Package 'neurocluster'

August 18, 2025

Type Package

Title Spatially contrained clustering for neuroimaging data

Version 0.1.0

Author Bradley Buchsbaum

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

Description More about what it does (maybe more than one line)
Use four spaces when indenting paragraphs within the Description.

License What license is it under?

Encoding UTF-8

LazyData true

Imports Rcpp (>= 0.11.3), RcppParallel, Matrix, FNN, assertthat, clue, igraph, matrixStats, neighborweights, neuroim2, neurosurf, purrr

LinkingTo Rcpp, RcppParallel

RoxygenNote 7.3.2.9000

Suggests testthat

R topics documented:

`acsc` *Adaptive Correlation Superclustering (ACSC)*

Description

Clusters fMRI voxels into spatially-coherent groups based on temporal correlation and spatial proximity. Includes optional refinement for boundary corrections.

Usage

```
acsc(
  bvec,
  mask,
  block_size = 2,
  ann_k = 10,
  alpha = 0.5,
  correlation_metric = c("pearson", "spearman", "robust"),
  spatial_weighting = c("gaussian", "binary"),
  refine = TRUE,
  max_refine_iter = 5,
  K = NULL
)
```

Arguments

`bvec`	A NeuroVec-like object containing 4D fMRI data.
`mask`	A NeuroVol-like object (logical or numeric mask).
`block_size`	Approximate side length of blocks (e.g., 2 or 3). Must be > 0.
`ann_k`	Number of approximate (or exact) nearest neighbors per block. Must be >= 1.
`alpha`	Weighting for correlation vs. spatial proximity (0 <= alpha <= 1).
`correlation_metric`	Correlation metric ("pearson", "spearman", "robust").

`spatial_weighting`
: Spatial adjacency weighting ("gaussian", "binary").

`refine`
: Logical; whether to refine boundaries.

`max_refine_iter`
: Maximum iterations for boundary refinement. Must be >= 0.

`K`
: (Optional) Desired number of clusters.

Value

A list with elements:

cluster_map 3D array with cluster labels per voxel.

graph An `igraph` object used for clustering.

init_block_label Initial coarse partition (3D array) matching `mask` dimensions.

`block_partition` *Partition voxel coordinates into coarse blocks*

Description

Partition voxel coordinates into coarse blocks

Usage

```
block_partition(coords, block_size)
```

`build_acsc_graph` *Build ACSC adjacency graph*

Description

Build ACSC adjacency graph

Usage

```
build_acsc_graph(block_summary, ann_k, alpha, spatial_weighting, block_size)
```

cl_class_ids.cluster_result *Extract Class IDs from Cluster Result*

Description

This function extracts the cluster class identifiers from a cluster result object. It is a method for the cl_class_ids generic from the clue package.

Usage

```
cl_class_ids.cluster_result(x)
```

Arguments

x	A cluster_result object containing clustering information.

Value

An integer vector of cluster assignments, one for each data point.

See Also

cl_class_ids for the generic function.

commute_cluster *Commute Time Clustering*

Description

The commute_cluster function performs spatially constrained clustering on a NeuroVec instance using the commute time distance and K-means clustering.

Usage

```
commute_cluster(
  bvec,
  mask,
  K = 100,
  ncomp = ceiling(sqrt(K * 2)),
  alpha = 0.5,
  sigma1 = 0.73,
  sigma2 = 5,
  connectivity = 27,
  weight_mode = c("binary", "heat")
)
```

Arguments

bvec	A NeuroVec instance supplying the data to cluster.
mask	A NeuroVol mask defining the voxels to include in the clustering result. If the mask contains numeric data, nonzero values will define the included voxels. If the mask is a LogicalNeuroVol, then TRUE will define the set of included voxels.
K	The number of clusters to find. Default is 100.
ncomp	The number of components to use for the commute time embedding. Default is the ceiling of sqrt(K2).
alpha	A numeric value controlling the balance between spatial and feature similarity. Default is 0.5.
sigma1	A numeric value controlling the spatial weighting function. Default is 0.73.
sigma2	A numeric value controlling the feature weighting function. Default is 5.
connectivity	An integer representing the number of nearest neighbors to consider when constructing the similarity graph. Default is 27.
weight_mode	A character string indicating the type of weight function for the similarity graph. Options are "binary" and "heat". Default is "heat".

Value

A list of class commute_time_cluster_result with the following elements:

clusvol An instance of type ClusteredNeuroVol.

cluster A vector of cluster indices equal to the number of voxels in the mask.

centers A matrix of cluster centers with each column representing the feature vector for a cluster.

coord_centers A matrix of spatial coordinates with each row corresponding to a cluster.

See Also

snic

Examples

```
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
vec <- replicate(10, NeuroVol(array(runif(202020), c(20,20,20)),
NeuroSpace(c(20,20,20))), simplify=FALSE)
vec <- do.call(concat, vec)

commute_res <- commute_cluster(vec, mask, K=100)
```

compute_centroids *Compute Centroids of Clusters*

Description

The compute_centroids function calculates the center and centroid of each cluster given a feature matrix, grid, and cluster assignments.

Usage

```
compute_centroids(feature_mat, grid, assignment, medoid = FALSE)
```

Arguments

feature_mat	A matrix of features, where columns represent data points and rows represent features.
grid	A matrix representing the spatial grid of data points.
assignment	A vector containing the cluster assignment for each data point.
medoid	A logical value indicating whether to calculate medoids instead of means for cluster centers and centroids. Default is FALSE.

Value

A list containing two elements:

center	A matrix containing the centers of each cluster.
centroid	A matrix containing the centroids of each cluster.

Examples

```
## Not run:
  # Assuming `feature_mat`, `grid`, and `assignment` are available
  centroids <- compute_centroids(feature_mat, grid, assignment)
  # To compute medoids instead of means
  medoids <- compute_centroids(feature_mat, grid, assignment, medoid=TRUE)

## End(Not run)
```

compute_cluster_centroids *Compute centroids of each cluster*

Description

Compute centroids of each cluster

Usage

```
compute_cluster_centroids(voxel_labels, feature_mat)
```

construct_block_label_array *Construct a 3D array of block labels*

Description

Construct a 3D array of block labels

Usage

```
construct_block_label_array(block_id, mask)
```

cor_to_centroid *Correlate a voxel's time-series with a cluster centroid*

Description

Correlate a voxel's time-series with a cluster centroid

Usage

```
cor_to_centroid(voxel_idx, lbl, feature_mat, cluster_centroids)
```

estimate_resolution *Estimate Louvain resolution parameter*

Description

Estimate Louvain resolution parameter

Usage

```
estimate_resolution(K, graph)
```

expand_block_labels *Expand block-level cluster labels to voxel level*

Description

Expand block-level cluster labels to voxel level

Usage

```
expand_block_labels(cluster_result, block_id, mask.idx)
```

find_boundary_voxels *Identify boundary voxels*

Description

boundary voxel = has at least one neighbor with a different label

Usage

```
find_boundary_voxels(voxel_labels, nn_index)
```

find_initial_points *Find Initial Cluster Centers for Supervoxel Algorithm*

Description

This function finds the initial cluster centers for a supervoxel algorithm. Supervoxels are used to partition 3D image data into volumetric regions, grouping similar voxels together. The initial cluster centers are crucial for the performance and quality of the final supervoxels.

Usage

```
find_initial_points(cds, grad, K = 100)
```

Arguments

cds	A matrix or data frame representing the spatial coordinates of the voxels.
grad	A vector representing the gradient values of the voxels.
K	The desired number of supervoxels (clusters) in the output (default: 100).

Value

A list containing two elements: selected - a vector of the selected indices corresponding to the initial cluster centers, coords - a matrix or data frame with the spatial coordinates of the initial cluster centers.

hello *Hello, World!*

Description

Prints 'Hello, world!'.

Usage

```
hello()
```

Examples

```
hello()
```

index_to_grid *Convert linear indices to 3D grid coordinates*

Description

Convert linear indices to 3D grid coordinates

Usage

```
index_to_grid(mask, indices)
```

is.cl_partition.cluster_result *Test if Object is a Partition*

Description

This function tests whether a cluster result object represents a partition. It is a method for the `is.cl_partition` generic from the `clue` package. For `cluster_result` objects, this always returns `TRUE` since cluster results represent valid partitions where each data point belongs to exactly one cluster.

Usage

```
is.cl_partition.cluster_result(x)
```

Arguments

`x`	A `cluster_result` object.

Value

`TRUE`, indicating that cluster results are always valid partitions.

See Also

`is.cl_partition` for the generic function.

knn_shrink *K-nearest-neighbor shrink*

Description

Replace each voxel by the mean of its k nearest neighbors in its local spatial neighborhood.

Usage

```
knn_shrink(bvec, mask, k = 5, connectivity = 27)
```

Arguments

bvec	A NeuroVec instance (the data).
mask	A NeuroVol mask defining the voxels to include. If numeric, nonzero = included.
k	The number of nearest neighbors to average over.
connectivity	The number of spatial neighbors to include in the search around each voxel.

Value

A SparseNeuroVec or similar object with the smoothed data.

Examples

```
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
bvec <- replicate(10,
                  NeuroVol(array(runif(20*20*20), c(20,20,20)),
                           NeuroSpace(c(20,20,20))),
                  simplify=FALSE)
bvec <- do.call(concat, bvec)

sbvec <- knn_shrink(bvec, mask, k=3)
```

merge_clus *Merge Clustering Results Using a Consensus Clustering Algorithm*

Description

The merge_clus function combines a set of clustering results using a consensus clustering algorithm.

Usage

```
merge_clus(x, method, ...)
```

Arguments

x	A clustering result, typically a list or an object of class "cluster_result".
method	A character string indicating the consensus clustering algorithm to use. Default is "SE". See cl_consensus for available methods.
...	Additional clustering results to be merged.

Value

A ClusteredNeuroVol instance.

See Also

cl_consensus, as.cl_hard_partition, cl_ensemble

Examples

```
# Assuming clustering1, clustering2, and clustering3 are objects of class "cluster_result"
merged_clustering <- merge_clus(clustering1, clustering2, clustering3, method="SE")
```

---

merge_clus.cluster_result

Merge Clustering Results for ClusteredNeuroVol Objects

Description

This method of merge_clus is specifically designed to merge clustering results for ClusteredNeuroVol objects.

Usage

```
## S3 method for class 'cluster_result'
merge_clus(x, method = "SE", ...)
```

Arguments

x	A ClusteredNeuroVol object or an object of class "cluster_result".
method	A character string indicating the consensus clustering algorithm to use. Default is "SE". See cl_consensus for available methods.
...	Additional clustering results to be merged.

Value

A ClusteredNeuroVol instance.

See Also

cl_consensus, as.cl_hard_partition, cl_ensemble

meta_clust *Meta Clustering for Cluster Results (S4 Generic)*

Description

The meta_clust generic function is used to define methods for different classes of input objects.

Usage

```
meta_clust(x, cuts)
```

Arguments

x	A clustering result, typically an object of class "cluster_result".
cuts	The number of cluster cuts to consider.

Value

Depends on the method called.

meta_clust.cluster_result *Meta Clustering for Cluster Results*

Description

The meta_clust function performs meta clustering on a given clustering result by applying hierarchical clustering or other clustering algorithms.

Usage

```
## S3 method for class 'cluster_result'
meta_clust(
  x,
  cuts = min(as.integer(length(x$centers)/2), 2),
  algo = "hclust",
  hclust_method = "ward.D"
)
```

Arguments

x	A clustering result, typically an object of class "cluster_result".
cuts	The number of cluster cuts to consider. Default is the minimum of half the number of centers and 2.
algo	A character string indicating the clustering algorithm to use. Default is "hclust" (hierarchical clustering).
hclust_method	A character string specifying the agglomeration method to use for hierarchical clustering. Default is "ward.D".

Value

A list containing:

`cvols`	A list of `ClusteredNeuroVol` instances.
`cuts`	The number of cluster cuts.
`cutmat`	A matrix representing the cluster assignments for each cut.
`hclus`	The hierarchical clustering result.

See Also

`hclust`, `cutree`

`preprocess_time_series` *Preprocess fMRI time-series data*

Description

Preprocess fMRI time-series data

Usage

```
preprocess_time_series(bvec, mask, correlation_metric)
```

`refine_voxel_boundaries` *Refine voxel boundaries using cached cluster centroids*

Description

For each boundary voxel, compare correlation with each neighboring cluster's cached centroid. This approach is much faster than comparing against all voxel time-series.

Usage

```
refine_voxel_boundaries(voxel_labels, feature_mat, coords, max_iter)
```

`run_louvain_clustering` *Run Louvain clustering*

Description

Run Louvain clustering

Usage

```
run_louvain_clustering(graph, resolution = NULL)
```

slice_msf *SLiCE-MSF: Slice-wise, Low-rank, Minimum-Spanning Forest Clustering*

Description

The slice_msf function performs spatially constrained clustering on a NeuroVec instance using the SLiCE-MSF algorithm. This method uses temporal sketching via DCT basis functions, split-half reliability weighting, and Felzenszwalb-Huttenlocher graph segmentation.

Usage

```
slice_msf(
  vec,
  mask,
  target_k_global = -1,
  target_k_per_slice = -1,
  r = 12,
  compactness = 5,
  min_size = 80,
  num_runs = 3,
  consensus = TRUE,
  stitch_z = TRUE,
  theta_link = 0.85,
  min_contact = 1,
  nbhd = 8,
  gamma = 1.5,
  k_fuse = NULL,
  min_size_fuse = NULL,
  use_features = FALSE,
  lambda = 0.7
)
```

Arguments

vec	A NeuroVec or SparseNeuroVec instance supplying the time series data to cluster.
mask	A NeuroVol mask defining the voxels to include in the clustering result. If the mask contains numeric data, nonzero values will define the included voxels. If the mask is a LogicalNeuroVol, then TRUE will define the set of included voxels.
target_k_global	Target number of clusters across entire volume (exact if positive, uses RAG agglomeration). Default is -1 (no target, uses natural FH clustering).
target_k_per_slice	Target number of clusters per slice (exact if positive, ignored if stitch_z=TRUE). Default is -1 (no target).
r	DCT sketch rank (number of basis functions, excluding DC). Default is 12.
compactness	A numeric value controlling the compactness of the clusters, with larger values resulting in more compact clusters. Internally mapped to the scale parameter k. Default is 5.

min_size	Minimum cluster size in voxels. Smaller clusters are merged. Default is 80.
num_runs	Number of independent segmentation runs. If > 1, consensus fusion is applied. Default is 3.
consensus	Logical. If TRUE and num_runs > 1, apply consensus fusion. Default is TRUE.
stitch_z	Logical. If TRUE, stitch 2D slice clusters into 3D clusters. Default is TRUE.
theta_link	Centroid similarity threshold for cross-slice stitching (0-1). Default is 0.85.
min_contact	Minimum touching voxels between slices to attempt stitching. Default is 1.
nbhd	Neighborhood connectivity (4, 6, or 8). Default is 8.
gamma	Reliability weighting exponent. Higher values emphasize reliable voxels. Default is 1.5.
k_fuse	Scale parameter for consensus fusion. If NULL, uses same as k. Default is NULL.
min_size_fuse	Minimum cluster size for consensus. If NULL, uses min_size. Default is NULL.
use_features	Use feature similarity in consensus fusion. Default is FALSE.
lambda	Mix parameter for consensus (0-1). Higher values weight label agreement. Default is 0.7.

Value

A `list` of class `slice_msf_cluster_result` with the following elements:

clusvol An instance of type ClusteredNeuroVol.

cluster A vector of cluster indices equal to the number of voxels in the mask.

centers A matrix of cluster centers with each column representing the feature vector for a cluster.

coord_centers A matrix of spatial coordinates with each row corresponding to a cluster.

runs If num_runs > 1, a list of individual run results.

References

Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. International journal of computer vision, 59(2), 167-181.

Examples

```
## Not run:
mask <- NeuroVol(array(1, c(64,64,32)), NeuroSpace(c(64,64,32)))
vec <- replicate(100, NeuroVol(array(rnorm(64*64*32), c(64,64,32)),
                NeuroSpace(c(64,64,32))), simplify=FALSE)
vec <- do.call(concat, vec)

# Single run
result <- slice_msf(vec, mask, K=200, num_runs=1)

# Multi-run consensus
result <- slice_msf(vec, mask, K=200, num_runs=3, consensus=TRUE)

## End(Not run)

# Using exact K targeting
result <- slice_msf(vec, mask, target_k_global=100, use_features=TRUE)
```

slice_msf_consensus *Consensus Fusion for SLiCE-MSF*

Description

Combines multiple SLiCE-MSF segmentation runs using consensus clustering.

Usage

```
slice_msf_consensus(
  run_results,
  mask,
  nbhd = 8,
  k_fuse = 0.3,
  min_size_fuse = 80,
  use_features = FALSE,
  lambda = 0.7
)
```

Arguments

run_results	List of results from slice_msf_single.
mask	A NeuroVol mask used in the original segmentation.
nbhd	Neighborhood connectivity (4, 6, or 8). Default is 8.
k_fuse	Scale parameter for fusion. Default is 0.30.
min_size_fuse	Minimum cluster size. Default is 80.
use_features	Use feature similarity in fusion. Default is FALSE.
lambda	Mix parameter (0-1). Default is 0.7.

Value

A list with fused labels.

slice_msf_single *Single Run SLiCE-MSF Segmentation*

Description

Lower-level function that performs a single run of SLiCE-MSF segmentation. Most users should use slice_msf instead.

Usage

```
slice_msf_single(
  vec,
  mask,
  r = 12,
  k = 0.32,
  min_size = 80,
  nbhd = 8,
  stitch_z = TRUE,
  theta_link = 0.85,
  min_contact = 1,
  gamma = 1.5
)
```

Arguments

vec	A NeuroVec instance supplying the time series data.
mask	A NeuroVol mask defining the voxels to include.
r	DCT sketch rank. Default is 12.
k	Scale parameter (0-2). Smaller values create more clusters. Default is 0.32.
min_size	Minimum cluster size. Default is 80.
nbhd	Neighborhood connectivity (4, 6, or 8). Default is 8.
stitch_z	Enable cross-slice stitching. Default is TRUE.
theta_link	Centroid similarity threshold for stitching. Default is 0.85.
min_contact	Minimum contact voxels for stitching. Default is 1.
gamma	Reliability weighting exponent. Default is 1.5.

Value

A list with labels, weights, and sketch matrices.

---

snic *SNIC: Simple Non-Iterative Clustering*

Description

The SNIC function performs a spatially constrained clustering on a NeuroVec instance using the Simple Non-Iterative Clustering (SNIC) algorithm.

Usage

```
snic(vec, mask, compactness = 5, K = 500, max_iter = 100)
```

Arguments

vec	A NeuroVec instance supplying the data to cluster.
mask	A NeuroVol mask defining the voxels to include in the clustering result. If the mask contains numeric data, nonzero values will define the included voxels. If the mask is a LogicalNeuroVol, then TRUE will define the set of included voxels.
compactness	A numeric value controlling the compactness of the clusters, with larger values resulting in more compact clusters. Default is 5.
K	The number of clusters to find. Default is 500.
max_iter	Maximum number of iterations for the SNIC algorithm. Default is 100. Currently ignored as SNIC algorithm uses internal convergence criteria.

Value

A list of class snic_cluster_result with the following elements:

clusvol An instance of type ClusteredNeuroVol.

gradvol A NeuroVol instance representing the spatial gradient of the reference volume.

cluster A vector of cluster indices equal to the number of voxels in the mask.

centers A matrix of cluster centers with each column representing the feature vector for a cluster.

coord_centers A matrix of spatial coordinates with each row corresponding to a cluster.

See Also

supervoxels

Examples

```
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
vec <- replicate(10, NeuroVol(array(runif(202020), c(20,20,20)),
NeuroSpace(c(20,20,20))), simplify=FALSE)
vec <- do.call(concat, vec)

snic_res <- snic(vec, mask, compactness=5, K=100)
```

---

spatial_gradient *Spatial Gradient Calculation*

Description

The spatial_gradient function calculates the spatial gradient of a NeuroVol instance within the specified mask.

Usage

```
spatial_gradient(vol, mask, sigma = 0.5)
```

Arguments

vol	A NeuroVol instance for which the spatial gradient should be calculated.
mask	A NeuroVol mask defining the voxels to include in the spatial gradient calculation. If the mask contains numeric data, nonzero values will define the included voxels. If the mask is a LogicalNeuroVol, then TRUE will define the set of included voxels.
sigma	A numeric value controlling the spatial weighting function. Default is 0.5.

Value

A NeuroVol instance containing the spatial gradient values for the input vol.

See Also

spatial_laplacian, weighted_spatial_adjacency

Examples

```
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
input_vol <- NeuroVol(array(runif(202020), c(20,20,20)),
NeuroSpace(c(20,20,20)))

gradient_vol <- spatial_gradient(input_vol, mask)
```

---

summarize_blocks *Summarize voxel blocks*

Description

Summarize voxel blocks

Usage

```
summarize_blocks(feature_mat, coords, block_id)
```

---

supervoxels *Supervoxel Clustering (3D volumes)*

Description

Cluster a NeuroVec instance into a set of spatially constrained clusters.

Usage

```
supervoxels(
  bvec,
  mask,
  K = 500,
  sigma1 = 1,
  sigma2 = 2.5,
  iterations = 50,
  connectivity = 27,
  use_medoid = FALSE,
  use_gradient = TRUE,
  alpha = 0.5
)
```

Arguments

bvec	A NeuroVec instance supplying the data to cluster.
mask	A NeuroVol mask defining the voxels to include. If numeric, nonzero = included.
K	The number of clusters to find (default 500).
sigma1	The bandwidth of the heat kernel for the data vectors.
sigma2	The bandwidth of the heat kernel for the coordinate vectors.
iterations	The maximum number of cluster iterations.
connectivity	The number of nearest neighbors defining the neighborhood.
use_medoid	Logical; whether to use medoids rather than means for cluster centers.
use_gradient	Logical; use the image gradient to initialize clusters if possible.
alpha	The relative weighting of data similarity vs spatial similarity; alpha=1 = all data weighting, alpha=0 = purely spatial weighting.

Details

The algorithm:

1. Scale input data (bvec) so each feature dimension is centered and scaled.
2. If use_gradient = TRUE, initialize cluster seeds using gradient-based heuristics.
3. Run an iterative, spatially-constrained clustering that updates voxel assignments based on both feature similarity (bandwidth sigma1) and spatial proximity (bandwidth sigma2), weighted by alpha.
4. Return the final clusters, plus the feature-space and coordinate-space centers.

Value

A list (of class cluster_result) with elements:

clusvol	ClusteredNeuroVol containing the final clustering.
cluster	Integer vector of cluster assignments for each voxel.
centers	Matrix of cluster centers in feature space.
coord_centers	Matrix of cluster spatial centroids.

Examples

```
mask <- NeuroVol(array(1, c(20,20,20)), NeuroSpace(c(20,20,20)))
bvec <- replicate(10,
                  NeuroVol(array(runif(20*20*20), c(20,20,20)),
                           NeuroSpace(c(20,20,20))),
                  simplify=FALSE)
bvec <- do.call(concat, bvec)
cres1 <- supervoxels(bvec, mask, K=100, sigma1=1, sigma2=3)
```

`supervoxel_cluster_surface` *Supervoxel Clustering on a Surface*

Description

Cluster feature data on a cortical surface or mesh using a supervoxel-like approach.

Usage

```
supervoxel_cluster_surface(
  bsurf,
  K = 500,
  sigma1 = 1,
  sigma2 = 5,
  iterations = 50,
  connectivity = 6,
  use_medoid = FALSE
)
```

Arguments

`bsurf`	A `NeuroSurface` or similar object with geometry, coords, and data.
`K`	Number of clusters.
`sigma1`	Heat kernel bandwidth for feature similarity (data vectors).
`sigma2`	Heat kernel bandwidth for spatial similarity (coordinate vectors).
`iterations`	Max iterations.
`connectivity`	Neighborhood size on the surface (e.g., # of nearest mesh neighbors).
`use_medoid`	Whether to use medoids for cluster centers.

Value

A `list` with:

clusvol A `NeuroSurface` storing the final clustering result.

clusters Integer vector of cluster assignments (one per vertex).

centers Matrix of cluster centers.

coord_centers Matrix of spatial centroid coordinates.

index_sets List of vertex indices for each cluster.

`supervoxel_cluster_time` *Supervoxel Clustering in Time*

Description

Cluster feature matrix (rows = time points) in a "supervoxel" style but over temporal dimension.

Usage

```
supervoxel_cluster_time(
  feature_mat,
  K = min(nrow(feature_mat), 100),
  sigma1 = 1,
  sigma2 = 3,
  iterations = 50,
  TR = 2,
  filter = list(lp = 0, hp = 0),
  use_medoid = FALSE,
  nreps = 5
)
```

Arguments

`feature_mat`	A matrix (nrows = time points, ncols = features) or vice versa.
`K`	Number of clusters.
`sigma1`	Heat kernel bandwidth for feature similarity (data vectors).
`sigma2`	Heat kernel bandwidth for spatial similarity (coordinate vectors).
`iterations`	Maximum number of cluster iterations.
`TR`	Repetition time (seconds).
`filter`	List specifying optional frequency filters, e.g., `list(lp=0.1, hp=0)`.
`use_medoid`	Whether to use medoids for cluster centers.
`nreps`	Number of repeated initializations.

Value

A list of cluster results (one per repetition), each of which has the same structure as `supervoxel_cluster_fit()`.

Examples

```
feature_mat <- matrix(rnorm(100 * 10), 100, 10)
library(future)
plan(multicore)
cres <- supervoxel_cluster_time(t(feature_mat), K=20)
```

`tesselate` *Tesselate a Mask Volume into K Clusters using K-means*

Description

This function tesselates a given mask volume into K clusters using k-means clustering applied to spatial coordinates. It returns a clustered mask volume object.

Usage

```
tesselate(mask, K = 100)
```

Arguments

`mask`	A `NeuroVol` object representing the mask volume.
`K`	An integer value specifying the number of clusters (default: 100). If `K` exceeds the number of nonzero voxels, a warning is issued and `K` is set to the number of nonzero voxels.

Value

An instance of `ClusteredNeuroVol` representing the clustered mask volume.

Examples

```
# Assuming you have a NeuroVol object 'mask' and you want to create 150 clusters
clustered_volume <- tesselate(mask, K = 150)
```