Package 'MRIreduce'

May 7, 2026

Type Package

Title ROI-Based Transformation of Neuroimages into High-Dimensional Data Frames

Version 1.0.0

Maintainer Jinyao Tian <jinyaoti@usc.edu>

Description Converts NIfTI format T1/FL neuroimages into structured, high-dimensional 2D data frames with a focus on region of interest (ROI) based processing. The package incorporates the partition algorithm, which offers a flexible framework for agglomerative partitioning based on the Direct-Measure-Reduce approach. This method ensures that each reduced variable maintains a user-specified minimum level of information while remaining interpretable, as each maps uniquely to one variable in the reduced dataset. The partition framework is described in Millstein et al. (2020) <doi:10.1093/bioinformatics/btz661>. The package allows customization in variable selection, measurement of information loss, and data reduction methods for neuroimaging analysis and machine learning workflows.

License MIT + file LICENSE

Imports R6, Rcpp, fslr, neurobase, oro.nifti, parallel, partition, reshape2, reticulate

Depends R (>= 3.5.0)

SystemRequirements FSL (FMRIB Software Library, available at https://fsl.fmrib.ox.ac.uk/fsl/docs/#/install/index)

LinkingTo Rcpp

Encoding UTF-8

Language en-US

RoxygenNote 7.3.2

Suggests DT, EveTemplate, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://uscbiostats.github.io/MRIreduce/

Additional_repositories https://neuroconductor.org/releases/2020/05

Config/testthat/edition 3

NeedsCompilation yes

Author Joshua Milstein [aut],
Jinyao Tian [aut, cre]

Repository CRAN

Date/Publication 2026-04-21 20:42:25 UTC

Contents

`eve_Fl` *Process FLAIR Neuroimages and Register to EVE Brain Template*

Description

This function processes FLAIR (Fluid-Attenuated Inversion Recovery) neuroimages for better lesion detection, especially in the periventricular area by suppressing the CSF signal. It involves steps such as reading the image, reorienting, bias correction using N4, brain extraction, registration to the EVE template, and tissue segmentation. It finally calculates the intracranial volume and outputs the data.

Usage

```
eve_Fl(
  fpath,
  outpath,
  fsl_path,
  fsl_outputtype = "NIFTI_GZ",
  template_img_path = NULL
)
```

Arguments

fpath	Character string specifying the path to the FLAIR image file. The file should be in NIFTI file format (.nii.gz).
outpath	Character string specifying the output directory where the processed data is saved.
fsl_path	Character string specifying the path to the FSL software on the system.
fsl_outputtype	Character string specifying the type of output file format for FSL; defaults to "NIFTI_GZ".
template_img_path	Optional path to an EVE template image. If NULL, the function will try to use the optional package EveTemplate.

Details

The function uses FSL tools for image processing steps such as reorientation to standard space, bias correction, brain extraction, and tissue segmentation. Segmentation into different tissue types (CSF, grey matter, and white matter) is performed using FSL's FAST tool. Volumes are calculated based on the segmented tissues.

Value

Returns a list containing three elements: intensities, tissues, and brain_volume_cm3. Each element corresponds to the array of intensities, the segmented tissue data, and the calculated brain volumes, respectively. The function also saves these results as an .Rdata file at the specified output path.

Examples

```
if (FALSE) { # Replace paths with local files and an installed FSL setup.
eve_Fl("path/to/your/flair/image.nii.gz",
       "path/to/output/",
       "/usr/local/fsl",
       "NIFTI_GZ")
}
```

---

eve_T1 *Process T1-weighted Brain MRI Data with FSL and Register to EVE Atlas*

Description

This function takes a T1-weighted brain MRI image, performs bias correction, reorientation to standard space, brain extraction using FSL's BET, and registration to the EVE brain template. It then segments the brain volume into different tissue types, calculates intracranial volume and outputs the results as an Rdata file.

Usage

```
eve_T1(
  fpath,
  outpath,
  fsl_path,
  fsl_outputtype = "NIFTI_GZ",
  template_img_path = NULL
)
```

Arguments

fpath	Character string specifying the path to one T1-weighted MRI file. The file should be in NIFTI file format (.nii.gz).
outpath	Character string specifying the output directory where the results will be saved as an Rdata file.
fsl_path	Character string specifying the directory where FSL is installed.
fsl_outputtype	Character string specifying the FSL output file type. Defaults to "NIFTI_GZ".
template_img_path	Optional path to an EVE template image. If NULL, the function will try to use the optional package EveTemplate.

Details

The function begins by loading the EVE brain template for image registration. It then reads the T1-weighted MRI file and reorients it to standard space using FSL's fslreorient2std. Following reorientation, it applies bias correction with FSL's fsl_biascorrect, which is necessary to correct for field inhomogeneities that can affect quantitative analysis. The next step involves using FSL's Brain Extraction Tool (BET) to isolate the brain from non-brain tissue which is crucial for accurate subsequent analysis. After brain extraction, the image is registered to the EVE brain atlas using FLIRT, ensuring that the image is aligned with a standard coordinate space for comparable anatomical analysis. Subsequent to registration, the function uses FSL's FAST tool to segment the brain into white matter, grey matter, and cerebrospinal fluid, which are essential for studying brain structure and pathology. Finally, it calculates the volume of these tissues, providing key data points for clinical and research applications. Each step logs its progress with timestamps, aiding in debugging and optimization of processing times.

Value

Saves the image data after processing to the specified output path. Outputs an Rdata file containing three components: image intensities array, segmented tissues array, and brain volume metrics.

Examples

```
if (FALSE) { # Replace paths with local files and an installed FSL setup.
eve_T1("path/to/your/image.nii.gz", "path/to/output", "/usr/local/fsl", "NIFTI_GZ")
}
```

map2_eve *Plot Mask on EVE Template using Python and Nilearn*

Description

This function plots a mask image on the EVE template using Python's Nilearn library and optionally saves the plot as an image. It checks that the required Python libraries are available in the environment configured for `reticulate`.

Usage

```
map2_eve(
  mask_img_path,
  cmap = "bwr_r",
  alpha = 1,
  save_path = NULL,
  template_img_path = NULL,
  ...
)
```

Arguments

`mask_img_path`	A string representing the file path to the mask NIfTI file.
`cmap`	The colormap to use. Either a string (name of a matplotlib colormap) or a matplotlib colormap object. Default is 'bwr_r'.
`alpha`	Transparency level for the overlay. Default is 1.
`save_path`	A string representing the file path where the plot image should be saved (e.g., "output.png"). If NULL, the plot will be shown interactively instead of saved. Default is NULL.
`template_img_path`	Optional path to an EVE template image. If `NULL`, the function will try to use the optional package `EveTemplate`.
`...`	Additional arguments to customize the anatomical plot. These arguments are passed directly to the Python function `nilearn.plotting.plot_anat`. You can specify parameters such as `title`, `display_mode`, `cut_coords`, `dim`, etc. For more details on available options, refer to the official documentation at: https://nilearn.github.io/stable/modules/generated/nilearn.plotting.plot_anat.html

Details

The function uses the Python environment configured for `reticulate`. It checks that the required Python libraries `nilearn`, `nibabel`, and `matplotlib` are already available, then calls the Python helper `plot_mask_on_eve`. The background template can be provided directly, or resolved through the optional package `EveTemplate`. If `save_path` is provided, the plot is saved as an image at the specified location.

Value

None

Examples

```
if (FALSE) { # Requires a configured Python environment and local neuroimaging files.
map2_eve(
  mask_img_path = "/path/to/mask_nifti_GM_Volume.nii.gz",
  save_path = "/path/to/save_output.png",
  template_img_path = "/path/to/eve_template.nii.gz",
  cmap = "bwr_r",
  alpha = 0.8,
  title = "Mask on EVE Template"
)
}
```

`map_feature2_loc` *Map Reduced Feature Back to Brain Image Voxel Locations*

Description

This function maps a given feature name to voxel locations by identifying the pattern in the feature name. If the feature name contains "reduced_var", it follows the format "Roi_module_reduced_var" and this indicates that it is a reduced feature. Otherwise, it follows the format "Roi_Vnumber", which is not a reduced feature by Partition. Based on the pattern, the function extracts the appropriate region of interest (ROI) and retrieves voxel locations from intensity data.

Usage

```
map_feature2_loc(feature_name, threshold, main_dir)
```

Arguments

`feature_name`	String. A feature name.
`threshold`	a numeric between 0 and 1. The Partition threshold value applied to the data.
`main_dir`	String. The main directory containing the data files.

Details

Each voxel in a brain image corresponds to a specific feature, establishing a one-to-one mapping between a voxel and a feature. This relationship allows us to localize particular brain features to specific brain areas at the voxel level, enabling visualization of these features. However, after applying Super-Partition and Partition techniques, multiple brain features are aggregated into a single reduced feature as part of a data reduction process. The goal of the following function is to relate the reduced feature back to its component features, and subsequently identify the brain image voxels to which those component features are mapped. This function performs the following steps:

1. **Check Feature Name Format:** It first checks if the `feature_name` contains "reduced_var". If it does, the function assumes the format "Roi_module_reduced_var", otherwise it assumes the format "Roi_Vnumber".
2. **Extract ROI and Mapping Information:**
   - For the "Roi_module_reduced_var" format, the ROI and module number are extracted, and the corresponding partition file is read to retrieve the mapping vector.
   - For the "Roi_Vnumber" format, the `V` number is extracted from the feature name and used as the mapping vector.
3. **Extract Voxel Locations:** The mapping vector is then used to extract voxel locations from the intensity data.

Value

A data frame containing the voxel locations (x,y,z Coordinates) corresponding to the extracted mapping column.

Examples

```
if (FALSE) { # After running the partition pipeline and creating local output files.
loc_df <- map_feature2_loc(
  feature_name = "inferior_frontal_gyrus_left_module4_reduced_var_13",
  threshold = 0.8,
  main_dir = "/path/to/data"
)
}
```

`PartitionPipeline` *Partition Pipeline for Image Analysis*

Description

This R6 class is designed to streamline the processing pipeline for image analysis, including steps from initial processing to combining independent variables with reduced variables by tissue type by ROI.

Methods

Public methods:

- `PartitionPipeline$new()`
- `PartitionPipeline$iproc()`
- `PartitionPipeline$supparfun()`
- `PartitionPipeline$map_suppar_roi()`
- `PartitionPipeline$partition_intensities()`
- `PartitionPipeline$tissue_segment()`

- PartitionPipeline$Cmb_tissue_type()
- PartitionPipeline$process_indep_variables()
- PartitionPipeline$Cmb_indep_with_dep()
- PartitionPipeline$clone()

Method `new()`**:**

Usage:

```
PartitionPipeline$new(
  tind = NULL,
  nfl = NULL,
  main_dir = NULL,
  tissue_type = NULL,
  outp_volume = TRUE,
  ICC_thresh_vec = NULL,
  suppar_thresh_vec = seq(0.7, 1, 0.01),
  B = 2000,
  roi = NULL,
  num_cores = NULL
)
```

Method `iproc()`**:**

Usage:

```
PartitionPipeline$iproc()
```

Method `supparfun()`**:**

Usage:

```
PartitionPipeline$supparfun()
```

Method `map_suppar_roi()`**:**

Usage:

```
PartitionPipeline$map_suppar_roi()
```

Method `partition_intensities()`**:**

Usage:

```
PartitionPipeline$partition_intensities()
```

Method `tissue_segment()`**:**

Usage:

```
PartitionPipeline$tissue_segment()
```

Method `Cmb_tissue_type()`**:**

Usage:

```
PartitionPipeline$Cmb_tissue_type()
```

Method `process_indep_variables()`**:**

Usage:

```
PartitionPipeline$process_indep_variables()
```

Method `Cmb_indep_with_dep()`**:**

Usage:

```
PartitionPipeline$Cmb_indep_with_dep()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PartitionPipeline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`run_partition_pipeline` *Run Partition Pipeline on Neuroimaging Data*

Description

This function initializes and executes a partitioning pipeline designed for processing neuroimaging data. It handles tasks such as image processing, super partition analysis, mapping, and combining of data based on specified thresholds and parameters.

Usage

```
run_partition_pipeline(
  tind,
  nfl,
  main_dir,
  tissue_type,
  ICC_thresh_vec,
  num_cores = 1,
  suppar_thresh_vec = seq(0.7, 1, 0.01),
  B = 2000,
  outp_volume = TRUE
)
```

Arguments

`tind`	Index or identifier for the type of tissue under analysis.
`nfl`	List of file names (full paths) that need to be processed.
`main_dir`	Main directory where outputs and intermediate results will be saved.
`tissue_type`	Type of tissue for segmentation.
`ICC_thresh_vec`	A vector of threshold values for Intraclass Correlation Coefficient used in the Partition Algorithm.

num_cores Number of cores to use for parallel processing. Default to 1.

suppar_thresh_vec
Optional; a sequence of threshold values used in Super Partitioning. Default is a sequence from 0.7 to 1 by 0.01.

B Optional; the maximum size of modules to be considered in partitioning. Default is 2000.

outp_volume Optional; a logical indicating whether volume outputs should be generated. Default is TRUE.

Details

The function configures and runs a series of operations that are typical in neuroimage analysis, especially focusing on ROI-based transformations. Each step of the pipeline, from initial image processing (iproc) to the final combination of independent variables with reduced variables (Cmb_indep_with_dep), is executed in sequence. Adjustments to the pipeline's behavior can be made by changing the function parameters, allowing for custom analysis flows.

Value

The function does not return a value but will output results directly to the specified main_dir as side effects of the processing steps.

---

suppar *Super Partitioning of Variables Based on Correlation*

Description

This function identifies and groups highly correlated variables into modules from a given dataset using a series of correlation computations stored across temporary files. It utilizes a hierarchical chunk processing method to handle large datasets.

Usage

```
suppar(
  tmp,
  thresh = NULL,
  n.chunkf = 10000,
  B = 2000,
  compute.corr = TRUE,
  dist.thresh = NULL,
  dir.tmp
)
```

Arguments

tmp	A data frame or matrix of data to be analyzed.
thresh	Numeric vector; thresholds for defining the correlation strength necessary to consider two variables as connected or dependent. The function creates modules of variables that have correlations above these thresholds.
n.chunkf	Integer; the number of features to process per chunk in the correlation analysis.
B	Integer; the maximum size of a module. If a module reaches this size, it will not be merged with other modules even if its members are correlated with members of another module.
compute.corr	Logical; should the correlation be computed (TRUE) or pre-computed correlations be used (FALSE).
dist.thresh	Optional; a distance threshold to apply before computing correlations, allowing for spatial constraints on correlation computation.
dir.tmp	Directory path where temporary correlation files are stored.

Details

suppar function starts by setting up the environment and preparing the data. If compute.corr is TRUE, it computes the correlation and stores the results in temporary files in dir.tmp. It then loads these files one by one, aggregates correlated variables into groups using the partagg function, and finally, it cleans up the temporary files.

The corfun1 and corfun2 are helper functions called within suppar to manage the computation of correlations in chunks and saving those in a manageable manner, which helps in processing large datasets without overwhelming memory resources.

partagg, an Rcpp function, efficiently processes and aggregates variables into modules based on the correlation data read from the temporary files. It ensures that the size of any module does not exceed the B parameter.

Value

A list containing two elements: - A list of character vectors, where each vector contains the names of variables that form a module. - A character vector of independent variables not included in any module.

Examples

```
tmp <- matrix(rnorm(200), ncol = 20)
dir_tmp <- file.path(tempdir(), "suppar-example")
suppar(tmp = tmp, thresh = c(0.3, 0.1), n.chunkf = 20, B = 10,
       compute.corr = TRUE, dir.tmp = dir_tmp)
```

Index