‘. Larger ‘ef‘ usually improves recall at the cost of runtime.
...	Additional arguments passed to the nearest neighbor search function (Rnanoflann::nn).

Details

Distances passed to ‘FUN‘ are Euclidean distances. The default ‘backend = "nanoflann"‘ uses exact Euclidean search. Set ‘backend = "hnsw"‘ to opt in to approximate search via ‘RcppHNSW‘; squared L2 distances from ‘RcppHNSW‘ are converted back to Euclidean distances before weighting.

Value

If ’as’ is "igraph", an igraph object representing the weighted k-nearest neighbors graph. If ’as’ is "sparse", a sparse adjacency matrix.

Examples

```
X <- matrix(rnorm(10 * 10), 10, 10)
w <- weighted_knn(X, k = 5)
```

---

weighted_spatial_adjacency

Weighted Spatial Adjacency

Description

Constructs a spatial adjacency matrix, where weights are determined by a secondary feature matrix.

Usage

```
weighted_spatial_adjacency(
  coord_mat,
  feature_mat,
  wsigma = 0.73,
  alpha = 0.5,
  nnk = 27,
  weight_mode = c("binary", "heat"),
  sigma = 1,
  dthresh = sigma * 2.5,
  include_diagonal = TRUE,
  normalized = FALSE,
  stochastic = FALSE
)
```

Arguments

coord_mat	A matrix with the spatial coordinates of the data points, where each row represents a point and each column represents a coordinate dimension.
feature_mat	A matrix with the feature vectors of the data points, where each row represents a point and each column represents a feature dimension. The number of rows in feature_mat must be equal to the number of rows in coord_mat.
wsigma	The spatial weight scale (default: 0.73).
alpha	The mixing parameter between 0 and 1 (default: 0.5). A value of 0 results in a purely spatial adjacency matrix, while a value of 1 results in a purely feature-based adjacency matrix.
nnk	The number of nearest neighbors to consider (default: 27).
weight_mode	The mode to use for weighting the adjacency matrix, either "binary" or "heat" (default: "binary").
sigma	The bandwidth for heat kernel weights (default: 1).
dthresh	The distance threshold for nearest neighbors (default: sigma * 2.5).
include_diagonal	A logical value indicating whether to include diagonal elements in the adjacency matrix (default: TRUE).
normalized	A logical value indicating whether to normalize the adjacency matrix (default: FALSE).
stochastic	A logical value indicating whether to make the resulting adjacency matrix doubly stochastic (default: FALSE).

Value

A sparse adjacency matrix with weighted spatial relationships.

Examples

```
set.seed(123)
coord_mat <- as.matrix(expand.grid(x=1:9, y=1:9, z=1:9))
fmat <- matrix(rnorm(nrow(coord_mat) * 100), nrow(coord_mat), 100)
wsa1 <- weighted_spatial_adjacency(coord_mat, fmat, nnk=3,
                                   weight_mode="binary", alpha=1,
                                   stochastic=TRUE)
wsa2 <- weighted_spatial_adjacency(coord_mat, fmat, nnk=27,
                                   weight_mode="heat", alpha=0,
                                   stochastic=TRUE, sigma=2.5)
```

within_class_neighbors *Within-Class Neighbors*

Description

A generic function to compute the within-class neighbors of a graph.

Usage

```
within_class_neighbors(x, ng, ...)
```

Arguments

x	An object.
ng	A neighbor graph object.
...	Additional arguments passed to specific methods.

Value

An object representing the within-class neighbors of the input graph, the structure of which depends on the input object's class.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
ng <- neighbor_graph(diag(3))
within_class_neighbors(cg, ng)
```

within_class_neighbors.class_graph *Within-Class Neighbors for class_graph Objects*

Description

Compute the within-class neighbors of a class_graph object.

Usage

```
## S3 method for class 'class_graph'
within_class_neighbors(x, ng, ...)
```

Arguments

x	A class_graph object.
ng	A neighbor graph object.
...	Additional arguments (currently ignored).

Value

A neighbor_graph object representing the within-class neighbors of the input class_graph.

Examples

```
labs <- factor(c("a","a","b"))
cg <- class_graph(labs)
ng <- neighbor_graph(matrix(c(0,1,0,1,0,0,0,0,0),3))
within_class_neighbors(cg, ng)
```

Index