

---

# **NiMARE**

***Release 0.0.1+0.g1b4d841.dirty***

**Nov 20, 2019**



---

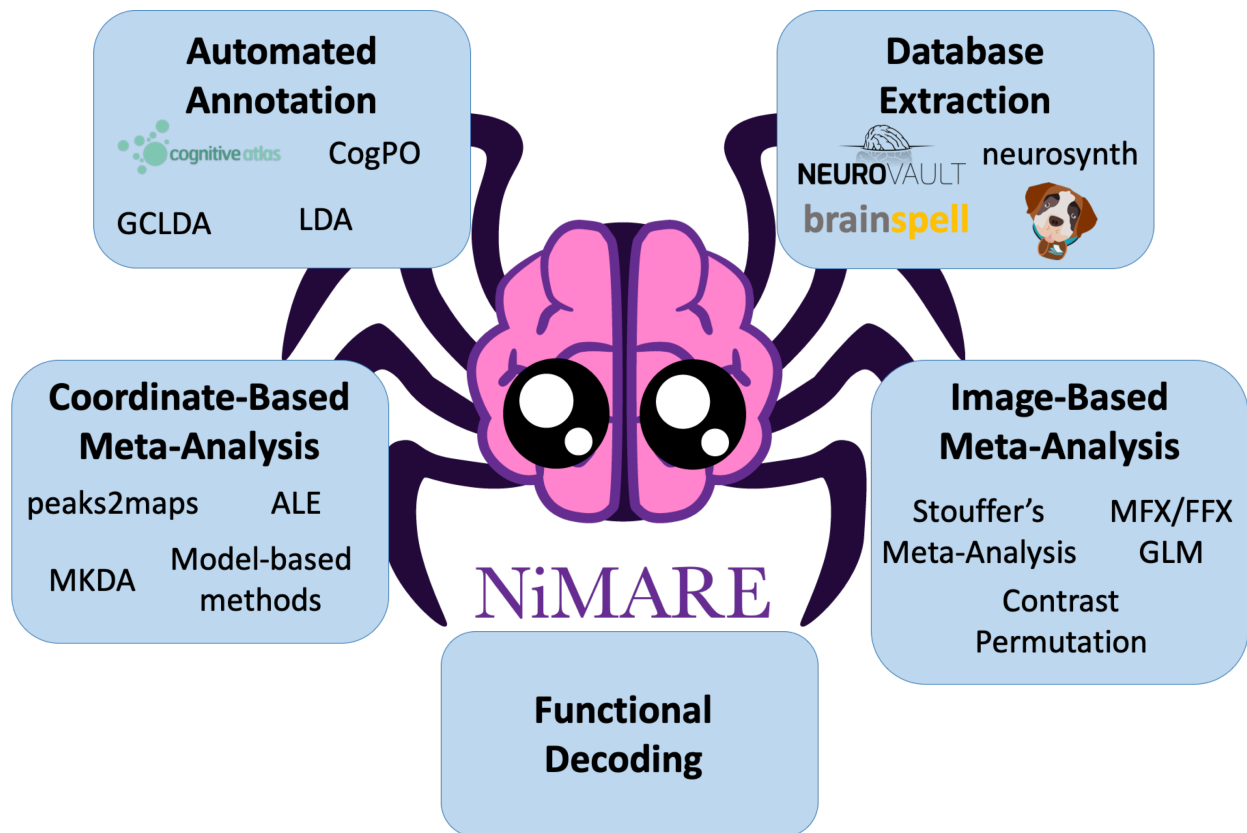
## Contents:

---

<b>1</b>	<b>About NiMARE</b>	<b>3</b>
1.1	A Proposed Meta-Analytic Ecosystem . . . . .	3
<b>2</b>	<b>Installation</b>	<b>7</b>
2.1	What Next? . . . . .	7
<b>3</b>	<b>Command Line Interface</b>	<b>9</b>
3.1	Sub-commands: . . . . .	9
<b>4</b>	<b>Contributing to NiMARE</b>	<b>15</b>
4.1	Governance . . . . .	15
4.2	Code of conduct . . . . .	15
4.3	Asking questions about using NiMARE . . . . .	16
4.4	Labels . . . . .	16
4.5	Making a change . . . . .	16
4.6	Recognizing contributions . . . . .	17
4.7	Thank you! . . . . .	17
<b>5</b>	<b>NiMARE Developer Guide</b>	<b>19</b>
5.1	Coding Style . . . . .	19
<b>6</b>	<b>API</b>	<b>21</b>
6.1	<code>nimare.dataset</code> : Dataset IO . . . . .	21
6.2	<code>nimare.meta</code> : Meta-analytic algorithms . . . . .	24
6.3	<code>nimare.correct</code> : Multiple comparisons correction . . . . .	86
6.4	<code>nimare.annotate</code> : Automated annotation . . . . .	87
6.5	<code>nimare.decode</code> : Functional characterization analysis . . . . .	98
6.6	<code>nimare.parcellate</code> : Meta-analytic parcellation . . . . .	109
6.7	<code>nimare.io</code> : Input/Output . . . . .	123
6.8	<code>nimare.stats</code> : Statistical functions . . . . .	124
6.9	<code>nimare.utils</code> : Utility functions and submodules . . . . .	125
6.10	<code>nimare.workflows</code> : Common workflows . . . . .	127
6.11	<code>nimare.base</code> : Base classes . . . . .	129
<b>7</b>	<b>Indices and tables</b>	<b>131</b>
	<b>Bibliography</b>	<b>133</b>

<b>Python Module Index</b>	<b>137</b>
<b>Index</b>	<b>139</b>

NiMARE is a Python package for neuroimaging meta-analyses. It makes conducting scary meta-analyses a dream!  
To install NiMARE check out our [installation guide](#).



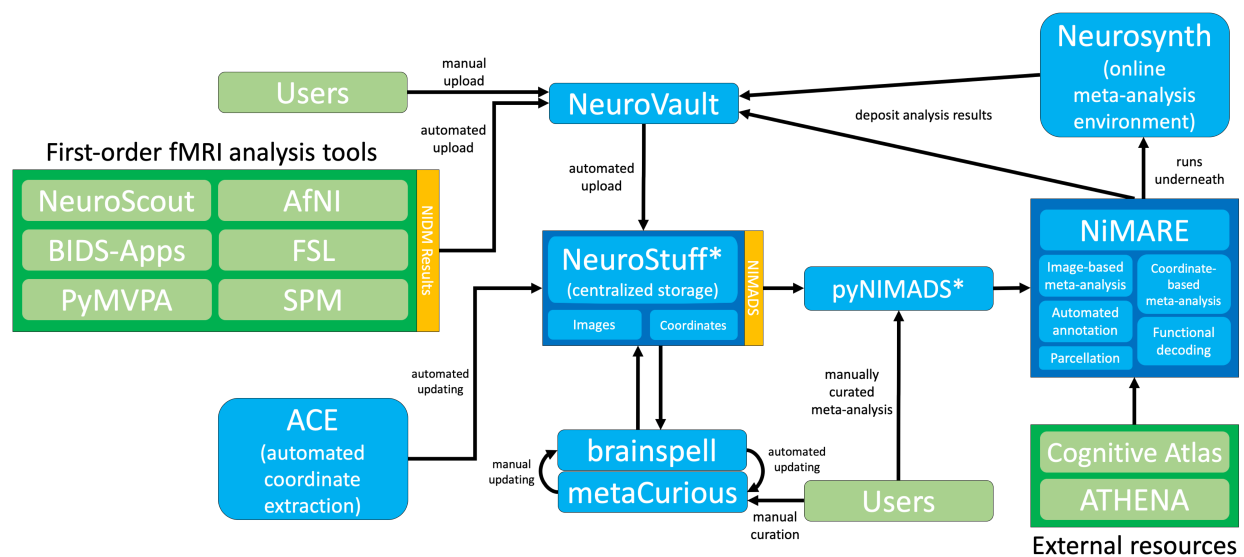


NiMARE is a Python package for performing meta-analyses, and derivative analyses using meta-analytic data, of the neuroimaging literature. While meta-analytic packages exist which implement one or two algorithms each, NiMARE provides a standard syntax for performing a wide range of analyses and for interacting with databases of coordinates and images from fMRI studies (e.g., [brainspell](#), [Neurosynth](#), and [NeuroVault](#)).

NiMARE joins a growing Python ecosystem for neuroimaging research, which includes such tools as [Nipype](#), [Nistats](#), and [Nilearn](#). As with these other tools, NiMARE is open source, collaboratively developed, and built with ease of use in mind.

This page outlines NiMARE’s purpose and its role in a proposed meta-analytic ecosystem.

## 1.1 A Proposed Meta-Analytic Ecosystem



NiMARE aims to fill a gap in a burgeoning meta-analytic ecosystem. The goal of NiMARE is to collect a wide range of meta-analytic tools in one Python library. Currently, those methods are spread out across a range of programming languages and user interfaces, or are never even translated from the original papers into useable tools. NiMARE operates on NIMADS-format datasets, which users will be able to compile by searching the NeuroStuff database with the pyNIMADS library. A number of other services in the ecosystem will then use NiMARE functions to perform meta-analyses, including Neurosynth 2.0, [NeuroVault](#), and [metaCurious](#).

---

**Note:** This page outlines a *tentative* plan for a system of services for neuroimaging meta-analysis. Several of the services detailed here do not currently exist or only partially support the functionality described below. This plan is likely to change over time.

---

### 1.1.1 Neurosynth 2.0

[Neurosynth](#) currently stores a coordinated-based database of over 14,000 neuroimaging papers (automatically curated by [ACE](#)), provides a web interface for automated meta-analyses, functional decoding, and gene expression visualization, and provides a Python package implementing the above methods.

In order to improve modularization, the next iteration of Neurosynth will limit itself to a web interface for meta-analytic model specification and providing a centralized storage for large-scale meta-analyses, but not actually implementing the algorithms used to run those meta-analyses or to perform the other services provided on the website (e.g., functional decoding and topic modeling). The algorithms currently implemented in the [Neurosynth Python package](#) will be implemented (among many others) in NiMARE. Under the current plan, the database at the moment stored by Neurosynth will instead be stored in the NeuroStuff database, which will also store other coordinate- and image-based meta-analytic databases in NIMADS format.

### 1.1.2 NeuroVault

[NeuroVault](#) is a database for unthresholded images. Users may upload individual maps or [NIDM Results](#), which can be exported from a number of fMRI analysis tools, like [AfNI](#), [SPM](#), [FSL](#), and [NeuroScout](#).

NeuroVault also has integrations with [NeuroPower](#) (for power analyses) and [Neurosynth](#) (for functional decoding), and supports simple image-based meta-analyses.

### 1.1.3 brainspell

[brainspell](#) is a clone of the Neurosynth database meant for crowdsourced manual annotation. It provides a website where users can correct mistakes made by ACE or can add labels from multiple cognitive ontologies (including the [Cognitive Paradigm Ontology](#) and the [Cognitive Atlas](#)) to experiments.

### 1.1.4 metaCurious

[metaCurious](#) is a new frontend (i.e., website) for brainspell, oriented toward meta-analysts. MetaCurious provides search and curation tools for researchers to build meta-analytic samples for analysis. Search criteria, reasons for exclusion, and other labels may be added by the researcher and fed back into the underlying database, resulting in goal-oriented manual annotation. MetaCurious generates GitHub repositories for meta-analytic samples, which will also be NiMARE-compatible in the future.



### 1.1.5 NIMADS

NIMADS is a new standard for organizing and representing meta-analytic neuroimaging data. NIMADS will be used by NeuroStuff, pyNIMADS, *metaCurious*, and NiMARE.

### 1.1.6 NeuroStuff

NeuroStuff (tentatively named) will act as a centralized repository for coordinates and maps from neuroimaging studies, stored in NIMADS format. Users will be able to query and add to the repository using its API and the pyNIMADS Python library.

### 1.1.7 pyNIMADS

pyNIMADS (also tentatively named) is a planned Python library that will act as a wrapper for the NeuroStuff API, allowing users to query the database and to build NiMARE-compatible datasets for analysis.



# CHAPTER 2

---

## Installation

---

You'll need to set up your software environment to be able to use `nimare`. To set up a local environment, you will need Python  $\geq 3.6$  and the following packages will need to be installed:

- nibabel
- numpy
- scipy
- pandas
- statsmodels
- nipy
- scikit-learn
- nilearn

You can then install `nimare` with:

```
python setup.py develop
```

## 2.1 What Next?

For an overview of what you can do with NiMARE see [NiMARE Documentation](#).

To get right to using NiMARE see the documentation on the [command line interface](#).

If you have questions, or need help with using NiMARE, check out [NeuroStars](#).



---

## Command Line Interface

---

NiMARE provides several workflows as command-line interfaces, including ALE meta-analysis, meta-analytic coactivation modeling (MACM) analysis, peaks2maps image reconstruction, and contrast map meta-analysis. Each workflow should generate a boilerplate paragraph with details about the workflow and citations that can be used in a manuscript.

To use NiMARE from the command line, open a terminal window and type:

```
nimare --help
```

This will print the instructions for using the command line interface in your command line.

```
usage: nimare [-h]
              {ale, conperm, peaks2maps, macm, scale, metacluster, sleuth2nimare,
              ↪neurosynth2nimare}
              ...
```

### 3.1 Sub-commands:

#### 3.1.1 ale

Run an activation likelihood estimation (ALE) meta-analysis on a Sleuth text file. ALE is a permutation-based meta-analysis of coordinates that uses 3D Gaussians to model activation.

```
nimare ale [-h] [--output_dir PATH] [--prefix PREFIX] [--file2 SLEUTH_FILE2]
           [--n_iters N_ITERS] [--v_thr V_THR] [--fwhm FWHM]
           [--n_cores N_CORES]
           sleuth_file
```

#### Positional Arguments

<b>sleuth_file</b>	Sleuth text file to analyze.
--------------------	------------------------------

### Named Arguments

<b>--output_dir</b>	Output directory. Default: “.”
<b>--prefix</b>	Common prefix for output maps. Default: “”
<b>--file2</b>	Optional second Sleuth file for subtraction analysis.
<b>--n_iters</b>	Number of iterations for permutation testing. Default: 10000
<b>--v_thr</b>	Voxel p-value threshold used to create clusters. Default: 0.001
<b>--fwhm</b>	Override sample size-based kernel determination with a single FWHM (in mm) applied to all experiments. Useful when sample size is not available for all data.
<b>--n_cores</b>	Number of processes to use for meta-analysis. If -1, use all available cores.

### 3.1.2 conperm

Meta-analysis of contrast maps using random effects and two-sided inference with empirical (permutation-based) null distribution and Family Wise Error multiple comparisons correction. Input may be a list of 3D files or a single 4D file.

```
nimare conperm [-h] [--output_dir PATH] [--prefix PREFIX] [--n_iters N_ITERS]  
FILE [FILE ...]
```

### Positional Arguments

<b>FILE</b>	Data to analyze. May be a single 4D file or a list of 3D files.
-------------	---

### Named Arguments

<b>--output_dir</b>	Output directory. Default: “.”
<b>--prefix</b>	Common prefix for output maps. Default: “”
<b>--n_iters</b>	Number of iterations for permutation testing. Default: 10000

### 3.1.3 peaks2maps

Method for performing coordinate-based meta-analysis that uses a pretrained deep neural network to reconstruct un-thresholded maps from peak coordinates. The reconstructed maps are evaluated for statistical significance using a permutation-based approach with Family Wise Error multiple comparison correction.

```
nimare peaks2maps [-h] [--output_dir PATH] [--prefix PREFIX]
                  [--n_iters N_ITERS]
                  sleuth_file
```

### Positional Arguments

**sleuth\_file** Sleuth text file to analyze.

### Named Arguments

**--output\_dir** Output directory.  
Default: “.”

**--prefix** Common prefix for output maps.  
Default: “”

**--n\_iters** Number of iterations for permutation testing.  
Default: 10000

### 3.1.4 macm

Run a meta-analytic coactivation modeling (MACM) analysis using activation likelihood estimation (ALE) on a NiMARE dataset file and a target mask.

```
nimare macm [-h] --mask MASK_FILE [--output_dir PATH] [--prefix PREFIX]
             [--n_iters N_ITERS] [--v_thr V_THR] [--n_cores N_CORES]
             dataset_file
```

### Positional Arguments

**dataset\_file** Dataset file to analyze.

### Named Arguments

**--mask, --mask\_file** Mask file

**--output\_dir** Output directory.  
Default: “.”

**--prefix** Common prefix for output maps.  
Default: “”

**--n\_iters** Number of iterations for permutation testing.  
Default: 10000

**--v\_thr** Voxel p-value threshold used to create clusters.  
Default: 0.001

**--n\_cores** Number of processes to use for meta-analysis. If -1, use all available cores.

### 3.1.5 scale

Method for performing Specific CoActivation Likelihood Estimation (SCALE), a modified meta-analytic coactivation modeling (MACM) that takes activation frequency bias into account, for delineating distinct core networks of coactivation, using a permutation-based approach.

```
nimare scale [-h] [--baseline BASELINE] [--output_dir PATH] [--prefix PREFIX]
             [--n_iters N_ITERS] [--v_thr V_THR] [--n_cores N_CORES]
             dataset_file
```

#### Positional Arguments

<b>dataset_file</b>	Dataset file to analyze.
---------------------	--------------------------

#### Named Arguments

<b>--baseline</b>	Voxel-wise baseline activation rates.
<b>--output_dir</b>	Output directory. Default: “.”
<b>--prefix</b>	Common prefix for output maps. Default: “”
<b>--n_iters</b>	Number of iterations for permutation testing. Default: 2500
<b>--v_thr</b>	Voxel p-value threshold used to create clusters. Default: 0.001
<b>--n_cores</b>	Number of processes to use for meta-analysis. If -1, use all available cores.

### 3.1.6 metacluster

Method for investigating recurrent patterns of activation across a meta-analytic dataset, thus identifying trends across a collection of experiments.

```
nimare metacluster [-h] [--output_dir PATH] [--prefix PREFIX]
                   [--kernel {ALEKernel,MKDAKernel,KDAKernel,Peaks2MapsKernel}]
                   [--algorithm {kmeans,dbscan,spectral}] --clust_range
                   CLUST_RANGE CLUST_RANGE (--coord | --img)
                   dataset_file
```

#### Positional Arguments

<b>dataset_file</b>	Dataset file to analyze.
---------------------	--------------------------



## Named Arguments

<b>--output_dir</b>	Output directory. Default: “.”
<b>--prefix</b>	Common prefix for output maps. Default: “”
<b>--kernel</b>	Possible choices: ALEKernel, MKDAKernel, KDAKernel, Peaks2MapsKernel Kernel estimator, for coordinate-based metaclustering. Default: “ALEKernel”
<b>--algorithm</b>	Possible choices: kmeans, dbscan, spectral Clustering algorithm to be used, from sklearn.cluster. Default: “kmeans”
<b>--clust_range</b>	Select a range for k over which clustering solutions will be evaluated (e.g., 2 10 will evaluate solutions with k = 2 clusters to k = 10 clusters).
<b>--coord</b>	Is input data coordinate-based? Default: False
<b>--img</b>	Is input data image-based? Default: False

### 3.1.7 sleuth2nimare

Convert a Sleuth text file to a NiMARE json file.

```
nimare sleuth2nimare [-h] text_file out_file
```

## Positional Arguments

<b>text_file</b>	Sleuth text file to convert.
<b>out_file</b>	Output file.

### 3.1.8 neurosynth2nimare

Convert a Neurosynth text file to a NiMARE json file.

```
nimare neurosynth2nimare [-h] [--annotations_file FILE] text_file out_file
```

## Positional Arguments

<b>text_file</b>	Neurosynth text file to convert.
<b>out_file</b>	Output file.

## Named Arguments

**--annotations\_file**    Optional annotations (features) file.

---

## Contributing to NiMARE

---

Welcome to the NiMARE repository! We're excited you're here and want to contribute.

These guidelines are designed to make it as easy as possible to get involved. If you have any questions that aren't discussed below, please let us know by opening an [issue](#)!

Before you start you'll need to set up a free [GitHub](#) account and sign in. Here are some [instructions](#).

### 4.1 Governance

Governance is a hugely important part of any project. It is especially important to have clear process and communication channels for open source projects that rely on a distributed network of volunteers, such as NiMARE.

NiMARE is currently supported by a small group of core developers. Even with only a couple of individuals involved in decision making processes, we've found that setting expectations and communicating a shared vision has great value.

By starting the governance structure early in our development, we hope to welcome more people into the contributing team. We are committed to continuing to update the governance structures as necessary. Every member of the NiMARE community is encouraged to comment on these processes and suggest improvements.

As the first interim [Benevolent Dictator for Life \(BDFL\)](#), Taylor Salo is ultimately responsible for any major decisions pertaining to NiMARE development. However, all potential changes are explicitly and openly discussed in the described channels of communication, and we strive for consensus amongst all community members.

### 4.2 Code of conduct

All NiMARE community members are expected to follow our [code of conduct](#) during any interaction with the project. That includes- but is not limited to- online conversations, in-person workshops or development sprints, and when giving talks about the software.

As stated in the code, severe or repeated violations by community members may result in exclusion from collective decision-making and rejection of future contributions to the NiMARE project.

## 4.3 Asking questions about using NiMARE

Please direct usage-related questions to [NeuroStars](#), with the tag “nimare”. The NiMARE developers follow NeuroStars, and will be able to answer your question there.

## 4.4 Labels

The current list of labels are [here](#) and include:

- *These issues contain a task that a member of the team has determined should require minimal knowledge of the existing codebase, and should be good for people new to the project.* If you are interested in contributing to NiMARE, but aren't sure where to start, we encourage you to take a look at these issues in particular.
- *These issues contain a task that a member of the team has determined we need additional help with.* If you feel that you can contribute to one of these issues, we especially encourage you to do so!
- *These issues point to problems in the project.* If you find new a bug, please give as much detail as possible in your issue, including steps to recreate the error. If you experience the same bug as one already listed, please add any additional information that you have as a comment.
- *These issues are asking for new features to be added to the project.* Please try to make sure that your requested feature is distinct from any others that have already been requested or implemented. If you find one that's similar but there are subtle differences please reference the other request in your issue.

## 4.5 Making a change

We appreciate all contributions to NiMARE, but those accepted fastest will follow a workflow similar to the following:

### 1. Comment on an existing issue or open a new issue referencing your addition.

This allows other members of the NiMARE development team to confirm that you aren't overlapping with work that's currently underway and that everyone is on the same page with the goal of the work you're going to carry out.

[This blog](#) is a nice explanation of why putting this work in up front is so useful to everyone involved.

### 2. Fork NiMARE.

Fork the [NiMARE repository](#) to your profile.

This is now your own unique copy of NiMARE. Changes here won't effect anyone else's work, so it's a safe space to explore edits to the code!

Make sure to [keep your fork up to date](#) with the master repository.

### 3. Make the changes you've discussed.

Try to keep the changes focused. We've found that working on a [new branch](#) makes it easier to keep your changes targeted.

When you're creating your pull request, please do your best to follow NiMARE's preferred style conventions. Namely, documentation should follow the [numpydoc](#) convention and code should adhere to [PEP8](#) as much as possible.

### 4. Submit a pull request.

Submit a [pull request](#).

A member of the development team will review your changes to confirm that they can be merged into the main codebase.

## 4.6 Recognizing contributions

We welcome and recognize all contributions from documentation to testing to code development. You can see a list of current contributors in our [zenodo](#) file. If you are new to the project, don't forget to add your name and affiliation there!

## 4.7 Thank you!

You're awesome.

- NOTE: These guidelines are based on contributing guidelines from the [STEMMRoleModels](#) project.



This guide provides a more detailed description of the organization and preferred coding style for NiMARE, for prospective code contributors.

## 5.1 Coding Style

NiMARE code should follow PEP8 recommendations. Additionally, we have modeled NiMARE's code on [scikit-learn](#).





## 6.1 `nimare.dataset`: Dataset IO

Classes for representing datasets of images and/or coordinates.

<code>nimare.dataset</code>	Classes for representing datasets of images and/or coordinates.
<code>nimare.dataset.Dataset(source[, target, mask])</code>	Storage container for a coordinate- and/or image-based meta-analytic dataset/database.

### 6.1.1 `nimare.dataset.Dataset`

**class Dataset** (*source*, *target*='mni152\_2mm', *mask*=None)

Storage container for a coordinate- and/or image-based meta-analytic dataset/database.

#### Parameters

- **source** (*str*) – JSON file containing dictionary with database information or the dict() object
- **target** (*str*) – Desired coordinate space for coordinates. Names follow NIDM convention.
- **mask** (*str*, *Nifti1Image*, or any Nilearn *Masker*) – Mask(er) to use. If None, uses the target space image, with all non-zero voxels included in the mask.

#### Methods

<code>get(self, dict_)</code>	Retrieve files and/or metadata from the current Dataset.
-------------------------------	--

Continued on next page

Table 2 – continued from previous page

<code>get_images(self[, ids, imtype])</code>	Get images of a certain type for a subset of studies in the dataset.
<code>get_labels(self[, ids])</code>	Extract list of labels for which studies in Dataset have annotations.
<code>get_metadata(self[, ids, field])</code>	Get metadata from Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>get_studies_by_coordinate(self, xyz[, r])</code>	Extract list of studies with at least one focus within radius r of requested coordinates.
<code>get_studies_by_label(self[, labels, ...])</code>	Extract list of studies with a given label.
<code>get_studies_by_mask(self, mask)</code>	Extract list of studies with at least one coordinate in mask.
<code>get_texts(self[, ids, text_type])</code>	Extract list of texts of a given type for selected IDs.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, \**params)</code>	Set the parameters of this estimator.
<code>slice(self, ids)</code>	Return a reduced dataset with only requested IDs.
<code>update_path(self, new_path)</code>	Update paths to images.

**get** (*self*, *dict\_*)

Retrieve files and/or metadata from the current Dataset.

**Parameters** *dict* (*dict*) – Dictionary specifying images or metadata to collect

**Returns** *results* – A dictionary of lists of requested data.

**Return type** *dict*

**get\_images** (*self*, *ids=None*, *imtype='z'*)

Get images of a certain type for a subset of studies in the dataset.

**Parameters**

- **ids** (*list*, *optional*) – A list of IDs in the Dataset for which to find texts. Default is None, in which case all texts of requested type are returned.
- **imtype** (*str*, *optional*) – Type of image to extract. Corresponds to column name in Dataset.images DataFrame. Default is 'z'.

**Returns** *images* – List of images of requested type for selected IDs.

**Return type** *list*

**get\_labels** (*self*, *ids=None*)

Extract list of labels for which studies in Dataset have annotations.

**Parameters** *ids* (*list*, *optional*) – A list of IDs in the Dataset for which to find labels. Default is None, in which case all labels are returned.

**Returns** *labels* – List of labels for which there are annotations in the Dataset.

**Return type** *list*

**get\_metadata** (*self*, *ids=None*, *field='sample\_sizes'*)

Get metadata from Dataset.

**Parameters**

- **ids** (*list*, *optional*) – A list of IDs in the Dataset for which to find texts. Default is None, in which case all texts of requested type are returned.

- **field**(*str*, *optional*) – Metadata field to extract. Corresponds to column name in Dataset.metadata DataFrame. Default is ‘sample\_sizes’.

**Returns metadata** – List of values of requested type for selected IDs.

**Return type** *list*

**get\_params**(*self*, *deep=True*)

Get parameters for this estimator.

**Parameters deep**(*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**get\_studies\_by\_coordinate**(*self*, *xyz*, *r=20*)

Extract list of studies with at least one focus within radius *r* of requested coordinates.

**Parameters**

- **xyz**(*(X x 3) array\_like*) – List of coordinates against which to find studies.
- **r**(*float*, *optional*) – Radius (in mm) within which to find studies. Default is 20mm.

**Returns found\_ids** – A list of IDs from the Dataset with at least one focus within radius *r* of requested coordinates.

**Return type** *list*

**get\_studies\_by\_label**(*self*, *labels=None*, *label\_threshold=0.5*)

Extract list of studies with a given label.

**Parameters**

- **labels**(*list*, *optional*) – List of labels to use to search Dataset. If a contrast has all of the labels above the threshold, it will be returned. Default is None.
- **label\_threshold**(*float*, *optional*) – Default is 0.5.

**Returns found\_ids** – A list of IDs from the Dataset found by the search criteria.

**Return type** *list*

**get\_studies\_by\_mask**(*self*, *mask*)

Extract list of studies with at least one coordinate in mask.

**Parameters mask**(*img\_like*) – Mask across which to search for coordinates.

**Returns found\_ids** – A list of IDs from the Dataset with at least one focus in the mask.

**Return type** *list*

**get\_texts**(*self*, *ids=None*, *text\_type='abstract'*)

Extract list of texts of a given type for selected IDs.

**Parameters**

- **ids**(*list*, *optional*) – A list of IDs in the Dataset for which to find texts. Default is None, in which case all texts of requested type are returned.
- **text\_type**(*str*, *optional*) – Type of text to extract. Corresponds to column name in Dataset.texts DataFrame. Default is ‘abstract’.

**Returns texts** – List of texts of requested type for selected IDs.

**Return type** `list`

**classmethod** `load(filename, compressed=True)`

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** `class object`

**save** (`self, filename, compress=True`)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (`self, **params`)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**slice** (`self, ids`)

Return a reduced dataset with only requested IDs.

**Parameters** **ids** (`array_like`) – List of study IDs to include in new dataset

**Returns** **new\_dset** – Reduced Dataset containing only requested studies.

**Return type** `nimare.dataset.Dataset`

**update\_path** (`self, new_path`)

Update paths to images. Prepends new path to the relative path for files in `Dataset.images`.

**Parameters** **new\_path** (`str`) – Path to prepend to relative paths of files in `Dataset.images`.

## 6.2 `nimare.meta`: Meta-analytic algorithms

Coordinate-, image-, and effect-size-based meta-analysis estimators.

---

<code>nimare.meta</code>	Coordinate-, image-, and effect-size-based meta-analysis estimators.
<code>nimare.meta.esma</code>	Effect-size meta-analysis functions
<code>nimare.meta.ibma</code>	Image-based meta-analysis estimators

---

Continued on next page

Table 3 – continued from previous page

<code>nimare.meta.cbma.kernel</code>	Methods for estimating thresholded cluster maps from neuroimaging contrasts (Contrasts) from sets of foci and optional additional information (e.g., sample size and test statistic values).
<code>nimare.meta.cbma.ale</code>	CBMA methods from the activation likelihood estimation (ALE) family
<code>nimare.meta.cbma.mkda</code>	CBMA methods from the multilevel kernel density analysis (MKDA) family
<code>nimare.meta.cbma.model</code>	Model-based coordinate-based meta-analysis estimators

### 6.2.1 nimare.meta.esma

Effect-size meta-analysis functions

#### Functions

<code>fishers(z_maps[, two_sided])</code>	Run a Fisher’s image-based meta-analysis on z-statistics.
<code>rfx_glm(con_maps[, null, n_iters, two_sided])</code>	Run a random-effects (RFX) GLM on contrast maps.
<code>stouffers(z_maps[, inference, null, ...])</code>	Run a Stouffer’s image-based meta-analysis on z-statistic maps.
<code>weighted_stouffers(z_maps, sample_sizes[, ...])</code>	Run a Stouffer’s image-based meta-analysis on z-statistic maps.

#### `nimare.meta.esma.fishers`

**fishers** (`z_maps`, `two_sided=True`)

Run a Fisher’s image-based meta-analysis on z-statistics.

##### Parameters

- **z\_maps** ((`n_contrasts`, `n_voxels`) `numpy.ndarray`) – A 2D array of z-statistics.
- **two\_sided** (`bool`, optional) – Whether to do a two- or one-sided test. Default is `True`.

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

#### `nimare.meta.esma.rfx_glm`

**rfx\_glm** (`con_maps`, `null='theoretical'`, `n_iters=None`, `two_sided=True`)

Run a random-effects (RFX) GLM on contrast maps.

##### Parameters

- **con\_maps** ((`n_contrasts`, `n_voxels`) `numpy.ndarray`) – A 2D array of contrast maps in the same space, after masking.
- **null** (`{'theoretical', 'empirical'}`, optional) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Default is ‘theoretical’.

- **n\_iters** (`int` or `None`, optional) – The number of iterations to run in estimating the null distribution. Only used if `null = 'empirical'`.
- **two\_sided** (`bool`, optional) – Whether to do a two- or one-sided test. Default is `True`.

**Returns** **result** – Dictionary object containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

#### `nimare.meta.esma.stouffers`

**stouffers** (`z_maps`, `inference='ffx'`, `null='theoretical'`, `n_iters=None`, `two_sided=True`)

Run a Stouffer's image-based meta-analysis on z-statistic maps.

##### Parameters

- **z\_maps** ((`n_contrasts`, `n_voxels`) `numpy.ndarray`) – A 2D array of z-statistic maps in the same space, after masking.
- **inference** (`{'ffx', 'rfx'}`, optional) – Whether to use fixed-effects inference (default) or random-effects inference.
- **null** (`{'theoretical', 'empirical'}`, optional) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Empirical null is only possible if `inference = 'rfx'`.
- **n\_iters** (`int` or `None`, optional) – The number of iterations to run in estimating the null distribution. Only used if `inference = 'rfx'` and `null = 'empirical'`.
- **two\_sided** (`bool`, optional) – Whether to do a two- or one-sided test. Default is `True`.

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

#### `nimare.meta.esma.weighted_stouffers`

**weighted\_stouffers** (`z_maps`, `sample_sizes`, `two_sided=True`)

Run a Stouffer's image-based meta-analysis on z-statistic maps.

##### Parameters

- **z\_maps** ((`n_contrasts`, `n_voxels`) `numpy.ndarray`) – A 2D array of z-statistic maps in the same space, after masking.
- **sample\_sizes** ((`n_contrasts`,) `numpy.ndarray`) – A 1D array of sample sizes associated with contrasts in `z_maps`. Must be in same order as rows in `z_maps`.
- **two\_sided** (`bool`, optional) – Whether to do a two- or one-sided test. Default is `True`.

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

**fishers** (`z_maps`, `two_sided=True`)

Run a Fisher's image-based meta-analysis on z-statistics.

##### Parameters

- **z\_maps** ((`n_contrasts`, `n_voxels`) `numpy.ndarray`) – A 2D array of z-statistics.
- **two\_sided** (`bool`, optional) – Whether to do a two- or one-sided test. Default is `True`.

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

**rfx\_glm** (*con\_maps*, *null*='theoretical', *n\_iters*=None, *two\_sided*=True)

Run a random-effects (RFX) GLM on contrast maps.

**Parameters**

- **con\_maps** ((*n\_contrasts*, *n\_voxels*) `numpy.ndarray`) – A 2D array of contrast maps in the same space, after masking.
- **null** ({'theoretical', 'empirical'}, *optional*) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Default is 'theoretical'.
- **n\_iters** (`int` or `None`, *optional*) – The number of iterations to run in estimating the null distribution. Only used if `null` = 'empirical'.
- **two\_sided** (`bool`, *optional*) – Whether to do a two- or one-sided test. Default is True.

**Returns** **result** – Dictionary object containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

**stouffers** (*z\_maps*, *inference*='ffx', *null*='theoretical', *n\_iters*=None, *two\_sided*=True)

Run a Stouffer's image-based meta-analysis on z-statistic maps.

**Parameters**

- **z\_maps** ((*n\_contrasts*, *n\_voxels*) `numpy.ndarray`) – A 2D array of z-statistic maps in the same space, after masking.
- **inference** ({'ffx', 'rfx'}, *optional*) – Whether to use fixed-effects inference (default) or random-effects inference.
- **null** ({'theoretical', 'empirical'}, *optional*) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Empirical null is only possible if `inference` = 'rfx'.
- **n\_iters** (`int` or `None`, *optional*) – The number of iterations to run in estimating the null distribution. Only used if `inference` = 'rfx' and `null` = 'empirical'.
- **two\_sided** (`bool`, *optional*) – Whether to do a two- or one-sided test. Default is True.

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

**weighted\_stouffers** (*z\_maps*, *sample\_sizes*, *two\_sided*=True)

Run a Stouffer's image-based meta-analysis on z-statistic maps.

**Parameters**

- **z\_maps** ((*n\_contrasts*, *n\_voxels*) `numpy.ndarray`) – A 2D array of z-statistic maps in the same space, after masking.
- **sample\_sizes** ((*n\_contrasts*,) `numpy.ndarray`) – A 1D array of sample sizes associated with contrasts in `z_maps`. Must be in same order as rows in `z_maps`.
- **two\_sided** (`bool`, *optional*) – Whether to do a two- or one-sided test. Default is True.

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

## 6.2.2 nimare.meta.ibma

Image-based meta-analysis estimators

### Classes

<i>FFX_GLM</i> ([cdt, q, two_sided])	An image-based meta-analytic test using contrast and standard error images.
<i>Fishers</i> ([two_sided])	An image-based meta-analytic test using t- or z-statistic images.
<i>IBMAEstimator</i> (*args, **kwargs)	Base class for image-based meta-analysis methods.
<i>MFX_GLM</i> ([cdt, q, two_sided])	The gold standard image-based meta-analytic test.
<i>RFX_GLM</i> ([null, n_iters, two_sided])	A t-test on contrast images.
<i>Stouffers</i> ([inference, null, n_iters, two_sided])	A t-test on z-statistic images.
<i>WeightedStouffers</i> ([two_sided])	An image-based meta-analytic test using z-statistic images and sample sizes.

### `nimare.meta.ibma.FFX_GLM`

**class FFX\_GLM** (*cdt=0.01, q=0.05, two\_sided=True, \*args, \*\*kwargs*)

An image-based meta-analytic test using contrast and standard error images. Don't estimate variance, just take from first level.

#### Parameters

- **cdt** (*float*, optional) – Cluster-defining p-value threshold.
- **q** (*float*, optional) – Alpha for multiple comparisons correction.
- **two\_sided** (*bool*, optional) – Whether analysis should be two-sided (True) or one-sided (False).

#### Methods

<i>fit</i> (self, dataset)	Fit Estimator to Dataset.
<i>get_params</i> (self[, deep])	Get parameters for this estimator.
<i>load</i> (filename[, compressed])	Load a pickled class instance from file.
<i>save</i> (self, filename[, compress])	Pickle the class instance to the provided file.
<i>set_params</i> (self, **params)	Set the parameters of this estimator.

**fit** (*self, dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self, deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.



**Returns** `params` – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load(filename, compressed=True)`

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (`self, filename, compress=True`)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (`self, **params`)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

## `nimare.meta.ibma.Fishers`

**class** `Fishers(two_sided=True, *args, **kwargs)`

An image-based meta-analytic test using t- or z-statistic images. Requires z-statistic images, but will be extended to work with t-statistic images as well.

**Parameters** `two_sided` (`bool`, optional) – Whether to do a two- or one-sided test. Default is True.

## Notes

Sum of -log P-values (from T/Zs converted to Ps)

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.

Continued on next page

Table 7 – continued from previous page

<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

## *nimare.meta.ibma*.IBMAEstimator

**class IBMAEstimator** (*\*args*, *\*\*kwargs*)

Base class for image-based meta-analysis methods.

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**nimare.meta.ibma.MFX\_GLM****class MFX\_GLM** (*cdt=0.01, q=0.05, two\_sided=True, \*args, \*\*kwargs*)

The gold standard image-based meta-analytic test. Uses contrast and standard error images.

**Parameters**

- **cdt** (*float*, optional) – Cluster-defining p-value threshold.
- **q** (*float*, optional) – Alpha for multiple comparisons correction.
- **two\_sided** (*bool*, optional) – Whether analysis should be two-sided (True) or one-sided (False).

**Methods**

<i>fit</i> (self, dataset)	Fit Estimator to Dataset.
<i>get_params</i> (self[, deep])	Get parameters for this estimator.
<i>load</i> (filename[, compressed])	Load a pickled class instance from file.
<i>save</i> (self, filename[, compress])	Pickle the class instance to the provided file.
<i>set_params</i> (self, <i>\\*\\*params</i> )	Set the parameters of this estimator.

**fit** (*self, dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.**Returns** Results of Estimator fitting.**Return type** *nimare.base.base.MetaResult***get\_params** (*self, deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.**Returns** **params** – Parameter names mapped to their values.**Return type** mapping of string to any**classmethod load** (*filename, compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.**Return type** class object**save** (*self, filename, compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.

- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

## nimare.meta.ibma.RFX\_GLM

**class RFX\_GLM** (*null='theoretical', n\_iters=None, two\_sided=True, \*args, \*\*kwargs*)

A t-test on contrast images. Requires contrast images.

### Parameters

- **null** ({*'theoretical', 'empirical'*}, *optional*) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Default is 'theoretical'.
- **n\_iters** (*int* or *None*, *optional*) – The number of iterations to run in estimating the null distribution. Only used if *null = 'empirical'*.
- **two\_sided** (*bool*, *optional*) – Whether to do a two- or one-sided test. Default is True.

### Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**nimare.meta.ibma.Stouffers**

**class Stouffers** (*inference='ffx'*, *null='theoretical'*, *n\_iters=None*, *two\_sided=True*, *\*args*, *\*\*kwargs*)

A t-test on z-statistic images. Requires z-statistic images.

**Parameters**

- **inference** ({*'ffx'*, *'rfx'*}, *optional*) – Whether to use fixed-effects inference (default) or random-effects inference.
- **null** ({*'theoretical'*, *'empirical'*}, *optional*) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Empirical null is only possible if *inference = 'rfx'*.
- **n\_iters** (*int* or *None*, *optional*) – The number of iterations to run in estimating the null distribution. Only used if *inference = 'rfx'* and *null = 'empirical'*.
- **two\_sided** (*bool*, *optional*) – Whether to do a two- or one-sided test. Default is True.

**Methods**

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

## **`nimare.meta.ibma.WeightedStouffers`**

**class WeightedStouffers** (*two\_sided=True*, *\*args*, *\*\*kwargs*)

An image-based meta-analytic test using z-statistic images and sample sizes. Zs from bigger studies get bigger weights.

**Parameters** *two\_sided* (*bool*, *optional*) – Whether to do a two- or one-sided test. Default is True.

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*



## Functions

<code>ffx_glm(con_maps, se_maps, sample_sizes, mask)</code>	Run a fixed-effects GLM on contrast and standard error images.
<code>fsl_glm(con_maps, se_maps, sample_sizes, ...)</code>	Run a GLM with FSL.
<code>mfx_glm(con_maps, se_maps, sample_sizes, mask)</code>	Run a mixed-effects GLM on contrast and standard error images.

### `nimare.meta.ibma.ffx_glm`

**ffx\_glm**(*con\_maps*, *se\_maps*, *sample\_sizes*, *mask*, *cdt*=0.01, *q*=0.05, *work\_dir*='ffx\_glm', *two\_sided*=True)  
Run a fixed-effects GLM on contrast and standard error images.

#### Parameters

- **con\_maps** ((n\_contrasts, n\_voxels) `numpy.ndarray`) – A 2D array of contrast maps in the same space, after masking.
- **var\_maps** ((n\_contrasts, n\_voxels) `numpy.ndarray`) – A 2D array of contrast standard error maps in the same space, after masking. Must match shape and order of `con_maps`.
- **sample\_sizes** ((n\_contrasts,) `numpy.ndarray`) – A 1D array of sample sizes associated with contrasts in `con_maps` and `var_maps`. Must be in same order as rows in `con_maps` and `var_maps`.
- **mask** (`nibabel.Nifti1Image`) – Mask image, used to unmask results maps in compiling output.
- **cdt** (`float`, optional) – Cluster-defining p-value threshold.
- **q** (`float`, optional) – Alpha for multiple comparisons correction.
- **work\_dir** (`str`, optional) – Working directory for FSL flameo outputs.
- **two\_sided** (`bool`, optional) – Whether analysis should be two-sided (True) or one-sided (False).

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

### `nimare.meta.ibma.fsl_glm`

**fsl\_glm**(*con\_maps*, *se\_maps*, *sample\_sizes*, *mask*, *inference*, *cdt*=0.01, *q*=0.05, *work\_dir*='fsl\_glm', *two\_sided*=True)  
Run a GLM with FSL.

### `nimare.meta.ibma.mfx_glm`

**mfx\_glm**(*con\_maps*, *se\_maps*, *sample\_sizes*, *mask*, *cdt*=0.01, *q*=0.05, *work\_dir*='mfx\_glm', *two\_sided*=True)  
Run a mixed-effects GLM on contrast and standard error images.

#### Parameters

- **con\_maps** ((n\_contrasts, n\_voxels) `numpy.ndarray`) – A 2D array of contrast maps in the same space, after masking.

- **var\_maps** ((n\_contrasts, n\_voxels) `numpy.ndarray`) – A 2D array of contrast standard error maps in the same space, after masking. Must match shape and order of `con_maps`.
- **sample\_sizes** ((n\_contrasts,) `numpy.ndarray`) – A 1D array of sample sizes associated with contrasts in `con_maps` and `var_maps`. Must be in same order as rows in `con_maps` and `var_maps`.
- **mask** (`nibabel.Nifti1Image`) – Mask image, used to unmask results maps in compiling output.
- **cdt** (`float`, optional) – Cluster-defining p-value threshold.
- **q** (`float`, optional) – Alpha for multiple comparisons correction.
- **work\_dir** (`str`, optional) – Working directory for FSL flameo outputs.
- **two\_sided** (`bool`, optional) – Whether analysis should be two-sided (True) or one-sided (False).

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

**class** **FFX\_GLM** (`cdt=0.01, q=0.05, two_sided=True, *args, **kwargs`)

An image-based meta-analytic test using contrast and standard error images. Don't estimate variance, just take from first level.

#### Parameters

- **cdt** (`float`, optional) – Cluster-defining p-value threshold.
- **q** (`float`, optional) – Alpha for multiple comparisons correction.
- **two\_sided** (`bool`, optional) – Whether analysis should be two-sided (True) or one-sided (False).

#### Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (`self, dataset`)

Fit Estimator to Dataset.

**Parameters** **dataset** (`nimare.dataset.Dataset`) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (`self, deep=True`)

Get parameters for this estimator.

**Parameters** **deep** (`boolean, optional`) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** *self*

**class Fishers** (*two\_sided=True*, *\*args*, *\*\*kwargs*)

An image-based meta-analytic test using t- or z-statistic images. Requires z-statistic images, but will be extended to work with t-statistic images as well.

**Parameters** **two\_sided** (*bool*, optional) – Whether to do a two- or one-sided test. Default is True.

## Notes

Sum of -log P-values (from T/Zs converted to Ps)

## Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**class IBMAEstimator** (*\*args*, *\*\*kwargs*)

Base class for image-based meta-analysis methods.

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**class MFX\_GLM** (*cdt=0.01*, *q=0.05*, *two\_sided=True*, *\*args*, *\*\*kwargs*)

The gold standard image-based meta-analytic test. Uses contrast and standard error images.

**Parameters**

- **cdt** (*float*, optional) – Cluster-defining p-value threshold.
- **q** (*float*, optional) – Alpha for multiple comparisons correction.
- **two\_sided** (*bool*, optional) – Whether analysis should be two-sided (True) or one-sided (False).

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**class RFX\_GLM** (*null='theoretical', n\_iters=None, two\_sided=True, \*args, \*\*kwargs*)

A t-test on contrast images. Requires contrast images.

#### Parameters

- **null** (*{'theoretical', 'empirical'}, optional*) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Default is 'theoretical'.
- **n\_iters** (*int or None, optional*) – The number of iterations to run in estimating the null distribution. Only used if `null = 'empirical'`.
- **two\_sided** (*bool, optional*) – Whether to do a two- or one-sided test. Default is True.

#### Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self, dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self, deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename, compressed=True*)

Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool, optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self, filename, compress=True*)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (*str*) – File to which object will be saved.

- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

### Returns

**Return type** *self*

**class Stouffers** (*inference='ffx'*, *null='theoretical'*, *n\_iters=None*, *two\_sided=True*, *\*args*, *\*\*kwargs*)

A t-test on z-statistic images. Requires z-statistic images.

### Parameters

- **inference** ({*'ffx'*, *'rfx'*}, optional) – Whether to use fixed-effects inference (default) or random-effects inference.
- **null** ({*'theoretical'*, *'empirical'*}, optional) – Whether to use a theoretical null T distribution or an empirically- derived null distribution determined via sign flipping. Empirical null is only possible if *inference = 'rfx'*.
- **n\_iters** (*int* or *None*, optional) – The number of iterations to run in estimating the null distribution. Only used if *inference = 'rfx'* and *null = 'empirical'*.
- **two\_sided** (*bool*, optional) – Whether to do a two- or one-sided test. Default is True.

### Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>\**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

### Parameters



- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**class WeightedStouffers** (*two\_sided=True*, *\*args*, *\*\*kwargs*)

An image-based meta-analytic test using z-statistic images and sample sizes. Zs from bigger studies get bigger weights.

**Parameters** **two\_sided** (*bool*, optional) – Whether to do a two- or one-sided test. Default is True.

## Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>\**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** `params` – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load(filename, compressed=True)`

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (`self, filename, compress=True`)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (`self, **params`)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**ffx\_glm** (`con_maps, se_maps, sample_sizes, mask, cdt=0.01, q=0.05, work_dir='ffx_glm', two_sided=True`)

Run a fixed-effects GLM on contrast and standard error images.

**Parameters**

- **con\_maps** ((`n_contrasts, n_voxels`) `numpy.ndarray`) – A 2D array of contrast maps in the same space, after masking.
- **var\_maps** ((`n_contrasts, n_voxels`) `numpy.ndarray`) – A 2D array of contrast standard error maps in the same space, after masking. Must match shape and order of `con_maps`.
- **sample\_sizes** ((`n_contrasts,`) `numpy.ndarray`) – A 1D array of sample sizes associated with contrasts in `con_maps` and `var_maps`. Must be in same order as rows in `con_maps` and `var_maps`.
- **mask** (`nibabel.Nifti1Image`) – Mask image, used to unmask results maps in compiling output.
- **cdt** (`float`, optional) – Cluster-defining p-value threshold.
- **q** (`float`, optional) – Alpha for multiple comparisons correction.
- **work\_dir** (`str`, optional) – Working directory for FSL flameo outputs.
- **two\_sided** (`bool`, optional) – Whether analysis should be two-sided (True) or one-sided (False).

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

**fsl\_glm**(*con\_maps*, *se\_maps*, *sample\_sizes*, *mask*, *inference*, *cdt*=0.01, *q*=0.05, *work\_dir*='fsl\_glm', *two\_sided*=True)  
Run a GLM with FSL.

**mfx\_glm**(*con\_maps*, *se\_maps*, *sample\_sizes*, *mask*, *cdt*=0.01, *q*=0.05, *work\_dir*='mfx\_glm', *two\_sided*=True)  
Run a mixed-effects GLM on contrast and standard error images.

#### Parameters

- **con\_maps** ((n\_contrasts, n\_voxels) `numpy.ndarray`) – A 2D array of contrast maps in the same space, after masking.
- **var\_maps** ((n\_contrasts, n\_voxels) `numpy.ndarray`) – A 2D array of contrast standard error maps in the same space, after masking. Must match shape and order of `con_maps`.
- **sample\_sizes** ((n\_contrasts,) `numpy.ndarray`) – A 1D array of sample sizes associated with contrasts in `con_maps` and `var_maps`. Must be in same order as rows in `con_maps` and `var_maps`.
- **mask** (`nibabel.Nifti1Image`) – Mask image, used to unmask results maps in compiling output.
- **cdt** (`float`, optional) – Cluster-defining p-value threshold.
- **q** (`float`, optional) – Alpha for multiple comparisons correction.
- **work\_dir** (`str`, optional) – Working directory for FSL flameo outputs.
- **two\_sided** (`bool`, optional) – Whether analysis should be two-sided (True) or one-sided (False).

**Returns** **result** – Dictionary containing maps for test statistics, p-values, and negative log(p) values.

**Return type** `dict`

## 6.2.3 nimare.meta.cbma.kernel

Methods for estimating thresholded cluster maps from neuroimaging contrasts (Contrasts) from sets of foci and optional additional information (e.g., sample size and test statistic values).

NOTE: Currently imagining output from “dataset.get\_coordinates” as a DataFrame of peak coords and sample sizes/statistics (a la Neurosynth).

### Classes

<code>ALEKernel</code> ([fwhm, n])	Generate ALE modeled activation images from coordinates and sample size.
<code>KDAKernel</code> ([r, value])	Generate KDA modeled activation images from coordinates.
<code>KernelTransformer</code> ()	Base class for modeled activation-generating methods.
<code>MKDAKernel</code> ([r, value])	Generate MKDA modeled activation images from coordinates.

Continued on next page

Table 21 – continued from previous page

<code>Peaks2MapsKernel([resample_to_mask])</code>	Generate peaks2maps modeled activation images from coordinates.
---	---

**nimare.meta.cbma.kernel.ALEKernel****class ALEKernel** (*fwhm=None, n=None*)

Generate ALE modeled activation images from coordinates and sample size.

**Parameters**

- **fwhm** (*float*, optional) – Full-width half-max for Gaussian kernel, if you want to have a constant kernel across Contrasts. Mutually exclusive with *n*.
- **n** (*int*, optional) – Sample size, used to derive FWHM for Gaussian kernel based on formulae from Eickhoff et al. (2012). This sample size overwrites the Contrast-specific sample sizes in the dataset, in order to hold kernel constant across Contrasts. Mutually exclusive with *fwhm*.

**Methods**

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.
<code>transform(self, dataset[, mask, masked])</code>	Generate ALE modeled activation images for each Contrast in dataset.

**get\_params** (*self, deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename, compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self, filename, compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.

- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**transform** (*self*, *dataset*, *mask=None*, *masked=False*)

Generate ALE modeled activation images for each Contrast in dataset.

**Parameters**

- **dataset** (*nimare.dataset.Dataset* or *pandas.DataFrame*) – Dataset for which to make images. Can be a DataFrame if necessary.
- **mask** (*img\_like*, optional) – Only used if dataset is a DataFrame.
- **masked** (*bool*, optional) – Return an array instead of a niimg.

**Returns** *imgs* – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** *list* of *nibabel.Nifti1Image*

## **nimare.meta.cbma.kernel.KDAKernel**

**class KDAKernel** (*r=6*, *value=1*)

Generate KDA modeled activation images from coordinates.

**Parameters**

- **r** (*int*, optional) – Sphere radius, in mm.
- **value** (*int*, optional) – Value for sphere.

## **Methods**

<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.
<i>transform</i> ( <i>self</i> , <i>dataset</i> [, <i>mask</i> , <i>masked</i> ])	Generate KDA modeled activation images for each Contrast in dataset.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load(filename, compressed=True)`

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (`self, filename, compress=True`)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (`self, **params`)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**transform** (`self, dataset, mask=None, masked=False`)

Generate KDA modeled activation images for each Contrast in dataset. Differs from MKDA images in that binary spheres are summed together in map (i.e., resulting image is not binary if coordinates are close to one another).

**Parameters**

- **dataset** (`nimare.dataset.Dataset` or `pandas.DataFrame`) – Dataset for which to make images. Can be a DataFrame if necessary.
- **mask** (`img_like`, optional) – Only used if dataset is a DataFrame.
- **masked** (`bool`, optional) – Return an array instead of a niimg.

**Returns** `imgs` – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** `list` of `nibabel.Nifti1Image`

## `nimare.meta.cbma.kernel`.KernelTransformer

**class** `KernelTransformer`

Base class for modeled activation-generating methods.

Coordinate-based meta-analyses leverage coordinates reported in neuroimaging papers to simulate the thresholded statistical maps from the original analyses. This generally involves convolving each coordinate with a kernel (typically a Gaussian or binary sphere) that may be weighted based on some additional measure, such as statistic value or sample size.

## Methods

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.
<code>transform(self, dataset)</code>	Add stuff to transformer.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**transform** (*self*, *dataset*)

Add stuff to transformer.

**nimare.meta.cbma.kernel.MKDAKernel**

**class** **MKDAKernel** (*r=10*, *value=1*)

Generate MKDA modeled activation images from coordinates.

**Parameters**

- **r** (`int`, optional) – Sphere radius, in mm.
- **value** (`int`, optional) – Value for sphere.

**Methods**

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.
<code>transform(self, dataset[, mask, masked])</code>	Generate MKDA modeled activation images for each Contrast in dataset.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*



**transform** (*self*, *dataset*, *mask=None*, *masked=False*)

Generate MKDA modeled activation images for each Contrast in dataset. For each Contrast, a binary sphere of radius *r* is placed around each coordinate. Voxels within overlapping regions between proximal coordinates are set to 1, rather than the sum.

#### Parameters

- **dataset** (*nimare.dataset.Dataset* or *pandas.DataFrame*) – Dataset for which to make images. Can be a DataFrame if necessary.
- **mask** (*img\_like*, *optional*) – Only used if dataset is a DataFrame.
- **masked** (*bool*, *optional*) – Return an array instead of a niimg.

**Returns** *imgs* – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** *list* of *nibabel.Nifti1Image*

### `nimare.meta.cbma.kernel.Peaks2MapsKernel`

**class Peaks2MapsKernel** (*resample\_to\_mask=True*)

Generate peaks2maps modeled activation images from coordinates.

#### Methods

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.
<code>transform(self, dataset[, mask, masked])</code>	Generate peaks2maps modeled activation images for each Contrast in dataset.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** *self*

**transform** (*self*, *dataset*, *mask=None*, *masked=False*)

Generate peaks2maps modeled activation images for each Contrast in dataset.

#### Parameters

- **ids** (*list*) – A list of Contrast IDs for which to generate modeled activation images.
- **masked** (*boolean*) – Whether to mask the maps generated by peaks2maps

**Returns** *imgs* – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** *list* of *nibabel.Nifti1Image*

**class ALEKernel** (*fwhm=None*, *n=None*)

Generate ALE modeled activation images from coordinates and sample size.

#### Parameters

- **fwhm** (*float*, optional) – Full-width half-max for Gaussian kernel, if you want to have a constant kernel across Contrasts. Mutually exclusive with *n*.
- **n** (*int*, optional) – Sample size, used to derive FWHM for Gaussian kernel based on formulae from Eickhoff et al. (2012). This sample size overwrites the Contrast-specific sample sizes in the dataset, in order to hold kernel constant across Contrasts. Mutually exclusive with *fwhm*.

## Methods

<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.
<i>transform</i> ( <i>self</i> , <i>dataset</i> [, <i>mask</i> , <i>masked</i> ])	Generate ALE modeled activation images for each Contrast in dataset.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this esti-

mator and contained subobjects that are estimators.

**Returns** `params` – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load(filename, compressed=True)`

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (`self, filename, compress=True`)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (`self, **params`)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**transform** (`self, dataset, mask=None, masked=False`)

Generate ALE modeled activation images for each Contrast in dataset.

**Parameters**

- **dataset** (`nimare.dataset.Dataset` or `pandas.DataFrame`) – Dataset for which to make images. Can be a DataFrame if necessary.
- **mask** (`img_like`, optional) – Only used if dataset is a DataFrame.
- **masked** (`bool`, optional) – Return an array instead of a niimg.

**Returns** `imgs` – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** `list` of `nibabel.Nifti1Image`

**class** `MKDAKernel(r=10, value=1)`

Generate MKDA modeled activation images from coordinates.

**Parameters**

- **r** (`int`, optional) – Sphere radius, in mm.
- **value** (`int`, optional) – Value for sphere.

## Methods

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.
<code>transform(self, dataset[, mask, masked])</code>	Generate MKDA modeled activation images for each Contrast in dataset.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**transform** (*self*, *dataset*, *mask=None*, *masked=False*)

Generate MKDA modeled activation images for each Contrast in dataset. For each Contrast, a binary sphere of radius *r* is placed around each coordinate. Voxels within overlapping regions between proximal coordinates are set to 1, rather than the sum.

**Parameters**

- **dataset** (*nimare.dataset.Dataset* or *pandas.DataFrame*) – Dataset for which to make images. Can be a DataFrame if necessary.
- **mask** (*img\_like*, *optional*) – Only used if dataset is a DataFrame.
- **masked** (*bool*, *optional*) – Return an array instead of a niimg.

**Returns** **imgs** – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** *list* of *nibabel.Nifti1Image*

**class** **KDAKernel** (*r=6*, *value=1*)

Generate KDA modeled activation images from coordinates.

#### Parameters

- **r** (*int*, *optional*) – Sphere radius, in mm.
- **value** (*int*, *optional*) – Value for sphere.

#### Methods

<i>get_params</i> (self[, deep])	Get parameters for this estimator.
<i>load</i> (filename[, compressed])	Load a pickled class instance from file.
<i>save</i> (self, filename[, compress])	Pickle the class instance to the provided file.
<i>set_params</i> (self, **params)	Set the parameters of this estimator.
<i>transform</i> (self, dataset[, mask, masked])	Generate KDA modeled activation images for each Contrast in dataset.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (*str*) – File to which object will be saved.

- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** *self*

**transform** (*self*, *dataset*, *mask=None*, *masked=False*)

Generate KDA modeled activation images for each Contrast in dataset. Differs from MKDA images in that binary spheres are summed together in map (i.e., resulting image is not binary if coordinates are close to one another).

#### Parameters

- **dataset** (*nimare.dataset.Dataset* or *pandas.DataFrame*) – Dataset for which to make images. Can be a DataFrame if necessary.
- **mask** (*img\_like*, optional) – Only used if dataset is a DataFrame.
- **masked** (*bool*, optional) – Return an array instead of a niimg.

**Returns** *imgs* – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** *list* of *nibabel.Nifti1Image*

**class Peaks2MapsKernel** (*resample\_to\_mask=True*)

Generate peaks2maps modeled activation images from coordinates.

## Methods

<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>\**params</i> )	Set the parameters of this estimator.
<i>transform</i> ( <i>self</i> , <i>dataset</i> [, <i>mask</i> , <i>masked</i> ])	Generate peaks2maps modeled activation images for each Contrast in dataset.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.

- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)  
Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)  
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**transform** (*self*, *dataset*, *mask=None*, *masked=False*)  
Generate peaks2maps modeled activation images for each Contrast in dataset.

**Parameters**

- **ids** (`list`) – A list of Contrast IDs for which to generate modeled activation images.
- **masked** (`boolean`) – Whether to mask the maps generated by peaks2maps

**Returns** `imgs` – A list of modeled activation images (one for each of the Contrasts in the input dataset).

**Return type** `list` of `nibabel.Nifti1Image`

## 6.2.4 nimare.meta.cbma.ale

CBMA methods from the activation likelihood estimation (ALE) family

### Classes

<code>ALE([kernel_estimator])</code>	Activation likelihood estimation
<code>ALESubtraction([n_iters])</code>	ALE subtraction analysis.
<code>SCALE([voxel_thresh, n_iters, n_cores, ijk, ...])</code>	Specific coactivation likelihood estimation [Re194cb71ed63-1].

### `nimare.meta.cbma.ale.ALE`

**class** **ALE** (*kernel\_estimator=<class 'nimare.meta.cbma.kernel.ALEKernel'>*, *\*\*kwargs*)  
Activation likelihood estimation

**Parameters**

- **kernel\_estimator** (`nimare.meta.cbma.base.KernelTransformer`, optional) – Kernel with which to convolve coordinates from dataset. Default is ALEKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the `kernel_estimator` can be assigned here, with the prefix ‘`kernel__`’ in the variable name.

## Notes

The ALE algorithm was originally developed in [Rbfa6233af19f-1], then updated in [Rbfa6233af19f-2] and [Rbfa6233af19f-3].

## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.

- **compressed** (*bool, optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**



- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** *self*

### `nimare.meta.cbma.ale.ALESubtraction`

**class ALESubtraction** (*n\_iters=10000*)

ALE subtraction analysis.

**Parameters** *n\_iters* (*int*, optional) – Default is 10000.

#### Notes

This method was originally developed in [Rfe11a1766f4a-1] and refined in [Rfe11a1766f4a-2].

#### References

#### Methods

<code>fit(self, ale1, ale2[, image1, image2, ...])</code>	Run a subtraction analysis comparing two groups of experiments from separate meta-analyses.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, <i>\\*\\*</i>params)</code>	Set the parameters of this estimator.

**fit** (*self*, *ale1*, *ale2*, *image1=None*, *image2=None*, *ma\_maps1=None*, *ma\_maps2=None*)

Run a subtraction analysis comparing two groups of experiments from separate meta-analyses.

#### Parameters

- **ale2** (*ale1*,) – Fitted ALE models for datasets to compare.
- **image2** (*image1*,) – Cluster-level FWE-corrected z-statistic maps associated with the respective models.
- **ma\_maps1** (*(E x V) array\_like or None, optional*) – Experiments by voxels array of modeled activation values. If not provided, MA maps will be generated from dataset1.
- **ma\_maps2** (*(E x V) array\_like or None, optional*) – Experiments by voxels array of modeled activation values. If not provided, MA maps will be generated from dataset2.

**Returns** Results of ALE subtraction analysis.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

## `nimare.meta.cbma.ale.SCALE`

**class SCALE** (*voxel\_thresh=0.001*, *n\_iters=10000*, *n\_cores=-1*, *ijk=None*, *kernel\_estimator=<class 'nimare.meta.cbma.kernel.ALEKernel'>*, *\*\*kwargs*)

Specific coactivation likelihood estimation [Re194cb71ed63-1].

**Parameters**

- **voxel\_thresh** (*float*, *optional*) – Uncorrected voxel-level threshold. Default: 0.001
- **n\_iters** (*int*, *optional*) – Number of iterations for correction. Default: 10000
- **n\_cores** (*int*, *optional*) – Number of processes to use for meta-analysis. If -1, use all available cores. Default: -1

- **ijk** (`str` or  $(N \times 3)$  `array_like`) – Tab-delimited file of coordinates from database or numpy array with ijk coordinates. Voxels are rows and i, j, k (meaning matrix-space) values are the three columns.
- **kernel\_estimator** (`nimare.meta.cbma.base.KernelTransformer`, optional) – Kernel with which to convolve coordinates from dataset. Default is ALEKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the `kernel_estimator` can be assigned here, with the prefix ‘kernel\_\_’ in the variable name.

## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **kwargs)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (`nimare.dataset.Dataset`) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (`boolean`, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.

- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** *self*

**class ALE** (*kernel\_estimator=<class 'nimare.meta.cbma.kernel.ALEKernel'>*, *\*\*kwargs*)

Activation likelihood estimation

#### Parameters

- **kernel\_estimator** (*nimare.meta.cbma.base.KernelTransformer*, optional) – Kernel with which to convolve coordinates from dataset. Default is ALEKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the *kernel\_estimator* can be assigned here, with the prefix 'kernel\_\_' in the variable name.

## Notes

The ALE algorithm was originally developed in [Rbfa6233af19f-1], then updated in [Rbfa6233af19f-2] and [Rbfa6233af19f-3].

## References

## Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load(filename, compressed=True)`

Load a pickled class instance from file.

#### Parameters

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (`self, filename, compress=True`)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (`self, **params`)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

#### Returns

**Return type** `self`

**class** `ALESubtraction(n_iters=10000)`

ALE subtraction analysis.

**Parameters** `n_iters` (`int`, optional) – Default is 10000.

## Notes

This method was originally developed in [Rfe11a1766f4a-1] and refined in [Rfe11a1766f4a-2].

## References

## Methods

<code>fit(self, ale1, ale2[, image1, image2, ...])</code>	Run a subtraction analysis comparing two groups of experiments from separate meta-analyses.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (`self, ale1, ale2, image1=None, image2=None, ma_maps1=None, ma_maps2=None`)

Run a subtraction analysis comparing two groups of experiments from separate meta-analyses.

**Parameters**

- **ale2** (*ale1*,) – Fitted ALE models for datasets to compare.
- **image2** (*image1*,) – Cluster-level FWE-corrected z-statistic maps associated with the respective models.
- **ma\_maps1** ((*E x V*) *array\_like* or *None*, *optional*) – Experiments by voxels array of modeled activation values. If not provided, MA maps will be generated from dataset1.
- **ma\_maps2** ((*E x V*) *array\_like* or *None*, *optional*) – Experiments by voxels array of modeled activation values. If not provided, MA maps will be generated from dataset2.

**Returns** Results of ALE subtraction analysis.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**class SCALE** (*voxel\_thresh=0.001, n\_iters=10000, n\_cores=-1, ijk=None, kernel\_estimator=<class 'nimare.meta.cbma.kernel.ALEKernel'>, \*\*kwargs*)  
 Specific coactivation likelihood estimation [Re194cb71ed63-1].

#### Parameters

- **voxel\_thresh** (*float, optional*) – Uncorrected voxel-level threshold. Default: 0.001
- **n\_iters** (*int, optional*) – Number of iterations for correction. Default: 10000
- **n\_cores** (*int, optional*) – Number of processes to use for meta-analysis. If -1, use all available cores. Default: -1
- **ijk** (*str* or (*N x 3*) *array\_like*) – Tab-delimited file of coordinates from database or numpy array with ijk coordinates. Voxels are rows and i, j, k (meaning matrix-space) values are the three columns.
- **kernel\_estimator** (*nimare.meta.cbma.base.KernelTransformer, optional*) – Kernel with which to convolve coordinates from dataset. Default is ALEKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the kernel\_estimator can be assigned here, with the prefix 'kernel\_\_' in the variable name.

#### References

#### Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self, dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self, deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename, compressed=True*)

Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.

- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)  
Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)  
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

## 6.2.5 nimare.meta.cbma.mkda

CBMA methods from the multilevel kernel density analysis (MKDA) family

### Classes

<code>KDA([kernel_estimator])</code>	Kernel density analysis.
<code>MKDACHi2([prior, kernel_estimator])</code>	Multilevel kernel density analysis- Chi-square analysis [ <a href="#">Rb50f9c63f995-1</a> ].
<code>MKDADensity([kernel_estimator])</code>	Multilevel kernel density analysis- Density analysis [ <a href="#">R8ac5f6c30bba-1</a> ].

### `nimare.meta.cbma.mkda.KDA`

**class** `KDA` (*kernel\_estimator=<class 'nimare.meta.cbma.kernel.KDAKernel'>*, *\*\*kwargs*)  
Kernel density analysis.

**Parameters**

- **kernel\_estimator** (`nimare.meta.cbma.base.KernelTransformer`, optional) – Kernel with which to convolve coordinates from dataset. Default is `KDAKernel`.
- **\*\*kwargs** – Keyword arguments. Arguments for the `kernel_estimator` can be assigned here, with the prefix `'kernel__'` in the variable name.

### Notes

Kernel density analysis was first introduced in [[R258c842da77f-1](#)] and [[R258c842da77f-2](#)].



## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** self

`nimare.meta.cbma.mkda.MKDACHi2`

```
class MKDACHi2 (prior=0.5, kernel_estimator=<class 'nimare.meta.cbma.kernel.MKDAKernel'>,
               **kwargs)
```

Multilevel kernel density analysis- Chi-square analysis [Rb50f9c63f995-1].

#### Parameters

- **prior** (*float*, *optional*) – Uniform prior probability of each feature being active in a map in the absence of evidence from the map. Default: 0.5
- **kernel\_estimator** (`nimare.meta.cbma.base.KernelTransformer`, *optional*) – Kernel with which to convolve coordinates from dataset. Default is MKDAKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the `kernel_estimator` can be assigned here, with the prefix ‘kernel\_\_’ in the variable name.

#### References

#### Methods

<code>fit(self, dataset, dataset2)</code>	Fit Estimator to datasets.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*, *dataset2*)  
Fit Estimator to datasets.

**Parameters** **dataset2** (*dataset*,) – Dataset objects to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)  
Get parameters for this estimator.

**Parameters** **deep** (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)  
Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

`nimare.meta.cbma.mkda.MKDADensity`

**class MKDADensity** (*kernel\_estimator=<class 'nimare.meta.cbma.kernel.MKDAKernel'>*, *\*\*kwargs*)

Multilevel kernel density analysis- Density analysis [R8ac5f6c30bba-1].

**Parameters**

- **kernel\_estimator** (`nimare.meta.cbma.base.KernelTransformer`, optional) – Kernel with which to convolve coordinates from dataset. Default is MKDAKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the `kernel_estimator` can be assigned here, with the prefix `'kernel__'` in the variable name.

## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, \**params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** `dataset` (`nimare.dataset.Dataset`) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** `deep` (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** `params` – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load` (*filename, compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool, optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self, filename, compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool, optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self, \*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**class** `KDA` (*kernel\_estimator=<class 'nimare.meta.cbma.kernel.KDAKernel'>, \*\*kwargs*)

Kernel density analysis.

**Parameters**

- **kernel\_estimator** (*nimare.meta.cbma.base.KernelTransformer, optional*) – Kernel with which to convolve coordinates from dataset. Default is `KDAKernel`.
- **\*\*kwargs** – Keyword arguments. Arguments for the `kernel_estimator` can be assigned here, with the prefix `'kernel__'` in the variable name.

## Notes

Kernel density analysis was first introduced in [R258c842da77f-1] and [R258c842da77f-2].

## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, <i>\\*\\*</i>params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, **\\\*\\\**params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**class MKDACHi2** (*prior=0.5*, *kernel\_estimator=<class 'nimare.meta.cbma.kernel.MKDAKernel'>*, **\\\*\\\**kargs*)

Multilevel kernel density analysis- Chi-square analysis [Rb50f9c63f995-1].

### Parameters

- **prior** (*float*, *optional*) – Uniform prior probability of each feature being active in a map in the absence of evidence from the map. Default: 0.5
- **kernel\_estimator** (*nimare.meta.cbma.base.KernelTransformer*, *optional*) – Kernel with which to convolve coordinates from dataset. Default is MKDAKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the `kernel_estimator` can be assigned here, with the prefix ‘`kernel__`’ in the variable name.

### References

### Methods

<code>fit(self, dataset, dataset2)</code>	Fit Estimator to datasets.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*, *dataset2*)  
Fit Estimator to datasets.

**Parameters** **dataset2** (*dataset*,) – Dataset objects to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)  
Get parameters for this estimator.

**Parameters** **deep** (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)  
Load a pickled class instance from file.

### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and `gzip` will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)  
Pickle the class instance to the provided file.

### Parameters

- **filename** (*str*) – File to which object will be saved.

- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** *self*

**class MKDADensity** (*kernel\_estimator=<class 'nimare.meta.cbma.kernel.MKDAKernel'>*, *\*\*kwargs*)

Multilevel kernel density analysis- Density analysis [R8ac5f6c30bba-1].

#### Parameters

- **kernel\_estimator** (*nimare.meta.cbma.base.KernelTransformer*, optional) – Kernel with which to convolve coordinates from dataset. Default is MKDAKernel.
- **\*\*kwargs** – Keyword arguments. Arguments for the kernel\_estimator can be assigned here, with the prefix 'kernel\_\_' in the variable name.

## References

## Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean*, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.

- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)  
Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)  
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

## 6.2.6 nimare.meta.cbma.model

Model-based coordinate-based meta-analysis estimators

### Classes

<code>BHICP()</code>	Bayesian hierarchical cluster process model [Rbb0a73000b2f-1].
<code>HPGRF()</code>	Hierarchical Poisson/Gamma random field model [Rb6084ac49a63-1].
<code>SBLFR()</code>	Spatial Bayesian latent factor regression model [R270830ed9ee2-1].
<code>SBR()</code>	Spatial binary regression model [Rbf3e8ffed16f-1].

### `nimare.meta.cbma.model.BHICP`

**class** `BHICP`  
Bayesian hierarchical cluster process model [Rbb0a73000b2f-1].

**Warning:** This method is not yet implemented.

### References



## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

`nimare.meta.cbma.model.HPGRF`**class HPGRF**

Hierarchical Poisson/Gamma random field model [Rb6084ac49a63-1].

**Warning:** This method is not yet implemented.**References****Methods**

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self, dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.**Returns** Results of Estimator fitting.**Return type** `nimare.base.base.MetaResult`**get\_params** (*self, deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.**Returns** **params** – Parameter names mapped to their values.**Return type** mapping of string to any**classmethod load** (*filename, compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool, optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.**Return type** class object**save** (*self, filename, compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.

- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

`nimare.meta.cbma.model.SBLFR`

**class SBLFR**

Spatial Bayesian latent factor regression model [R270830ed9ee2-1].

**Warning:** This method is not yet implemented.

## References

## Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>\**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** *dataset* (`nimare.dataset.Dataset`) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.

- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

## `nimare.meta.cbma.model.SBR`

**class SBR**

Spatial binary regression model [[Rbf3e8ffed16f-1](#)].

**Warning:** This method is not yet implemented.

## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, <i>**params</i>)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (`nimare.dataset.Dataset`) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** `nimare.base.base.MetaResult`

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** `deep` (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** `params` – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load` (*filename, compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool, optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self, filename, compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool, optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self, \*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**class** `BHICP`

Bayesian hierarchical cluster process model [Rbb0a73000b2f-1].

**Warning:** This method is not yet implemented.

## References

## Methods

<code>fit</code> ( <i>self, dataset</i> )	Fit Estimator to Dataset.
<code>get_params</code> ( <i>self[, deep]</i> )	Get parameters for this estimator.
<code>load</code> ( <i>filename[, compressed]</i> )	Load a pickled class instance from file.
<code>save</code> ( <i>self, filename[, compress]</i> )	Pickle the class instance to the provided file.
<code>set_params</code> ( <i>self, **params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**class HPGRF**

Hierarchical Poisson/Gamma random field model [[Rb6084ac49a63-1](#)].

**Warning:** This method is not yet implemented.

## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** self

**class** SBLFR

Spatial Bayesian latent factor regression model [R270830ed9ee2-1].

**Warning:** This method is not yet implemented.

## References

## Methods

<code>fit(self, dataset)</code>	Fit Estimator to Dataset.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool, optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** **obj** – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.



- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**class SBR**

Spatial binary regression model [Rbf3e8ffed16f-1].

**Warning:** This method is not yet implemented.

## References

## Methods

<i>fit</i> ( <i>self</i> , <i>dataset</i> )	Fit Estimator to Dataset.
<i>get_params</i> ( <i>self</i> [, <i>deep</i> ])	Get parameters for this estimator.
<i>load</i> ( <i>filename</i> [, <i>compressed</i> ])	Load a pickled class instance from file.
<i>save</i> ( <i>self</i> , <i>filename</i> [, <i>compress</i> ])	Pickle the class instance to the provided file.
<i>set_params</i> ( <i>self</i> , <i>**params</i> )	Set the parameters of this estimator.

**fit** (*self*, *dataset*)

Fit Estimator to Dataset.

**Parameters** **dataset** (*nimare.dataset.Dataset*) – Dataset object to analyze.

**Returns** Results of Estimator fitting.

**Return type** *nimare.base.base.MetaResult*

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

## 6.3 `nimare.correct`: Multiple comparisons correction

Multiple comparisons correction

<code>nimare.correct</code>	Multiple comparisons correction
<code>nimare.correct.FWECorrector</code> ([ <i>method</i> ])	Perform family-wise error rate correction on a meta-analysis.
<code>nimare.correct.FDRCorrector</code> ( <i>alpha</i> , [ <i>method</i> ])	Perform false discovery rate correction on a meta-analysis.

### 6.3.1 `nimare.correct.FWECorrector`

**class FWECorrector** (*method='bonferroni'*, *\*\*kwargs*)

Perform family-wise error rate correction on a meta-analysis.

**Parameters**

- **method** (*obj*:`str`) – The FWE correction to use. Either 'bonferroni' or 'permutation'.
- **\*\*kwargs** – Keyword arguments to be used by the FWE correction implementation.

**Methods**

transform	
-----------	--

### 6.3.2 `nimare.correct.FDRCorrector`

**class FDRCorrector** (*alpha=0.05*, *method='indep'*, *\*\*kwargs*)

Perform false discovery rate correction on a meta-analysis.

**Parameters**

- **alpha** (*obj*:float) – The FDR correction rate to use.
- **method** (*obj*:str) – The FDR correction to use. Either ‘indep’ (for independent or positively correlated values) or ‘negcorr’ (for general or negatively correlated tests).

**Methods**

transform	
-----------	--

## 6.4 nimare.annotate: Automated annotation

Automated annotation tools

<code>nimare.annotate</code>	Automated annotation tools
<code>nimare.annotate.ontology</code>	Automated annotation tools for existing ontologies.
<code>nimare.annotate.topic</code>	Automated annotation with text-derived topic models.
<code>nimare.annotate.vector</code>	Automated annotation with text-derived vector models.
<code>nimare.annotate.text</code>	Text extraction tools.

### 6.4.1 nimare.annotate.ontology

Automated annotation tools for existing ontologies.

**extract\_cogpo** ()

Predict Cognitive Paradigm Ontology [1] labels with ATHENA classifiers [2].

<b>Warning:</b> This function is not yet implemented.
---

**References**

**extract\_cogat** (*text\_df*, *id\_df*)

Extract Cognitive Atlas [1] terms and perform hierarchical expansion.

**Parameters**

- **text\_df** ((D x 2) `pandas.DataFrame`) – Pandas dataframe with two columns: ‘id’ and ‘text’. D = document.
- **id\_df** ((T x 3) `pandas.DataFrame`) – Cognitive Atlas ontology dataframe with three columns: ‘id’ (unique identifier for term), ‘alias’ (natural language expression of term), and ‘name’ (preferred name of term; currently unused). T = term.

**Returns**

- **counts\_df** ((D x T) `pandas.DataFrame`) – Term counts for documents in the corpus.
- **rep\_text\_df** ((D x 2) `pandas.DataFrame`) – Text DataFrame with terms replaced with their CogAt IDs.

## References

**expand\_counts** (*counts\_df*, *rel\_df*, *weights=None*)

Perform hierarchical expansion of counts across labels.

### Parameters

- **counts\_df** ((D x T) `pandas.DataFrame`) – Term counts for a corpus. T = term, D = document.
- **rel\_df** (`pandas.DataFrame`) – Long-form DataFrame of term-term relationships with three columns: ‘input’, ‘output’, and ‘rel\_type’.
- **weights** (`dict`) – Dictionary of weights per relationship type. E.g., {‘isKind’: 1}. Unspecified relationship types default to 0.

**Returns** **weighted\_df** – Term counts for a corpus after hierarchical expansion.

**Return type** (D x T) `pandas.DataFrame`

**class CogAtLemmatizer** (*ontology\_df=None*)

Replace synonyms and abbreviations with Cognitive Atlas [R51a14fd76a1a-1] identifiers in text.

**Parameters** **ontology\_df** (`pandas.DataFrame`, optional) – DataFrame with three columns (id, name, alias) and one row for each alias (e.g., synonym or abbreviation) for each term in the Cognitive Atlas. If None, loads ontology file from resources folder.

### ontology

Ontology in DataFrame form.

**Type** `pandas.DataFrame`

### regex

Dictionary linking aliases in ontology to regular expressions for lemmatization.

**Type** `dict`

## References

## Methods

---

<code>lemmatize</code> (self, text[, convert_uk])	Replace terms in text with unique Cognitive Atlas identifiers.
---	--

---

**lemmatize** (*self*, *text*, *convert\_uk=True*)

Replace terms in text with unique Cognitive Atlas identifiers.

### Parameters

- **text** (`str`) – Text to convert.
- **convert\_uk** (`bool`, optional) – Convert British English words in text to American English versions. Default is True.

**Returns** **text** – Text with Cognitive Atlas terms replaced with unique Cognitive Atlas identifiers.

**Return type** `str`

## 6.4.2 nimare.annotate.topic

Automated annotation with text-derived topic models.

**class BoltzmannModel** (*text\_df*, *coordinates\_df*)

Generate a deep Boltzmann machine topic model [R3153eaf72258-1].

**Warning:** This method is not yet implemented.

### References

### Methods

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

fit

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

### Returns

Return type `self`

```
class GCLDAModel(count_df, coordinates_df, mask='mni152_2mm', n_topics=100, n_regions=2,
                  symmetric=True, alpha=0.1, beta=0.01, gamma=0.01, delta=1.0, dobs=25,
                  roi_size=50.0, seed_init=1)
```

Generate a generalized correspondence latent Dirichlet allocation (GCLDA) [R1dd767d5a89c-1] topic model.

### Parameters

- **count\_df** (`pandas.DataFrame`) – A DataFrame with feature counts for the model. The index is 'id', used for identifying studies. Other columns are features (e.g., unigrams and bigrams from Neurosynth), where each value is the number of times the feature is found in a given article.
- **coordinates\_df** (`pandas.DataFrame`, optional) – A DataFrame with a list of foci in the dataset. The index is 'id', used for identifying studies. Additional columns include 'x', 'y' and 'z' (foci in standard space).
- **n\_topics** (`int`, optional) – Number of topics to generate in model. The default is 100.
- **n\_regions** (`int`, optional) – Number of subregions per topic ( $\geq 1$ ). The default is 2.
- **alpha** (`float`, optional) – Prior count on topics for each document. The default is 0.1.
- **beta** (`float`, optional) – Prior count on word-types for each topic. The default is 0.01.
- **gamma** (`float`, optional) – Prior count added to y-counts when sampling z assignments. The default is 0.01.
- **delta** (`float`, optional) – Prior count on subregions for each topic. The default is 1.0.
- **dobs** (`int`, optional) – Spatial region 'default observations' (# observations weighting Sigma estimates in direction of default 'roi\_size' value). The default is 25.
- **roi\_size** (`float`, optional) – Default spatial 'region of interest' size (default value of diagonals in covariance matrix for spatial distribution, which the distributions are biased towards). The default is 50.0.
- **symmetric** (`bool`, optional) – Whether or not to use symmetry constraint on subregions. Symmetry requires `n_regions = 2`. The default is False.
- **seed\_init** (`int`, optional) – Initial value of random seed. The default is 1.

### References

See also:

`nimare.decode.continuous.gclda_decode_map` GCLDA map decoding

`nimare.decode.discrete.gclda_decode_roi` GCLDA ROI decoding

`nimare.decode.encode.encode_gclda` GCLDA text-to-map encoding

### Methods

<code>compute_log_likelihood(self[, model, ...])</code>	Compute log-likelihood [1] of a model object given current model.
<code>fit(self[, n_iters, loglikely_freq, verbose])</code>	Run multiple iterations.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>get_probs(self)</code>	Get conditional probability of selecting each voxel in the brain mask given each topic.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**compute\_log\_likelihood** (*self*, *model=None*, *update\_vectors=True*)

Compute log-likelihood [1] of a model object given current model. Computes the log-likelihood of data in any model object (either train or test) given the posterior predictive distributions over peaks and word-types for the model. Note that this is not computing the joint log-likelihood of model parameters and data.

#### Parameters

- **model** (`gclda.Model`, optional) – The model for which log-likelihoods will be calculated. If not provided, log-likelihood will be calculated for the current model (*self*).
- **update\_vectors** (`bool`, optional) – Whether to update model’s log-likelihood vectors or not.

#### Returns

- **x\_loglikely** (`float`) – Total log-likelihood of all peak tokens.
- **w\_loglikely** (`float`) – Total log-likelihood of all word tokens.
- **tot\_loglikely** (`float`) – Total log-likelihood of peak + word tokens.

## References

**fit** (*self*, *n\_iters=10000*, *loglikely\_freq=10*, *verbose=1*)

Run multiple iterations.

#### Parameters

- **n\_iters** (`int`, optional) – Number of iterations to run. Default is 10000.
- **loglikely\_freq** (`int`, optional) – The frequency with which log-likelihood is updated. Default value is 1 (log-likelihood is updated every iteration).
- **verbose** (`{0, 1, 2}`, optional) – Determines how much info is printed to console. 0 = none, 1 = a little, 2 = a lot. Default value is 2.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** **deep** (`boolean`, optional) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** **params** – Parameter names mapped to their values.

**Return type** mapping of string to any

**get\_probs** (*self*)

Get conditional probability of selecting each voxel in the brain mask given each topic.

#### Returns

- **p\_topic\_g\_voxel** (`numpy.ndarray` of `numpy.float64`) – A voxel-by-topic array of conditional probabilities:  $p(\text{topic}|\text{voxel})$ . For cell  $ij$ , the value is the probability of topic  $j$  being selected given voxel  $i$  is active.
- **p\_voxel\_g\_topic** (`numpy.ndarray` of `numpy.float64`) – A voxel-by-topic array of conditional probabilities:  $p(\text{voxel}|\text{topic})$ . For cell  $ij$ , the value is the probability of voxel  $i$  being selected given topic  $j$  has already been selected.
- **p\_topic\_g\_word** (`numpy.ndarray` of `numpy.float64`) – A word-by-topic array of conditional probabilities:  $p(\text{topic}|\text{word})$ . For cell  $ij$ , the value is the probability of topic  $i$  being selected given word  $j$  is present.
- **p\_word\_g\_topic** (`numpy.ndarray` of `numpy.float64`) – A word-by-topic array of conditional probabilities:  $p(\text{word}|\text{topic})$ . For cell  $ij$ , the value is the probability of word  $j$  being selected given topic  $i$  has already been selected.

**classmethod** `load` (`filename`, `compressed=True`)

Load a pickled class instance from file.

#### Parameters

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** `class object`

**save** (`self`, `filename`, `compress=True`)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (`self`, `**params`)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

#### Returns

**Return type** `self`

**class** `LDAModel` (`text_df`, `n_topics=50`, `n_iters=1000`, `alpha='auto'`, `beta=0.001`)

Perform topic modeling using Latent Dirichlet Allocation [R6c0c7b2ff831-1] with the Java toolbox MALLET [R6c0c7b2ff831-2], as performed in [R6c0c7b2ff831-3].

#### Parameters

- **text\_df** (`pandas.DataFrame`) – A pandas DataFrame with two columns ('id' and 'text') containing article text\_df.
- **n\_topics** (`int`, optional) – Number of topics to generate. Default=50.
- **n\_words** (`int`, optional) – Number of top words to return for each topic. Default=31, based on Poldrack et al. (2012). Not used.



- **n\_iters** (`int`, optional) – Number of iterations to run in training topic model. Default=1000.
- **alpha** (`float`, optional) – The Dirichlet prior on the per-document topic distributions. Default: 50 / n\_topics, based on Poldrack et al. (2012).
- **beta** (`float`, optional) – The Dirichlet prior on the per-topic word distribution. Default: 0.001, based on Poldrack et al. (2012).

## References

## Methods

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (`str`) – Name of file containing object.
- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

### 6.4.3 `nimare.annotate.vector`

Automated annotation with text-derived vector models.

**class** `Word2BrainModel` (*text\_df*, *coordinates\_df*)  
Generate a Word2Brain vector model [R3b75f33f3695-1].

**Warning:** This method is not yet implemented.

#### References

#### Methods

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** `load` (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** self

**class Text2BrainModel** (*text\_df, coordinates\_df*)  
Generate a Text2Brain vector model [[Rce8bb220c1e2-1](#)].

**Warning:** This method is not yet implemented.

## References

## Methods

<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**get\_params** (*self, deep=True*)  
Get parameters for this estimator.

**Parameters** *deep* (*boolean, optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename, compressed=True*)  
Load a pickled class instance from file.

#### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool, optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self, filename, compress=True*)  
Pickle the class instance to the provided file.

#### Parameters

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool, optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self, \*\*params*)  
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

## 6.4.4 `nimare.annotate.text`

Text extraction tools.

### Functions

<code>download_abstracts(dataset, email)</code>	Download the abstracts for a list of PubMed IDs.
<code>generate_cooccurrence(text_df, ...)</code>	Build co-occurrence matrix from documents.
<code>generate_counts(text_df, text_column, tfidf)</code>	Generate tf-idf weights for unigrams/bigrams derived from textual data.
<code>uk_to_us(text)</code>	Convert UK spellings to US based on a converter.

#### `nimare.annotate.text.download_abstracts`

**download\_abstracts** (*dataset, email*)

Download the abstracts for a list of PubMed IDs. Uses the BioPython package.

**Parameters**

- **dataset** (*nimare.dataset.Dataset* or *list of str*) – A Dataset object where IDs are in the form PMID-EXPID or a list of PubMed IDs
- **email** (*str*) – Email address to use to call the PubMed API

**Returns** `dataset` – Dataset with abstracts added.

**Return type** *nimare.dataset.Dataset* or *list of str*

#### `nimare.annotate.text.generate_cooccurrence`

**generate\_cooccurrence** (*text\_df, text\_column='abstract', vocabulary=None, window=5*)

Build co-occurrence matrix from documents. Not the same approach as used by the GloVe model.

**Parameters**

- **text\_df** ((*D x 2*) *pandas.DataFrame*) – A DataFrame with two columns ('id' and 'text'). *D* = document.
- **vocabulary** (*list*, optional) – List of words in vocabulary to extract from text.
- **window** (*int*, optional) – Window size for cooccurrence. Words which appear within window words of one another co-occur.

**Returns** `df` – One cooccurrence matrix per document in `text_df`.

**Return type** (*V, V, D*) *pandas.Panel*

**nimare.annotate.text.generate\_counts****generate\_counts** (*text\_df*, *text\_column*='abstract', *tfidf*=True)

Generate tf-idf weights for unigrams/bigrams derived from textual data.

**Parameters** **text\_df** ((D x 2) `pandas.DataFrame`) – A DataFrame with two columns ('id' and 'text'). D = document.**Returns** **weights\_df** – A DataFrame where the index is 'id' and the columns are the unigrams/bigrams derived from the data. D = document. T = term.**Return type** (D x T) `pandas.DataFrame`**nimare.annotate.text.uk\_to\_us****uk\_to\_us** (*text*)

Convert UK spellings to US based on a converter.

english\_spellings.csv: From <http://www.tysto.com/uk-us-spelling-list.html>**Parameters** **text** (`str`) –**Returns** **text****Return type** `str`**download\_abstracts** (*dataset*, *email*)

Download the abstracts for a list of PubMed IDs. Uses the BioPython package.

**Parameters**

- **dataset** (`nimare.dataset.Dataset` or `list` of `str`) – A Dataset object where IDs are in the form PMID-EXPID or a list of PubMed IDs
- **email** (`str`) – Email address to use to call the PubMed API

**Returns** **dataset** – Dataset with abstracts added.**Return type** `nimare.dataset.Dataset` or `list` of `str`**generate\_cooccurrence** (*text\_df*, *text\_column*='abstract', *vocabulary*=None, *window*=5)

Build co-occurrence matrix from documents. Not the same approach as used by the GloVe model.

**Parameters**

- **text\_df** ((D x 2) `pandas.DataFrame`) – A DataFrame with two columns ('id' and 'text'). D = document.
- **vocabulary** (`list`, optional) – List of words in vocabulary to extract from text.
- **window** (`int`, optional) – Window size for cooccurrence. Words which appear within window words of one another co-occur.

**Returns** **df** – One cooccurrence matrix per document in `text_df`.**Return type** (V, V, D) `pandas.Panel`**generate\_counts** (*text\_df*, *text\_column*='abstract', *tfidf*=True)

Generate tf-idf weights for unigrams/bigrams derived from textual data.

**Parameters** **text\_df** ((D x 2) `pandas.DataFrame`) – A DataFrame with two columns ('id' and 'text'). D = document.**Returns** **weights\_df** – A DataFrame where the index is 'id' and the columns are the unigrams/bigrams derived from the data. D = document. T = term.

**Return type** (D x T) `pandas.DataFrame`

**uk\_to\_us** (*text*)

Convert UK spellings to US based on a converter.

english\_spellings.csv: From <http://www.tysto.com/uk-us-spelling-list.html>

**Parameters** **text** (*str*) –

**Returns** **text**

**Return type** *str*

## 6.5 nimare.decode: Functional characterization analysis

Functional decoding tools

<code>nimare.decode</code>	Functional decoding tools
<code>nimare.decode.discrete</code>	Methods for decoding subsets of voxels (e.g., ROIs) or experiments (e.g., from meta-analytic clustering on a database) into text.
<code>nimare.decode.continuous</code>	Methods for decoding unthresholded brain maps into text.
<code>nimare.decode.encode</code>	Methods for encoding text into brain maps.

### 6.5.1 nimare.decode.discrete

Methods for decoding subsets of voxels (e.g., ROIs) or experiments (e.g., from meta-analytic clustering on a database) into text.

#### Functions

<code>brainmap_decode(coordinates, annotations, ids)</code>	Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the BrainMap method [1].
<code>gclda_decode_roi(model, roi[, topic_priors, ...])</code>	Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the method described in [1].
<code>neurosynth_decode(coordinates, annotations, ids)</code>	Perform discrete functional decoding according to Neurosynth's meta-analytic method [1].

#### `nimare.decode.discrete.brainmap_decode`

**brainmap\_decode** (*coordinates, annotations, ids, ids2=None, features=None, frequency\_threshold=0.001, u=0.05, correction='fdr\_bh'*)

Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the BrainMap method [1].

#### References

**nimare.decode.discrete.gclda\_decode\_roi****gclda\_decode\_roi** (*model*, *roi*, *topic\_priors=None*, *prior\_weight=1.0*)

Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the method described in [1].

**Parameters**

- **model** (*nimare.annotate.topic.GCLDAModel*) – Model object needed for decoding.
- **roi** (*nibabel.nifti1.Nifti1Image* or *str*) – Binary image to decode into text. If string, path to a file with the binary image.
- **topic\_priors** (*numpy.ndarray* of *float*, optional) – A 1d array of size (*n\_topics*) with values for topic weighting. If *None*, no weighting is done. Default is *None*.
- **prior\_weight** (*float*, optional) – The weight by which the prior will affect the decoding. Default is 1.

**Returns**

- **decoded\_df** (*pandas.DataFrame*) – A *DataFrame* with the word-tokens and their associated weights.
- **topic\_weights** (*numpy.ndarray* of *float*) – The weights of the topics used in decoding.

**Notes**

Notation	Meaning
$v$	Voxel
$t$	Topic
$w$	Word type
$r$	Region of interest (ROI)
$p(v t)$	Probability of topic given voxel ( <i>p_topic_g_voxel</i> )
$\tau_t$	Topic weight vector ( <i>topic_weights</i> )
$p(w t)$	Probability of word type given topic ( <i>p_word_g_topic</i> )

1. Compute  $p(v|t)$ .
  - From `gclda.model.Model.get_spatial_probs()`
2. Compute topic weight vector ( $\tau_t$ ) by adding across voxels within ROI.
  - $$\tau_t = \sum_i p(t|v_i)$$
3. Multiply  $\tau_t$  by  $p(w|t)$ .
  - $$p(w|r) \propto \tau_t \cdot p(w|t)$$
4. The resulting vector (*word\_weights*) reflects arbitrarily scaled term weights for the ROI.

## References

### `nimare.decode.discrete.neurosynth_decode`

**neurosynth\_decode** (*coordinates, annotations, ids, ids2=None, features=None, frequency\_threshold=0.001, prior=0.5, u=0.05, correction='fdr\_bh'*)

Perform discrete functional decoding according to Neurosynth's meta-analytic method [1]. This does not employ correlations between unthresholded maps, which are the method of choice for decoding within Neurosynth and Neurovault. Metadata (i.e., feature labels) for studies within the selected sample (*ids*) are compared to the unselected studies remaining in the database (*dataset*).

## References

**brainmap\_decode** (*coordinates, annotations, ids, ids2=None, features=None, frequency\_threshold=0.001, u=0.05, correction='fdr\_bh'*)

Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the BrainMap method [1].

## References

**gclda\_decode\_roi** (*model, roi, topic\_priors=None, prior\_weight=1.0*)

Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the method described in [1].

### Parameters

- **model** (*nimare.annotate.topic.GCLDAModel*) – Model object needed for decoding.
- **roi** (*nibabel.nifti1.Nifti1Image* or *str*) – Binary image to decode into text. If string, path to a file with the binary image.
- **topic\_priors** (*numpy.ndarray* of *float*, optional) – A 1d array of size (*n\_topics*) with values for topic weighting. If *None*, no weighting is done. Default is *None*.
- **prior\_weight** (*float*, optional) – The weight by which the prior will affect the decoding. Default is 1.

### Returns

- **decoded\_df** (*pandas.DataFrame*) – A *DataFrame* with the word-tokens and their associated weights.
- **topic\_weights** (*numpy.ndarray* of *float*) – The weights of the topics used in decoding.



## Notes

Notation	Meaning
$v$	Voxel
$t$	Topic
$w$	Word type
$r$	Region of interest (ROI)
$p(v t)$	Probability of topic given voxel ( <code>p_topic_g_voxel</code> )
$\tau_t$	Topic weight vector ( <code>topic_weights</code> )
$p(w t)$	Probability of word type given topic ( <code>p_word_g_topic</code> )

1. Compute  $p(v|t)$ .
  - From `gclda.model.Model.get_spatial_probs()`
2. Compute topic weight vector ( $\tau_t$ ) by adding across voxels within ROI.
  - $$\tau_t = \sum_i p(t|v_i)$$
3. Multiply  $\tau_t$  by  $p(w|t)$ .
  - $$p(w|r) \propto \tau_t \cdot p(w|t)$$
4. The resulting vector (`word_weights`) reflects arbitrarily scaled term weights for the ROI.

## References

**neurosynth\_decode** (*coordinates*, *annotations*, *ids*, *ids2=None*, *features=None*, *frequency\_threshold=0.001*, *prior=0.5*, *u=0.05*, *correction='fdr\_bh'*)

Perform discrete functional decoding according to Neurosynth's meta-analytic method [1]. This does not employ correlations between unthresholded maps, which are the method of choice for decoding within Neurosynth and Neurovault. Metadata (i.e., feature labels) for studies within the selected sample (*ids*) are compared to the unselected studies remaining in the database (*dataset*).

## References

### 6.5.2 nimare.decode.continuous

Methods for decoding unthresholded brain maps into text.

## Functions

---

<code>corr_decode(img, dataset[, features, ...])</code>	<b>param img</b> Input image to decode. Must have same affine/dimensions as dataset
<code>corr_dist_decode(img, dataset[, features, ...])</code>	Builds feature-specific distributions of correlations with input image for image-based meta-analytic functional decoding.
<code>gclda_decode_map(model, image[, ...])</code>	Perform image-to-text decoding for continuous inputs (e.g., unthresholded statistical maps), according to the method described in [1].

---

### `nimare.decode.continuous.corr_decode`

**corr\_decode** (*img*, *dataset*, *features=None*, *frequency\_threshold=0.001*, *meta\_estimator=None*, *target\_image='specificity\_z'*)

#### Parameters

- **img** (`nibabel.Nifti1.Nifti1Image`) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **dataset** – A dataset with coordinates.
- **features** (`list`, optional) – List of features in dataset annotations to use for decoding. Default is `None`, which uses all features available.
- **frequency\_threshold** (`float`, optional) – Threshold to apply to dataset annotations. Values greater than or equal to the threshold as assigned as label+, while values below the threshold are considered label-. Default is 0.001.
- **meta\_estimator** (initialized `nimare.meta.cbma.base.CBMAEstimator`, optional) – Defaults to `MKDACHi2`.
- **target\_image** (`str`, optional) – Image from `meta_estimator`'s results to use for decoding. Dependent on estimator.

**Returns** **out\_df** – A `DataFrame` with two columns: 'feature' (label) and 'r' (correlation coefficient). There will be one row for each feature.

**Return type** `pandas.DataFrame`

### `nimare.decode.continuous.corr_dist_decode`

**corr\_dist\_decode** (*img*, *dataset*, *features=None*, *frequency\_threshold=0.001*, *target\_image='z'*)

Builds feature-specific distributions of correlations with input image for image-based meta-analytic functional decoding.

#### Parameters

- **img** (`nibabel.Nifti1.Nifti1Image`) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **dataset** – A dataset with images.
- **features** (`list`, optional) – List of features in dataset annotations to use for decoding. Default is `None`, which uses all features available.

- **frequency\_threshold** (*float*, optional) – Threshold to apply to dataset annotations. Values greater than or equal to the threshold as assigned as label+, while values below the threshold are considered label-. Default is 0.001.
- **target\_image** ({'z', 'con'}, *optional*) – Image type from database to use for decoding.

**Returns** **out\_df** – DataFrame with a row for each feature used for decoding and two columns: mean and std. Values describe the distributions of correlation coefficients (in terms of Fisher-transformed z-values).

**Return type** `pandas.DataFrame`

### `nimare.decode.continuous.gclda_decode_map`

**gclda\_decode\_map** (*model*, *image*, *topic\_priors=None*, *prior\_weight=1*)

Perform image-to-text decoding for continuous inputs (e.g., unthresholded statistical maps), according to the method described in [1].

#### Parameters

- **model** (*nimare.annotate.topic.GCLDAModel*) – Model object needed for decoding.
- **image** (*nibabel.nifti1.Nifti1Image* or *str*) – Whole-brain image to decode into text. Must be in same space as model and dataset. Model's template available in *model.dataset.mask\_img*.
- **topic\_priors** (*numpy.ndarray* of *float*, optional) – A 1d array of size (*n\_topics*) with values for topic weighting. If *None*, no weighting is done. Default is *None*.
- **prior\_weight** (*float*, optional) – The weight by which the prior will affect the decoding. Default is 1.

#### Returns

- **decoded\_df** (*pandas.DataFrame*) – A DataFrame with the word-tokens and their associated weights.
- **topic\_weights** (*numpy.ndarray* of *float*) – The weights of the topics used in decoding.

### Notes

Notation	Meaning
$v$	Voxel
$t$	Topic
$w$	Word type
$i$	Input image
$p(v t)$	Probability of topic given voxel ( <i>p_topic_g_voxel</i> )
$\tau_t$	Topic weight vector ( <i>topic_weights</i> )
$p(w t)$	Probability of word type given topic ( <i>p_word_g_topic</i> )
$\omega$	1d array from input image ( <i>input_values</i> )

1. Compute  $p(t|v)$  (*p\_topic\_g\_voxel*).

- `gclda.model.Model.get_spatial_probs()`
- 2. Squeeze input image to 1d array  $\omega$  (`input_values`).
- 3. Compute topic weight vector ( $\tau_t$ ) by multiplying  $p(t|v)$  by input image.
  - $\tau_t = p(t|v) \cdot \omega$
- 4. Multiply  $\tau_t$  by  $p(w|t)$ .
  - $p(w|i) \propto \tau_t \cdot p(w|t)$
- 5. The resulting vector (`word_weights`) reflects arbitrarily scaled term weights for the input image.

## References

`corr_decode`(*img*, *dataset*, *features=None*, *frequency\_threshold=0.001*, *meta\_estimator=None*, *target\_image='specificity\_z'*)

### Parameters

- **img** (`nibabel.Nifti1.Nifti1Image`) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **dataset** – A dataset with coordinates.
- **features** (`list`, optional) – List of features in dataset annotations to use for decoding. Default is `None`, which uses all features available.
- **frequency\_threshold** (`float`, optional) – Threshold to apply to dataset annotations. Values greater than or equal to the threshold as assigned as label+, while values below the threshold are considered label-. Default is 0.001.
- **meta\_estimator** (initialized `nimare.meta.cbma.base.CBMAEstimator`, optional) – Defaults to `MKDACHi2`.
- **target\_image** (`str`, optional) – Image from `meta_estimator`'s results to use for decoding. Dependent on estimator.

**Returns** `out_df` – A `DataFrame` with two columns: 'feature' (label) and 'r' (correlation coefficient). There will be one row for each feature.

**Return type** `pandas.DataFrame`

`corr_dist_decode`(*img*, *dataset*, *features=None*, *frequency\_threshold=0.001*, *target\_image='z'*)

Builds feature-specific distributions of correlations with input image for image-based meta-analytic functional decoding.

### Parameters

- **img** (`nibabel.Nifti1.Nifti1Image`) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **dataset** – A dataset with images.
- **features** (`list`, optional) – List of features in dataset annotations to use for decoding. Default is `None`, which uses all features available.

- **frequency\_threshold** (`float`, optional) – Threshold to apply to dataset annotations. Values greater than or equal to the threshold as assigned as label+, while values below the threshold are considered label-. Default is 0.001.
- **target\_image** (`{'z', 'con'}`, optional) – Image type from database to use for decoding.

**Returns** `out_df` – DataFrame with a row for each feature used for decoding and two columns: mean and std. Values describe the distributions of correlation coefficients (in terms of Fisher-transformed z-values).

**Return type** `pandas.DataFrame`

**gclda\_decode\_map** (`model`, `image`, `topic_priors=None`, `prior_weight=1`)

Perform image-to-text decoding for continuous inputs (e.g., unthresholded statistical maps), according to the method described in [1].

#### Parameters

- **model** (`nimare.annotate.topic.GCLDAModel`) – Model object needed for decoding.
- **image** (`nibabel.nifti1.Nifti1Image` or `str`) – Whole-brain image to decode into text. Must be in same space as model and dataset. Model's template available in `model.dataset.mask_img`.
- **topic\_priors** (`numpy.ndarray` of `float`, optional) – A 1d array of size (`n_topics`) with values for topic weighting. If `None`, no weighting is done. Default is `None`.
- **prior\_weight** (`float`, optional) – The weight by which the prior will affect the decoding. Default is 1.

#### Returns

- **decoded\_df** (`pandas.DataFrame`) – A DataFrame with the word-tokens and their associated weights.
- **topic\_weights** (`numpy.ndarray` of `float`) – The weights of the topics used in decoding.

#### Notes

Notation	Meaning
$v$	Voxel
$t$	Topic
$w$	Word type
$i$	Input image
$p(v t)$	Probability of topic given voxel ( <code>p_topic_g_voxel</code> )
$\tau_t$	Topic weight vector ( <code>topic_weights</code> )
$p(w t)$	Probability of word type given topic ( <code>p_word_g_topic</code> )
$\omega$	1d array from input image ( <code>input_values</code> )

1. Compute  $p(t|v)$  (`p_topic_g_voxel`).
  - From `gclda.model.Model.get_spatial_probs()`
2. Squeeze input image to 1d array  $\omega$  (`input_values`).

3. Compute topic weight vector ( $\tau_t$ ) by multiplying  $p(t|v)$  by input image.
  - $\tau_t = p(t|v) \cdot \omega$
4. Multiply  $\tau_t$  by  $p(w|t)$ .
  - $p(w|i) \propto \tau_t \cdot p(w|t)$
5. The resulting vector (`word_weights`) reflects arbitrarily scaled term weights for the input image.

## References

### 6.5.3 nimare.decode.encode

Methods for encoding text into brain maps.

#### Functions

<code>encode_glda(model, text[, out_file, ...])</code>	Perform text-to-image encoding according to the method described in [1].
<code>text2brain()</code>	Perform text-to-image encoding with the text2brain model [1].

#### `nimare.decode.encode.encode_glda`

**encode\_glda** (*model*, *text*, *out\_file=None*, *topic\_priors=None*, *prior\_weight=1.0*)

Perform text-to-image encoding according to the method described in [1].

##### Parameters

- **model** (`nimare.annotate.topic.GCLDAModel`) – Model object needed for decoding.
- **text** (`str` or `list`) – Text to encode into an image.
- **out\_file** (`str`, optional) – If not None, writes the encoded image to a file.
- **topic\_priors** (`numpy.ndarray` of `float`, optional) – A 1d array of size (`n_topics`) with values for topic weighting. If None, no weighting is done. Default is None.
- **prior\_weight** (`float`, optional) – The weight by which the prior will affect the encoding. Default is 1.

##### Returns

- **img** (`nibabel.nifti1.Nifti1Image`) – The encoded image.
- **topic\_weights** (`numpy.ndarray` of `float`) – The weights of the topics used in encoding.

## Notes

Notation	Meaning
$v$	Voxel
$t$	Topic
$w$	Word type
$h$	Input text
$p(v t)$	Probability of topic given voxel (p_topic_g_voxel)
$\tau_t$	Topic weight vector (topic_weights)
$p(w t)$	Probability of word type given topic (p_word_g_topic)
$\omega$	1d array from input image (input_values)

1. Compute  $p(v|t)$  (p\_voxel\_g\_topic).
  - From `gclda.model.Model.get_spatial_probs()`
2. Compute  $p(t|w)$  (p\_topic\_g\_word).
3. Vectorize input text according to model vocabulary.
4. Reduce  $p(t|w)$  to only include word types in input text.
5. Compute  $p(t|h)$  (p\_topic\_g\_text) by multiplying  $p(t|w)$  by word counts for input text.
6. Sum topic weights ( $\tau_t$ ) across words.
  - $$\tau_t = \sum_i p(t|h_i)$$
7. Compute voxel weights.
  - $$p(v|h) \propto p(v|t) \cdot \tau_t$$
8. The resulting array (voxel\_weights) reflects arbitrarily scaled voxel weights for the input text.
9. Unmask and reshape voxel\_weights into brain image.

## References

`nimare.decode.encode.text2brain`

`text2brain()`

Perform text-to-image encoding with the text2brain model [1].

**Warning:** This method is not yet implemented.

## References

`encode_gclda(model, text, out_file=None, topic_priors=None, prior_weight=1.0)`

Perform text-to-image encoding according to the method described in [1].

### Parameters

- **model** (`nimare.annotate.topic.GCLDAModel`) – Model object needed for decoding.
- **text** (`str` or `list`) – Text to encode into an image.
- **out\_file** (`str`, optional) – If not None, writes the encoded image to a file.
- **topic\_priors** (`numpy.ndarray` of `float`, optional) – A 1d array of size (`n_topics`) with values for topic weighting. If None, no weighting is done. Default is None.
- **prior\_weight** (`float`, optional) – The weight by which the prior will affect the encoding. Default is 1.

### Returns

- **img** (`nibabel.nifti1.Nifti1Image`) – The encoded image.
- **topic\_weights** (`numpy.ndarray` of `float`) – The weights of the topics used in encoding.

### Notes

Notation	Meaning
$v$	Voxel
$t$	Topic
$w$	Word type
$h$	Input text
$p(v t)$	Probability of topic given voxel ( <code>p_topic_g_voxel</code> )
$\tau_t$	Topic weight vector ( <code>topic_weights</code> )
$p(w t)$	Probability of word type given topic ( <code>p_word_g_topic</code> )
$\omega$	1d array from input image ( <code>input_values</code> )

1. Compute  $p(v|t)$  (`p_voxel_g_topic`).
  - From `gclda.model.Model.get_spatial_probs()`
2. Compute  $p(t|w)$  (`p_topic_g_word`).
3. Vectorize input text according to model vocabulary.
4. Reduce  $p(t|w)$  to only include word types in input text.
5. Compute  $p(t|h)$  (`p_topic_g_text`) by multiplying  $p(t|w)$  by word counts for input text.
6. Sum topic weights ( $\tau_t$ ) across words.
  - $$\tau_t = \sum_i p(t|h_i)$$
7. Compute voxel weights.
  - $$\tau_t \propto p(v|h) \cdot p(v|t)$$
8. The resulting array (`voxel_weights`) reflects arbitrarily scaled voxel weights for the input text.
9. Unmask and reshape `voxel_weights` into brain image.



## References

**text2brain()**

Perform text-to-image encoding with the text2brain model [1].

**Warning:** This method is not yet implemented.

## References

## 6.6 nimare.parcellate: Meta-analytic parcellation

Meta-analytic parcellation tools

<code>nimare.parcellate</code>	Meta-analytic parcellation tools
<code>nimare.parcellate.cbp</code>	Coactivation-based parcellation
<code>nimare.parcellate.mamp</code>	Meta-analytic activation modeling-based parcellation (MAMP).
<code>nimare.parcellate.mapbot</code>	Meta-analytic parcellation based on text (MAPBOT).

### 6.6.1 nimare.parcellate.cbp

Coactivation-based parcellation

## Classes

<code>CoordCBP(dataset, ids)</code>	Coordinate-based coactivation-based parcellation [R759c29a4323f-1].
<code>ImCBP(dataset, ids)</code>	Image-based coactivation-based parcellation

### `nimare.parcellate.cbp.CoordCBP`

**class CoordCBP** (*dataset, ids*)

Coordinate-based coactivation-based parcellation [R759c29a4323f-1].

## Notes

Here are the steps:

1. For each voxel in the mask, identify studies in dataset corresponding to that voxel. Selection criteria can be either based on a distance threshold (e.g., all studies with foci within 5mm of voxel) or based on a minimum number of studies (e.g., the 50 studies reporting foci closest to the voxel).
2. For each voxel, perform MACM (meta-analysis) using the identified studies.
3. Correlate statistical maps between voxel MACMs to generate  $n\_voxels \times n\_voxels$  correlation matrix.
4. Convert correlation coefficients to correlation distance ( $1 - r$ ) values.
5. Perform clustering on correlation distance matrix.

**Warning:** This method is currently untested.

## References

## Methods

<code>fit(self, target_mask[, method, r, n_exps, ...])</code>	Run CBP parcellation.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *target\_mask*, *method*='min\_distance', *r*=5, *n\_exps*=50, *n\_parcel*s=2, *meta\_estimator*=<class 'nimare.meta.cbma.ale.SCALE'>, \*\*kwargs)  
Run CBP parcellation.

### Parameters

- **target\_mask** (*img\_like*) – Image with binary mask for region of interest to be parcellated.
- **n\_parcel**s (*int* or *array\_like* of *int*, optional) – Number of parcels to generate for ROI. If *array\_like*, each parcel number will be evaluated and results for all will be returned. Default is 2.
- **n\_iter**s (*int*, optional) – Number of iterations to run for each parcel number. Default is 10000.
- **n\_cores** (*int*, optional) – Number of cores to use for model fitting.

### Returns

**Return type** results

**get\_params** (*self*, *deep*=True)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed*=True)

Load a pickled class instance from file.

### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)  
Pickle the class instance to the provided file.

#### Parameters

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)  
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

#### Returns

**Return type** *self*

### `nimare.parcellate.cbp.ImCBP`

**class ImCBP** (*dataset*, *ids*)  
Image-based coactivation-based parcellation

**Warning:** This method is not yet implemented.

#### Methods

*fit*(*self*, *target\_mask*[, *n\_parcel*s])

**param target\_mask** Image with binary mask for region of interest to be parcellated.

*get\_params*(*self*[, *deep*])

Get parameters for this estimator.

*load*(*filename*[, *compressed*])

Load a pickled class instance from file.

*save*(*self*, *filename*[, *compress*])

Pickle the class instance to the provided file.

*set\_params*(*self*, *\*\*params*)

Set the parameters of this estimator.

**fit** (*self*, *target\_mask*, *n\_parcel*s=2)

#### Parameters

- **target\_mask** (*img\_like*) – Image with binary mask for region of interest to be parcellated.
- **n\_parcel**s (*int* or *array\_like* of *int*, optional) – Number of parcels to generate for ROI. If *array\_like*, each parcel number will be evaluated and results for all will be returned. Default is 2.
- **n\_iters** (*int*, optional) – Number of iterations to run for each parcel number. Default is 10000.
- **n\_cores** (*int*, optional) – Number of cores to use for model fitting.

**Returns****Return type** results**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.**Returns** *params* – Parameter names mapped to their values.**Return type** mapping of string to any**classmethod** **load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.**Return type** class object**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns****Return type** self**class** **CoordCBP** (*dataset*, *ids*)Coordinate-based coactivation-based parcellation [[R759c29a4323f-1](#)].**Notes****Here are the steps:**

1. For each voxel in the mask, identify studies in dataset corresponding to that voxel. Selection criteria can be either based on a distance threshold (e.g., all studies with foci within 5mm of voxel) or based on a minimum number of studies (e.g., the 50 studies reporting foci closest to the voxel).
2. For each voxel, perform MACM (meta-analysis) using the identified studies.
3. Correlate statistical maps between voxel MACMs to generate n\_voxels X n\_voxels correlation matrix.

4. Convert correlation coefficients to correlation distance ( $1 - r$ ) values.
5. Perform clustering on correlation distance matrix.

**Warning:** This method is currently untested.

## References

## Methods

<code>fit(self, target_mask[, method, r, n_exps, ...])</code>	Run CBP parcellation.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *target\_mask*, *method*='min\_distance', *r*=5, *n\_exps*=50, *n\_parcel*s=2, *meta\_estimator*=<class 'nimare.meta.cbma.ale.SCALE'>, *\*\*kwargs*)  
Run CBP parcellation.

### Parameters

- **target\_mask** (*img\_like*) – Image with binary mask for region of interest to be parcellated.
- **n\_parcel**s (*int* or *array\_like* of *int*, optional) – Number of parcels to generate for ROI. If *array\_like*, each parcel number will be evaluated and results for all will be returned. Default is 2.
- **n\_iter**s (*int*, optional) – Number of iterations to run for each parcel number. Default is 10000.
- **n\_cores** (*int*, optional) – Number of cores to use for model fitting.

### Returns

**Return type** results

**get\_params** (*self*, *deep*=True)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed*=True)

Load a pickled class instance from file.

### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

**class ImCBP** (*dataset*, *ids*)

Image-based coactivation-based parcellation

**Warning:** This method is not yet implemented.

## Methods

---

*fit*(*self*, *target\_mask*[, *n\_parcel*s])

**param target\_mask** Image with binary mask for region of interest to be parcellated.

---

*get\_params*(*self*[, *deep*])

Get parameters for this estimator.

---

*load*(*filename*[, *compressed*])

Load a pickled class instance from file.

---

*save*(*self*, *filename*[, *compress*])

Pickle the class instance to the provided file.

---

*set\_params*(*self*, *\*\*params*)

Set the parameters of this estimator.

---

**fit** (*self*, *target\_mask*, *n\_parcel*s=2)

**Parameters**

- **target\_mask** (*img\_like*) – Image with binary mask for region of interest to be parcellated.
- **n\_parcel**s (`int` or array\_like of `int`, optional) – Number of parcels to generate for ROI. If array\_like, each parcel number will be evaluated and results for all will be returned. Default is 2.
- **n\_iters** (`int`, optional) – Number of iterations to run for each parcel number. Default is 10000.
- **n\_cores** (`int`, optional) – Number of cores to use for model fitting.

**Returns**

**Return type** results

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** self

## 6.6.2 nimare.parcellate.mamp

Meta-analytic activation modeling-based parcellation (MAMP).

### Classes

---

*MAMP*(dataset, ids)

Meta-analytic activation modeling-based parcellation (MAMP) [Re636c01f812e-1].

---

**nimare.parcellate.mamp.MAMP****class MAMP** (*dataset, ids*)

Meta-analytic activation modeling-based parcellation (MAMP) [Re636c01f812e-1].

**Parameters**

- **text** (*list of str*) – List of texts to use for parcellation.
- **mask** (*str or nibabel.Nifti1.Nifti1Image*) – Mask file or image.

**Notes**

MAMP works similarly to CBP, but skips the step of performing a MACM for each voxel. Here are the steps:

1. Create an MA map for each study in the dataset.
2. Concatenate MA maps across studies to create a 4D dataset.
3. Extract values across studies for voxels in mask, resulting in *n\_voxels X n\_studies* array.
4. Correlate “study series” between voxels to generate *n\_voxels X n\_voxels* correlation matrix.
5. Convert correlation coefficients to correlation distance ( $1 - r$ ) values.
6. Perform clustering on correlation distance matrix.

**Warning:** This method is not yet implemented.

**References****Methods**

<code>fit(self, target_mask[, n_parcel, ...])</code>	Run MAMP parcellation.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self, target\_mask, n\_parcel=2, kernel\_estimator=<class 'nimare.meta.cbma.kernel.ALEKernel'>, \*\*kwargs*)  
Run MAMP parcellation.

**Parameters**

- **target\_mask** (*img\_like*) – Image with binary mask for region of interest to be parcellated.
- **n\_parcel** (*int or array\_like of int, optional*) – Number of parcels to generate for ROI. If *array\_like*, each parcel number will be evaluated and results for all will be returned. Default is 2.
- **n\_iters** (*int, optional*) – Number of iterations to run for each parcel number. Default is 10000.
- **n\_cores** (*int, optional*) – Number of cores to use for model fitting.

**Returns**



**Return type** results

**get\_params** (*self*, *deep=True*)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed=True*)

Load a pickled class instance from file.

**Parameters**

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, *optional*) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

**Parameters**

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, *optional*) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

**Returns**

**Return type** *self*

**class** **MAMP** (*dataset*, *ids*)

Meta-analytic activation modeling-based parcellation (MAMP) [Re636c01f812e-1].

**Parameters**

- **text** (*list of str*) – List of texts to use for parcellation.
- **mask** (*str* or *nibabel.Nifti1.Nifti1Image*) – Mask file or image.

## Notes

MAMP works similarly to CBP, but skips the step of performing a MACM for each voxel. Here are the steps:

1. Create an MA map for each study in the dataset.
2. Concatenate MA maps across studies to create a 4D dataset.
3. Extract values across studies for voxels in mask, resulting in *n\_voxels* X *n\_studies* array.

4. Correlate “study series” between voxels to generate  $n\_voxels \times n\_voxels$  correlation matrix.
5. Convert correlation coefficients to correlation distance ( $1 - r$ ) values.
6. Perform clustering on correlation distance matrix.

**Warning:** This method is not yet implemented.

## References

## Methods

<code>fit(self, target_mask[, n_parcel, ...])</code>	Run MAMP parcellation.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *target\_mask*, *n\_parcel*=2, *kernel\_estimator*=<class 'nimare.meta.cbma.kernel.ALEKernel'>, \*\**kwargs*)  
Run MAMP parcellation.

### Parameters

- **target\_mask** (*img\_like*) – Image with binary mask for region of interest to be parcellated.
- **n\_parcel** (*int* or *array\_like* of *int*, optional) – Number of parcels to generate for ROI. If *array\_like*, each parcel number will be evaluated and results for all will be returned. Default is 2.
- **n\_iters** (*int*, optional) – Number of iterations to run for each parcel number. Default is 10000.
- **n\_cores** (*int*, optional) – Number of cores to use for model fitting.

### Returns

**Return type** results

**get\_params** (*self*, *deep*=True)

Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod** **load** (*filename*, *compressed*=True)

Load a pickled class instance from file.

### Parameters

- **filename** (*str*) – Name of file containing object.

- **compressed** (`bool`, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)  
Pickle the class instance to the provided file.

**Parameters**

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)  
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

**Returns**

**Return type** `self`

### 6.6.3 nimare.parcellate.mapbot

Meta-analytic parcellation based on text (MAPBOT).

#### Classes

---

<code>MAPBOT</code> ( <i>tfidf_df</i> , <i>coordinates_df</i> , <i>mask</i> )	Meta-analytic parcellation based on text (MAPBOT) [R84461f98615b-1].
---	--

---

#### `nimare.parcellate.mapbot.MAPBOT`

**class** `MAPBOT` (*tfidf\_df*, *coordinates\_df*, *mask*)  
Meta-analytic parcellation based on text (MAPBOT) [R84461f98615b-1].

**Parameters**

- **tfidf\_df** (`pandas.DataFrame`) – A DataFrame with feature counts for the model. The index is 'id', used for identifying studies. Other columns are features (e.g., unigrams and bigrams from Neurosynth), where each value is the number of times the feature is found in a given article.
- **coordinates\_df** (`pandas.DataFrame`) – A DataFrame with a list of foci in the dataset. The index is 'id', used for identifying studies. Additional columns include 'i', 'j' and 'k' (the matrix indices of the foci in standard space).
- **mask** (`str` or `nibabel.Nifti1.Nifti1Image`) – Mask file or image.

## Notes

MAPBOT uses both the reported foci for studies, as well as associated term weights. Here are the steps:

1. For each voxel in the mask, identify studies in dataset corresponding to that voxel. Selection criteria can be either based on a distance threshold (e.g., all studies with foci within 5mm of voxel) or based on a minimum number of studies (e.g., the 50 studies reporting foci closest to the voxel).
2. For each voxel, compute average frequency of each term across selected studies. This results in an  $n_{\text{voxels}} \times n_{\text{terms}}$  frequency matrix  $F$ .
3. Compute  $n_{\text{voxels}} \times n_{\text{voxels}}$  value matrix  $V$ :  $D = (F.T * F) * \text{ones}(F)$  -  $V = F * D^{-.5}$
4. Perform non-negative matrix factorization on value matrix.

**Warning:** This method is not yet implemented.

## References

## Methods

<code>fit(self, target_mask[, method, r, n_exps, ...])</code>	Run MAPBOT parcellation.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *target\_mask*, *method*='min\_distance', *r*=5, *n\_exps*=50, *n\_parcels*=2)  
Run MAPBOT parcellation.

### Parameters

- **region\_name** (*str*) – Name of region for parcellation.
- **n\_parcels** (*int*, optional) – Number of parcels to generate for ROI. If array\_like, each parcel number will be evaluated and results for all will be returned. Default is 2.

**get\_params** (*self*, *deep*=True)  
Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed*=True)  
Load a pickled class instance from file.

### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** `obj` – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress=True*)

Pickle the class instance to the provided file.

#### Parameters

- **filename** (`str`) – File to which object will be saved.
- **compress** (`bool`, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

#### Returns

**Return type** `self`

**class** **MAPBOT** (*tfidf\_df*, *coordinates\_df*, *mask*)

Meta-analytic parcellation based on text (MAPBOT) [R84461f98615b-1].

#### Parameters

- **tfidf\_df** (`pandas.DataFrame`) – A DataFrame with feature counts for the model. The index is 'id', used for identifying studies. Other columns are features (e.g., unigrams and bigrams from Neurosynth), where each value is the number of times the feature is found in a given article.
- **coordinates\_df** (`pandas.DataFrame`) – A DataFrame with a list of foci in the dataset. The index is 'id', used for identifying studies. Additional columns include 'i', 'j' and 'k' (the matrix indices of the foci in standard space).
- **mask** (`str` or `nibabel.Nifti1.Nifti1Image`) – Mask file or image.

## Notes

MAPBOT uses both the reported foci for studies, as well as associated term weights. Here are the steps:

1. For each voxel in the mask, identify studies in dataset corresponding to that voxel. Selection criteria can be either based on a distance threshold (e.g., all studies with foci within 5mm of voxel) or based on a minimum number of studies (e.g., the 50 studies reporting foci closest to the voxel).
2. For each voxel, compute average frequency of each term across selected studies. This results in an `n_voxels X n_terms` frequency matrix `F`.
3. Compute `n_voxels X n_voxels` value matrix `V`: -  $D = (F.T * F) * \text{ones}(F)$  -  $V = F * D^{-.5}$
4. Perform non-negative matrix factorization on value matrix.

**Warning:** This method is not yet implemented.

## References

## Methods

<code>fit(self, target_mask[, method, r, n_exps, ...])</code>	Run MAPBOT parcellation.
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>load(filename[, compressed])</code>	Load a pickled class instance from file.
<code>save(self, filename[, compress])</code>	Pickle the class instance to the provided file.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

**fit** (*self*, *target\_mask*, *method*='min\_distance', *r*=5, *n\_exps*=50, *n\_parcel*s=2)  
Run MAPBOT parcellation.

### Parameters

- **region\_name** (*str*) – Name of region for parcellation.
- **n\_parcel**s (*int*, optional) – Number of parcels to generate for ROI. If array\_like, each parcel number will be evaluated and results for all will be returned. Default is 2.

**get\_params** (*self*, *deep*=True)  
Get parameters for this estimator.

**Parameters** *deep* (*boolean*, *optional*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

**Returns** *params* – Parameter names mapped to their values.

**Return type** mapping of string to any

**classmethod load** (*filename*, *compressed*=True)  
Load a pickled class instance from file.

### Parameters

- **filename** (*str*) – Name of file containing object.
- **compressed** (*bool*, optional) – If True, the file is assumed to be compressed and gzip will be used to load it. Otherwise, it will assume that the file is not compressed. Default = True.

**Returns** *obj* – Loaded class object.

**Return type** class object

**save** (*self*, *filename*, *compress*=True)  
Pickle the class instance to the provided file.

### Parameters

- **filename** (*str*) – File to which object will be saved.
- **compress** (*bool*, optional) – If True, the file will be compressed with gzip. Otherwise, the uncompressed version will be saved. Default = True.

**set\_params** (*self*, *\*\*params*)  
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>\_\_<parameter> so that it's possible to update each component of a nested object.

### Returns

Return type `self`

## 6.7 nimare.io: Input/Output

Input/Output operations.

<code>nimare.io</code>	Input/Output operations.
<code>nimare.io.convert_neurosynth_to_json(...)</code>	[Convert Neurosynth dataset text file to a NiMARE json file.
<code>nimare.io.convert_sleuth_to_dict(text_file)</code>	Convert Sleuth text file to a dictionary.
<code>nimare.io.convert_sleuth_to_json(text_file, ...)</code>	Convert Sleuth output text file into json.
<code>nimare.io.convert_sleuth_to_dataset(text_file)</code>	Convert Sleuth output text file into dictionary and create NiMARE Dataset with dictionary.

### 6.7.1 nimare.io.convert\_neurosynth\_to\_json

**convert\_neurosynth\_to\_json** (*text\_file*, *out\_file*, *annotations\_file=None*)

Convert Neurosynth dataset text file to a NiMARE json file.

#### Parameters

- **text\_file** (*str*) – Text file with Neurosynth’s coordinates. Normally named “database.txt”.
- **out\_file** (*str*) – Output NiMARE-format json file.
- **annotations\_file** (*str* or *None*, optional) – Optional file with Neurosynth’s annotations. Normally named “features.txt”. Default is *None*.

### 6.7.2 nimare.io.convert\_sleuth\_to\_dict

**convert\_sleuth\_to\_dict** (*text\_file*)

Convert Sleuth text file to a dictionary.

**Parameters** **text\_file** (*str*) – Path to Sleuth-format text file.

**Returns** **dict\_** – NiMARE-organized dictionary containing experiment information from text file.

**Return type** **dict**

### 6.7.3 nimare.io.convert\_sleuth\_to\_json

**convert\_sleuth\_to\_json** (*text\_file*, *out\_file*)

Convert Sleuth output text file into json.

#### Parameters

- **text\_file** (*str*) – Path to Sleuth-format text file.
- **out\_file** (*str*) – Path to output json file.

## 6.7.4 `nimare.io.convert_sleuth_to_dataset`

**`convert_sleuth_to_dataset`** (*text\_file*, *target*='ale\_2mm')

Convert Sleuth output text file into dictionary and create NiMARE Dataset with dictionary.

**Parameters**

- **`text_file`** (*str*) – Path to Sleuth-format text file.
- **`target`** ({'ale\_2mm', 'mni152\_2mm'}, *optional*) – Target template space for coordinates. Default is 'ale\_2mm' (ALE-specific brainmask in MNI152 2mm space).

**Returns** Dataset object containing experiment information from *text\_file*.

**Return type** `nimare.dataset.Dataset`

## 6.8 `nimare.stats`: Statistical functions

Various statistical helper functions

<code>nimare.stats</code>	Various statistical helper functions
<code>nimare.stats.one_way</code> (data, n)	One-way chi-square test of independence.
<code>nimare.stats.two_way</code> (cells)	Two-way chi-square test of independence.
<code>nimare.stats.pearson</code> (x, y)	Correlates row vector x with each row vector in 2D array y.
<code>nimare.stats.null_to_p</code> (test_value, null_array)	Return two-sided p-value for test value against null array.
<code>nimare.stats.p_to_z</code> (p[, tail])	Convert p-values to z-values.
<code>nimare.stats.t_to_z</code> (t_values, dof)	From Vanessa Sochat's TtoZ package.
<code>nimare.stats.fdr</code> (p[, q])	Determine FDR threshold given a p value array and desired false discovery rate q.

### 6.8.1 `nimare.stats.one_way`

**`one_way`** (*data*, *n*)

One-way chi-square test of independence. Takes a 1D array as input and compares activation at each voxel to proportion expected under a uniform distribution throughout the array. Note that if you're testing activation with this, make sure that only valid voxels (e.g., in-mask gray matter voxels) are included in the array, or results won't make any sense!

**Returns**

**Return type** Chi2 values

### 6.8.2 `nimare.stats.two_way`

**`two_way`** (*cells*)

Two-way chi-square test of independence. Takes a 3D array as input: N(voxels) x 2 x 2, where the last two dimensions are the contingency table for each of N voxels. Returns an array of chi2 values.



### 6.8.3 `nimare.stats.pearson`

**pearson** (*x*, *y*)

Correlates row vector *x* with each row vector in 2D array *y*.

**Parameters**

- **x** ((1, N) array\_like)–
- **y** ((M, N) array\_like)–

### 6.8.4 `nimare.stats.null_to_p`

**null\_to\_p** (*test\_value*, *null\_array*, *tail*='two')

Return two-sided p-value for test value against null array.

### 6.8.5 `nimare.stats.p_to_z`

**p\_to\_z** (*p*, *tail*='two')

Convert p-values to z-values.

### 6.8.6 `nimare.stats.t_to_z`

**t\_to\_z** (*t\_values*, *dof*)

From Vanessa Sochat's TtoZ package.

### 6.8.7 `nimare.stats.fdr`

**fdr** (*p*, *q*=0.05)

Determine FDR threshold given a p value array and desired false discovery rate *q*.

## 6.9 `nimare.utils`: Utility functions and submodules

#### Utilities

<code>nimare.utils</code>	Utilities
<code>nimare.utils.get_template([space, mask])</code>	Load template file.
<code>nimare.utils.listify(obj)</code>	Wraps all non-list or tuple objects in a list; provides a simple way to accept flexible arguments.
<code>nimare.utils.round2(ndarray)</code>	Numpy rounds X.5 values to nearest even integer.
<code>nimare.utils.vox2mm(ijk, affine)</code>	Convert matrix subscripts to coordinates.
<code>nimare.utils.mm2vox(xyz, affine)</code>	Convert coordinates to matrix subscripts.
<code>nimare.utils.tal2mni(coords)</code>	Python version of BrainMap's tal2icbm_other.m.
<code>nimare.utils.mni2tal(coords)</code>	Python version of BrainMap's icbm_other2tal.m.
<code>nimare.utils.get_resource_path()</code>	Returns the path to general resources, terminated with separator.

### 6.9.1 `nimare.utils.get_template`

**get\_template** (*space*='mni152\_1mm', *mask*=None)

Load template file.

**Parameters**

- **space** ({'mni152\_1mm', 'mni152\_2mm', 'ale\_2mm'}, *optional*) – Template to load. Default is 'mni152\_1mm'.
- **mask** ({None, 'brain', 'gm'}, *optional*) – Whether to return the raw template (None), a brain mask ('brain'), or a gray-matter mask ('gm'). Default is None.

**Returns** `img` – Template image object.

**Return type** `nibabel.nifti1.Nifti1Image`

### 6.9.2 `nimare.utils.listify`

**listify** (*obj*)

Wraps all non-list or tuple objects in a list; provides a simple way to accept flexible arguments.

### 6.9.3 `nimare.utils.round2`

**round2** (*ndarray*)

Numpy rounds X.5 values to nearest even integer. We want to round to the nearest integer away from zero.

### 6.9.4 `nimare.utils.vox2mm`

**vox2mm** (*ijk*, *affine*)

Convert matrix subscripts to coordinates.  
[how-to-convert-between-voxel-and-mm.html](http://blog.chrisgorgolewski.org/2014/12/how-to-convert-between-voxel-and-mm.html)

From here: <http://blog.chrisgorgolewski.org/2014/12/how-to-convert-between-voxel-and-mm.html>

### 6.9.5 `nimare.utils.mm2vox`

**mm2vox** (*xyz*, *affine*)

Convert coordinates to matrix subscripts.  
[how-to-convert-between-voxel-and-mm.html](http://blog.chrisgorgolewski.org/2014/12/how-to-convert-between-voxel-and-mm.html)

From here: <http://blog.chrisgorgolewski.org/2014/12/how-to-convert-between-voxel-and-mm.html>

### 6.9.6 `nimare.utils.tal2mni`

**tal2mni** (*coords*)

Python version of BrainMap's tal2icbm\_other.m. This function converts coordinates from Talairach space to MNI space (normalized using templates other than those contained in SPM and FSL) using the tal2icbm transform developed and validated by Jack Lancaster at the Research Imaging Center in San Antonio, Texas. <http://www3.interscience.wiley.com/cgi-bin/abstract/114104479/ABSTRACT> outputs = tal2icbm\_other(inpoints) Where inpoints is N by 3 or 3 by N matrix of coordinates (N being the number of points) ric.uthscsa.edu 3/14/07

## 6.9.7 `nimare.utils.mni2tal`

### `mni2tal` (*coords*)

Python version of BrainMap's `icbm_other2tal.m`. This function converts coordinates from MNI space (normalized using templates other than those contained in SPM and FSL) to Talairach space using the `icbm2tal` transform developed and validated by Jack Lancaster at the Research Imaging Center in San Antonio, Texas. <http://www3.interscience.wiley.com/cgi-bin/abstract/114104479/ABSTRACT> `FORMAT` `outpoints = icbm_other2tal(inpoints)` Where `inpoints` is N by 3 or 3 by N matrix of coordinates (N being the number of points) `ric.uthscsa.edu` 3/14/07

## 6.9.8 `nimare.utils.get_resource_path`

### `get_resource_path()`

Returns the path to general resources, terminated with separator. Resources are kept outside package folder in “datasets”. Based on function by Yaroslav Halchenko used in Neurosynth Python package.

## 6.10 `nimare.workflows`: Common workflows

Common meta-analytic workflows

<code>nimare.workflows</code>	Common meta-analytic workflows
<code>nimare.workflows.ale</code>	Workflow for running an ALE meta-analysis from a Sleuth text file.
<code>nimare.workflows.conperm</code>	Workflow for running a contrast permutation meta-analysis on a set of images.
<code>nimare.workflows.macm</code>	Perform MACM with ALE algorithm.
<code>nimare.workflows.peaks2maps</code>	Workflow for contrast permutation meta-analysis on images constructed from coordinates using the Peaks2Maps kernel.
<code>nimare.workflows.scale</code>	Workflow for running a SCALE meta-analysis from a Sleuth text file.

### 6.10.1 `nimare.workflows.ale`

Workflow for running an ALE meta-analysis from a Sleuth text file.

#### Functions

<code>ale_sleuth_workflow(sleuth_file[, ...])</code>	Perform ALE meta-analysis from Sleuth text file.
--	--

#### `nimare.workflows.ale.ale_sleuth_workflow`

`ale_sleuth_workflow(sleuth_file, sleuth_file2=None, output_dir=None, prefix=None, n_iters=10000, v_thr=0.001, fwhm=None, n_cores=-1)`

Perform ALE meta-analysis from Sleuth text file.

`ale_sleuth_workflow(sleuth_file, sleuth_file2=None, output_dir=None, prefix=None, n_iters=10000, v_thr=0.001, fwhm=None, n_cores=-1)`

Perform ALE meta-analysis from Sleuth text file.

## 6.10.2 nimare.workflows.conperm

Workflow for running a contrast permutation meta-analysis on a set of images.

### Functions

---

<code>conperm_workflow(contrast_images[, ...])</code>	Contrast permutation workflow.
---	--------------------------------

---

**`nimare.workflows.conperm.conperm_workflow`**

**`conperm_workflow`** (*contrast\_images*, *output\_dir=None*, *prefix=""*, *n\_iters=10000*)

Contrast permutation workflow.

**`conperm_workflow`** (*contrast\_images*, *output\_dir=None*, *prefix=""*, *n\_iters=10000*)

Contrast permutation workflow.

## 6.10.3 nimare.workflows.macm

Perform MACM with ALE algorithm.

### Functions

---

<code>macm_workflow(dataset_file, mask_file[, ...])</code>	Perform MACM with ALE algorithm.
--	----------------------------------

---

**`nimare.workflows.macm.macm_workflow`**

**`macm_workflow`** (*dataset\_file*, *mask\_file*, *output\_dir=None*, *prefix=None*, *n\_iters=10000*, *v\_thr=0.001*,  
*n\_cores=-1*)

Perform MACM with ALE algorithm.

**`macm_workflow`** (*dataset\_file*, *mask\_file*, *output\_dir=None*, *prefix=None*, *n\_iters=10000*, *v\_thr=0.001*,  
*n\_cores=-1*)

Perform MACM with ALE algorithm.

## 6.10.4 nimare.workflows.peaks2maps

Workflow for contrast permutation meta-analysis on images constructed from coordinates using the Peaks2Maps kernel.

### Functions

---

<code>peaks2maps_workflow(sleuth_file[, ...])</code>
--

---

**`nimare.workflows.peaks2maps.peaks2maps_workflow`**

**`peaks2maps_workflow`** (*sleuth\_file*, *output\_dir=None*, *prefix=None*, *n\_iters=10000*)

**`peaks2maps_workflow`** (*sleuth\_file*, *output\_dir=None*, *prefix=None*, *n\_iters=10000*)

## 6.10.5 nimare.workflows.scale

Workflow for running a SCALE meta-analysis from a Sleuth text file.

### Functions

<code>scale_workflow(dataset_file, baseline[, ...])</code>	Perform SCALE meta-analysis from Sleuth text file or NiMARE json file.
--	--

#### `nimare.workflows.scale.scale_workflow`

**scale\_workflow** (*dataset\_file, baseline, output\_dir=None, prefix=None, n\_iters=2500, v\_thr=0.001*)  
Perform SCALE meta-analysis from Sleuth text file or NiMARE json file.

**Warning:** This method is not yet implemented.

**scale\_workflow** (*dataset\_file, baseline, output\_dir=None, prefix=None, n\_iters=2500, v\_thr=0.001*)  
Perform SCALE meta-analysis from Sleuth text file or NiMARE json file.

**Warning:** This method is not yet implemented.

## 6.11 nimare.base: Base classes

Base classes for datasets.

<code>nimare.base</code>	Base classes for datasets.
<code>nimare.base.base</code>	
<code>nimare.base.estimators</code>	



## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`





---

## Bibliography

---

- [Rbfa6233af19f-1] Turkeltaub, Peter E., et al. "Meta-analysis of the functional neuroanatomy of single-word reading: method and validation." *Neuroimage* 16.3 (2002): 765-780.
- [Rbfa6233af19f-2] Turkeltaub, Peter E., et al. "Minimizing within-experiment and within-group effects in activation likelihood estimation meta-analyses." *Human brain mapping* 33.1 (2012): 1-13.
- [Rbfa6233af19f-3] Eickhoff, Simon B., et al. "Activation likelihood estimation meta-analysis revisited." *Neuroimage* 59.3 (2012): 2349-2361.
- [Rfe11a1766f4a-1] Laird, Angela R., et al. "ALE meta-analysis: Controlling the false discovery rate and performing statistical contrasts." *Human brain mapping* 25.1 (2005): 155-164. <https://doi.org/10.1002/hbm.20136>
- [Rfe11a1766f4a-2] Eickhoff, Simon B., et al. "Activation likelihood estimation meta-analysis revisited." *Neuroimage* 59.3 (2012): 2349-2361. <https://doi.org/10.1016/j.neuroimage.2011.09.017>
- [Re194cb71ed63-1] Langner, Robert, et al. "Meta-analytic connectivity modeling revisited: controlling for activation base rates." *NeuroImage* 99 (2014): 559-570. <https://doi.org/10.1016/j.neuroimage.2014.06.007>
- [Rbfa6233af19f-1] Turkeltaub, Peter E., et al. "Meta-analysis of the functional neuroanatomy of single-word reading: method and validation." *Neuroimage* 16.3 (2002): 765-780.
- [Rbfa6233af19f-2] Turkeltaub, Peter E., et al. "Minimizing within-experiment and within-group effects in activation likelihood estimation meta-analyses." *Human brain mapping* 33.1 (2012): 1-13.
- [Rbfa6233af19f-3] Eickhoff, Simon B., et al. "Activation likelihood estimation meta-analysis revisited." *Neuroimage* 59.3 (2012): 2349-2361.
- [Rfe11a1766f4a-1] Laird, Angela R., et al. "ALE meta-analysis: Controlling the false discovery rate and performing statistical contrasts." *Human brain mapping* 25.1 (2005): 155-164. <https://doi.org/10.1002/hbm.20136>
- [Rfe11a1766f4a-2] Eickhoff, Simon B., et al. "Activation likelihood estimation meta-analysis revisited." *Neuroimage* 59.3 (2012): 2349-2361. <https://doi.org/10.1016/j.neuroimage.2011.09.017>
- [Re194cb71ed63-1] Langner, Robert, et al. "Meta-analytic connectivity modeling revisited: controlling for activation base rates." *NeuroImage* 99 (2014): 559-570. <https://doi.org/10.1016/j.neuroimage.2014.06.007>
- [R258c842da77f-1] Wager, Tor D., et al. "Valence, gender, and lateralization of functional brain anatomy in emotion: a meta-analysis of findings from neuroimaging." *Neuroimage* 19.3 (2003): 513-531. [https://doi.org/10.1016/S1053-8119\(03\)00078-8](https://doi.org/10.1016/S1053-8119(03)00078-8)
- [R258c842da77f-2] Wager, Tor D., John Jonides, and Susan Reading. "Neuroimaging studies of shifting attention: a meta-analysis." *Neuroimage* 22.4 (2004): 1679-1693. <https://doi.org/10.1016/j.neuroimage.2004.03.052>

- [Rb50f9c63f995-1] Wager, Tor D., Martin Lindquist, and Lauren Kaplan. "Meta-analysis of functional neuroimaging data: current and future directions." *Social cognitive and affective neuroscience* 2.2 (2007): 150-158. <https://doi.org/10.1093/scan/nsm015>
- [R8ac5f6c30bba-1] Wager, Tor D., Martin Lindquist, and Lauren Kaplan. "Meta-analysis of functional neuroimaging data: current and future directions." *Social cognitive and affective neuroscience* 2.2 (2007): 150-158. <https://doi.org/10.1093/scan/nsm015>
- [R258c842da77f-1] Wager, Tor D., et al. "Valence, gender, and lateralization of functional brain anatomy in emotion: a meta-analysis of findings from neuroimaging." *Neuroimage* 19.3 (2003): 513-531. [https://doi.org/10.1016/S1053-8119\(03\)00078-8](https://doi.org/10.1016/S1053-8119(03)00078-8)
- [R258c842da77f-2] Wager, Tor D., John Jonides, and Susan Reading. "Neuroimaging studies of shifting attention: a meta-analysis." *Neuroimage* 22.4 (2004): 1679-1693. <https://doi.org/10.1016/j.neuroimage.2004.03.052>
- [Rb50f9c63f995-1] Wager, Tor D., Martin Lindquist, and Lauren Kaplan. "Meta-analysis of functional neuroimaging data: current and future directions." *Social cognitive and affective neuroscience* 2.2 (2007): 150-158. <https://doi.org/10.1093/scan/nsm015>
- [R8ac5f6c30bba-1] Wager, Tor D., Martin Lindquist, and Lauren Kaplan. "Meta-analysis of functional neuroimaging data: current and future directions." *Social cognitive and affective neuroscience* 2.2 (2007): 150-158. <https://doi.org/10.1093/scan/nsm015>
- [Rbb0a73000b2f-1] Kang, Jian, et al. "Meta analysis of functional neuroimaging data via Bayesian spatial point processes." *Journal of the American Statistical Association* 106.493 (2011): 124-134. <https://doi.org/10.1198/jasa.2011.ap09735>
- [Rb6084ac49a63-1] Kang, Jian, et al. "A Bayesian hierarchical spatial point process model for multi-type neuroimaging meta-analysis." *The annals of applied statistics* 8.3 (2014): 1800.
- [R270830ed9ee2-1] Montagna, Silvia, et al. "Spatial Bayesian latent factor regression modeling of coordinate-based meta-analysis data." *Biometrics* 74.1 (2018): 342-353. <https://doi.org/10.1111/biom.12713>
- [Rbf3e8ffed16f-1] Yue, Yu Ryan, Martin A. Lindquist, and Ji Meng Loh. "Meta-analysis of functional neuroimaging data using Bayesian nonparametric binary regression." *The Annals of Applied Statistics* 6.2 (2012): 697-718. <https://doi.org/10.1214/11-AOAS523>
- [Rbb0a73000b2f-1] Kang, Jian, et al. "Meta analysis of functional neuroimaging data via Bayesian spatial point processes." *Journal of the American Statistical Association* 106.493 (2011): 124-134. <https://doi.org/10.1198/jasa.2011.ap09735>
- [Rb6084ac49a63-1] Kang, Jian, et al. "A Bayesian hierarchical spatial point process model for multi-type neuroimaging meta-analysis." *The annals of applied statistics* 8.3 (2014): 1800.
- [R270830ed9ee2-1] Montagna, Silvia, et al. "Spatial Bayesian latent factor regression modeling of coordinate-based meta-analysis data." *Biometrics* 74.1 (2018): 342-353. <https://doi.org/10.1111/biom.12713>
- [Rbf3e8ffed16f-1] Yue, Yu Ryan, Martin A. Lindquist, and Ji Meng Loh. "Meta-analysis of functional neuroimaging data using Bayesian nonparametric binary regression." *The Annals of Applied Statistics* 6.2 (2012): 697-718. <https://doi.org/10.1214/11-AOAS523>
- [1] Turner, Jessica A., and Angela R. Laird. "The cognitive paradigm ontology: design and application." *Neuroinformatics* 10.1 (2012): 57-66. <https://doi.org/10.1007/s12021-011-9126-x>
- [2] Riedel, Michael Cody, et al. "Automated, efficient, and accelerated knowledge modeling of the cognitive neuroimaging literature using the ATHENA toolkit." *Frontiers in neuroscience* 13 (2019): 494. <https://doi.org/10.3389/fnins.2019.00494>
- [1] Poldrack, Russell A., et al. "The cognitive atlas: toward a knowledge foundation for cognitive neuroscience." *Frontiers in neuroinformatics* 5 (2011): 17. <https://doi.org/10.3389/fninf.2011.00017>

- [R51a14fd76a1a-1] Poldrack, Russell A., et al. "The cognitive atlas: toward a knowledge foundation for cognitive neuroscience." *Frontiers in neuroinformatics* 5 (2011): 17. <https://doi.org/10.3389/fninf.2011.00017>
- [R3153eaf72258-1] Monti, Ricardo, et al. "Text-mining the NeuroSynth corpus using deep Boltzmann machines." 2016 International Workshop on Pattern Recognition in NeuroImaging (PRNI). IEEE, 2016. <https://doi.org/10.1109/PRNI.2016.7552329>
- [R1dd767d5a89c-1] Rubin, Timothy N., et al. "Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition." *PLoS computational biology* 13.10 (2017): e1005649. <https://doi.org/10.1371/journal.pcbi.1005649>
- [1] Newman, D., Asuncion, A., Smyth, P., & Welling, M. (2009). Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10(Aug), 1801-1828.
- [R6c0c7b2ff831-1] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." *Journal of machine Learning research* 3.Jan (2003): 993-1022.
- [R6c0c7b2ff831-2] McCallum, Andrew Kachites. "Mallet: A machine learning for language toolkit." (2002).
- [R6c0c7b2ff831-3] Poldrack, Russell A., et al. "Discovering relations between mind, brain, and mental disorders using topic mapping." *PLoS computational biology* 8.10 (2012): e1002707. <https://doi.org/10.1371/journal.pcbi.1002707>
- [R3b75f33f3695-1] Nunes, Abraham. "word2brain." *bioRxiv* (2018): 299024. <https://doi.org/10.1101/299024>
- [Rce8bb220c1e2-1] Dockès, Jérôme, et al. "Text to brain: predicting the spatial distribution of neuroimaging observations from text reports." *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, Cham, 2018. [https://doi.org/10.1007/978-3-030-00931-1\\_67](https://doi.org/10.1007/978-3-030-00931-1_67)
- [1] Amft, Maren, et al. "Definition and characterization of an extended social-affective default network." *Brain Structure and Function* 220.2 (2015): 1031-1049. <https://doi.org/10.1007/s00429-013-0698-0>
- [1] Rubin, Timothy N., et al. "Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition." *PLoS computational biology* 13.10 (2017): e1005649. <https://doi.org/10.1371/journal.pcbi.1005649>
- [1] Yarkoni, Tal, et al. "Large-scale automated synthesis of human functional neuroimaging data." *Nature methods* 8.8 (2011): 665. <https://doi.org/10.1038/nmeth.1635>
- [1] Amft, Maren, et al. "Definition and characterization of an extended social-affective default network." *Brain Structure and Function* 220.2 (2015): 1031-1049. <https://doi.org/10.1007/s00429-013-0698-0>
- [1] Rubin, Timothy N., et al. "Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition." *PLoS computational biology* 13.10 (2017): e1005649. <https://doi.org/10.1371/journal.pcbi.1005649>
- [1] Yarkoni, Tal, et al. "Large-scale automated synthesis of human functional neuroimaging data." *Nature methods* 8.8 (2011): 665. <https://doi.org/10.1038/nmeth.1635>
- [1] Rubin, Timothy N., et al. "Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition." *PLoS computational biology* 13.10 (2017): e1005649. <https://doi.org/10.1371/journal.pcbi.1005649>
- [1] Rubin, Timothy N., et al. "Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition." *PLoS computational biology* 13.10 (2017): e1005649. <https://doi.org/10.1371/journal.pcbi.1005649>
- [1] Rubin, Timothy N., et al. "Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition." *PLoS computational biology* 13.10 (2017): e1005649. <https://doi.org/10.1371/journal.pcbi.1005649>

- [1] Dockès, Jérôme, et al. “Text to brain: predicting the spatial distribution of neuroimaging observations from text reports.” International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham, 2018. [https://doi.org/10.1007/978-3-030-00931-1\\_67](https://doi.org/10.1007/978-3-030-00931-1_67)
- [1] Rubin, Timothy N., et al. “Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition.” PLoS computational biology 13.10 (2017): e1005649. <https://doi.org/10.1371/journal.pcbi.1005649>
- [1] Dockès, Jérôme, et al. “Text to brain: predicting the spatial distribution of neuroimaging observations from text reports.” International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham, 2018. [https://doi.org/10.1007/978-3-030-00931-1\\_67](https://doi.org/10.1007/978-3-030-00931-1_67)
- [R759c29a4323f-1] Bzdok, D., Laird, A. R., Zilles, K., Fox, P. T., & Eickhoff, S. B. (2013). An investigation of the structural, connectional, and functional subspecialization in the human amygdala. Human brain mapping, 34(12), 3247-3266. <https://doi.org/10.1002/hbm.22138>
- [R759c29a4323f-1] Bzdok, D., Laird, A. R., Zilles, K., Fox, P. T., & Eickhoff, S. B. (2013). An investigation of the structural, connectional, and functional subspecialization in the human amygdala. Human brain mapping, 34(12), 3247-3266. <https://doi.org/10.1002/hbm.22138>
- [Re636c01f812e-1] Yang, Yong, et al. “Identifying functional subdivisions in the human brain using meta-analytic activation modeling-based parcellation.” Neuroimage 124 (2016): 300-309. <https://doi.org/10.1016/j.neuroimage.2015.08.027>
- [Re636c01f812e-1] Yang, Yong, et al. “Identifying functional subdivisions in the human brain using meta-analytic activation modeling-based parcellation.” Neuroimage 124 (2016): 300-309. <https://doi.org/10.1016/j.neuroimage.2015.08.027>
- [R84461f98615b-1] Yuan, Rui, et al. “MAPBOT: Meta-analytic parcellation based on text, and its application to the human thalamus.” NeuroImage 157 (2017): 716-732. <https://doi.org/10.1016/j.neuroimage.2017.06.032>
- [R84461f98615b-1] Yuan, Rui, et al. “MAPBOT: Meta-analytic parcellation based on text, and its application to the human thalamus.” NeuroImage 157 (2017): 716-732. <https://doi.org/10.1016/j.neuroimage.2017.06.032>

### n

- `nimare.annotate`, 87
- `nimare.annotate.ontology`, 87
- `nimare.annotate.text`, 96
- `nimare.annotate.topic`, 89
- `nimare.annotate.vector`, 94
- `nimare.base`, 129
- `nimare.correct`, 86
- `nimare.dataset`, 21
- `nimare.decode`, 98
- `nimare.decode.continuous`, 101
- `nimare.decode.discrete`, 98
- `nimare.decode.encode`, 106
- `nimare.io`, 123
- `nimare.meta`, 24
- `nimare.meta.cbma.ale`, 59
- `nimare.meta.cbma.kernel`, 47
- `nimare.meta.cbma.mkda`, 68
- `nimare.meta.cbma.model`, 76
- `nimare.meta.esma`, 25
- `nimare.meta.ibma`, 28
- `nimare.parcellate`, 109
- `nimare.parcellate.cbp`, 109
- `nimare.parcellate.mamp`, 115
- `nimare.parcellate.mapbot`, 119
- `nimare.stats`, 124
- `nimare.utils`, 125
- `nimare.workflows`, 127
- `nimare.workflows.ale`, 127
- `nimare.workflows.conperm`, 128
- `nimare.workflows.macm`, 128
- `nimare.workflows.peaks2maps`, 128
- `nimare.workflows.scale`, 129



## A

ALE (class in *nimare.meta.cbma.ale*), 59, 64  
 ale\_sleuth\_workflow() (in module *nimare.workflows.ale*), 127  
 ALEKernel (class in *nimare.meta.cbma.kernel*), 48, 54  
 ALESubtraction (class in *nimare.meta.cbma.ale*), 61, 65

## B

BHICP (class in *nimare.meta.cbma.model*), 76, 81  
 BoltzmannModel (class in *nimare.annotate.topic*), 89  
 brainmap\_decode() (in module *nimare.decode.discrete*), 98, 100

## C

CogAtLemmatizer (class in *nimare.annotate.ontology*), 88  
 compute\_log\_likelihood() (*GCLDAModel* method), 91  
 conperm\_workflow() (in module *nimare.workflows.conperm*), 128  
 convert\_neurosynth\_to\_json() (in module *nimare.io*), 123  
 convert\_sleuth\_to\_dataset() (in module *nimare.io*), 124  
 convert\_sleuth\_to\_dict() (in module *nimare.io*), 123  
 convert\_sleuth\_to\_json() (in module *nimare.io*), 123  
 CoordCBP (class in *nimare.parcellate.cbp*), 109, 112  
 corr\_decode() (in module *nimare.decode.continuous*), 102, 104  
 corr\_dist\_decode() (in module *nimare.decode.continuous*), 102, 104

## D

Dataset (class in *nimare.dataset*), 21  
 download\_abstracts() (in module *nimare.annotate.text*), 96, 97

## E

encode\_gclda() (in module *nimare.decode.encode*), 106, 107  
 expand\_counts() (in module *nimare.annotate.ontology*), 88  
 extract\_cogat() (in module *nimare.annotate.ontology*), 87  
 extract\_cogpo() (in module *nimare.annotate.ontology*), 87

## F

fdr() (in module *nimare.stats*), 125  
 FDRCorrector (class in *nimare.correct*), 86  
 FFX\_GLM (class in *nimare.meta.ibma*), 28, 38  
 ffx\_glm() (in module *nimare.meta.ibma*), 37, 46  
 Fishers (class in *nimare.meta.ibma*), 29, 39  
 fishers() (in module *nimare.meta.esma*), 25, 26  
 fit() (*ALE* method), 60, 64  
 fit() (*ALESubtraction* method), 61, 65  
 fit() (*BHICP* method), 77, 82  
 fit() (*CoordCBP* method), 110, 113  
 fit() (*FFX\_GLM* method), 28, 38  
 fit() (*Fishers* method), 30, 39  
 fit() (*GCLDAModel* method), 91  
 fit() (*HPGRF* method), 78, 83  
 fit() (*IBMAEstimator* method), 31, 41  
 fit() (*ImCBP* method), 111, 114  
 fit() (*KDA* method), 69, 73  
 fit() (*MAMP* method), 116, 118  
 fit() (*MAPBOT* method), 120, 122  
 fit() (*MFX\_GLM* method), 32, 42  
 fit() (*MKDACHi2* method), 70, 74  
 fit() (*MKDADensity* method), 71, 75  
 fit() (*RFX\_GLM* method), 33, 43  
 fit() (*SBLFR* method), 79, 84  
 fit() (*SBR* method), 80, 85  
 fit() (*SCALE* method), 63, 67  
 fit() (*Stouffers* method), 35, 44  
 fit() (*WeightedStouffers* method), 36, 45



`fsl_glm()` (in module *nimare.meta.ibma*), 37, 47  
`FWECorrector` (class in *nimare.correct*), 86

## G

`gclda_decode_map()` (in module *nimare.decode.continuous*), 103, 105  
`gclda_decode_roi()` (in module *nimare.decode.discrete*), 99, 100  
`GCLDAModel` (class in *nimare.annotate.topic*), 90  
`generate_cooccurrence()` (in module *nimare.annotate.text*), 96, 97  
`generate_counts()` (in module *nimare.annotate.text*), 97  
`get()` (Dataset method), 22  
`get_images()` (Dataset method), 22  
`get_labels()` (Dataset method), 22  
`get_metadata()` (Dataset method), 22  
`get_params()` (ALE method), 60, 64  
`get_params()` (ALEKernel method), 48, 54  
`get_params()` (ALESubtraction method), 62, 66  
`get_params()` (BHICP method), 77, 82  
`get_params()` (BoltzmannModel method), 89  
`get_params()` (CoordCBP method), 110, 113  
`get_params()` (Dataset method), 23  
`get_params()` (FFX\_GLM method), 28, 38  
`get_params()` (Fishers method), 30, 40  
`get_params()` (GCLDAModel method), 91  
`get_params()` (HPGRF method), 78, 83  
`get_params()` (IBMAEstimator method), 31, 41  
`get_params()` (ImCBP method), 112, 115  
`get_params()` (KDA method), 69, 73  
`get_params()` (KDAKernel method), 49, 57  
`get_params()` (KernelTransformer method), 51  
`get_params()` (LDAModel method), 93  
`get_params()` (MAMP method), 117, 118  
`get_params()` (MAPBOT method), 120, 122  
`get_params()` (MFX\_GLM method), 32, 42  
`get_params()` (MKDACHi2 method), 70, 74  
`get_params()` (MKDADensity method), 71, 75  
`get_params()` (MKDAKernel method), 52, 56  
`get_params()` (Peaks2MapsKernel method), 53, 58  
`get_params()` (RFX\_GLM method), 33, 43  
`get_params()` (SBLFR method), 79, 84  
`get_params()` (SBR method), 80, 85  
`get_params()` (SCALE method), 63, 67  
`get_params()` (Stouffers method), 35, 44  
`get_params()` (Text2BrainModel method), 95  
`get_params()` (WeightedStouffers method), 36, 45  
`get_params()` (Word2BrainModel method), 94  
`get_probs()` (GCLDAModel method), 91  
`get_resource_path()` (in module *nimare.utils*), 127  
`get_studies_by_coordinate()` (Dataset method), 23

`get_studies_by_label()` (Dataset method), 23  
`get_studies_by_mask()` (Dataset method), 23  
`get_template()` (in module *nimare.utils*), 126  
`get_texts()` (Dataset method), 23

## H

`HPGRF` (class in *nimare.meta.cbma.model*), 78, 82

## I

`IBMAEstimator` (class in *nimare.meta.ibma*), 30, 40  
`ImCBP` (class in *nimare.parcellate.cbpy*), 111, 114

## K

`KDA` (class in *nimare.meta.cbma.mkda*), 68, 72  
`KDAKernel` (class in *nimare.meta.cbma.kernel*), 49, 57  
`KernelTransformer` (class in *nimare.meta.cbma.kernel*), 50

## L

`LDAModel` (class in *nimare.annotate.topic*), 92  
`lemmatize()` (CogAtLemmatizer method), 88  
`listify()` (in module *nimare.utils*), 126  
`load()` (*nimare.annotate.topic.BoltzmannModel* class method), 89  
`load()` (*nimare.annotate.topic.GCLDAModel* class method), 92  
`load()` (*nimare.annotate.topic.LDAModel* class method), 93  
`load()` (*nimare.annotate.vector.Text2BrainModel* class method), 95  
`load()` (*nimare.annotate.vector.Word2BrainModel* class method), 94  
`load()` (*nimare.dataset.Dataset* class method), 24  
`load()` (*nimare.meta.cbma.ale.ALE* class method), 60, 64  
`load()` (*nimare.meta.cbma.ale.ALESubtraction* class method), 62, 66  
`load()` (*nimare.meta.cbma.ale.SCALE* class method), 63, 67  
`load()` (*nimare.meta.cbma.kernel.ALEKernel* class method), 48, 55  
`load()` (*nimare.meta.cbma.kernel.KDAKernel* class method), 49, 57  
`load()` (*nimare.meta.cbma.kernel.KernelTransformer* class method), 51  
`load()` (*nimare.meta.cbma.kernel.MKDAKernel* class method), 52, 56  
`load()` (*nimare.meta.cbma.kernel.Peaks2MapsKernel* class method), 53, 58  
`load()` (*nimare.meta.cbma.mkda.KDA* class method), 69, 73  
`load()` (*nimare.meta.cbma.mkda.MKDACHi2* class method), 70, 74



`load()` (*nimare.meta.cbma.mkda.MKDADensity* class method), 72, 75  
`load()` (*nimare.meta.cbma.model.BHICP* class method), 77, 82  
`load()` (*nimare.meta.cbma.model.HPGRF* class method), 78, 83  
`load()` (*nimare.meta.cbma.model.SBLFR* class method), 79, 84  
`load()` (*nimare.meta.cbma.model.SBR* class method), 81, 85  
`load()` (*nimare.meta.ibma.FFX\_GLM* class method), 29, 38  
`load()` (*nimare.meta.ibma.Fishers* class method), 30, 40  
`load()` (*nimare.meta.ibma.IBMAEstimator* class method), 31, 41  
`load()` (*nimare.meta.ibma.MFX\_GLM* class method), 32, 42  
`load()` (*nimare.meta.ibma.RFX\_GLM* class method), 33, 43  
`load()` (*nimare.meta.ibma.Stouffers* class method), 35, 44  
`load()` (*nimare.meta.ibma.WeightedStouffers* class method), 36, 46  
`load()` (*nimare.parcellate.cbp.CoordCBP* class method), 110, 113  
`load()` (*nimare.parcellate.cbp.ImCBP* class method), 112, 115  
`load()` (*nimare.parcellate.mamp.MAMP* class method), 117, 118  
`load()` (*nimare.parcellate.mapbot.MAPBOT* class method), 120, 122

## M

`macm_workflow()` (in module *nimare.workflows.macm*), 128  
`MAMP` (class in *nimare.parcellate.mamp*), 116, 117  
`MAPBOT` (class in *nimare.parcellate.mapbot*), 119, 121  
`MFX_GLM` (class in *nimare.meta.ibma*), 32, 41  
`mfx_glm()` (in module *nimare.meta.ibma*), 37, 47  
`MKDACHi2` (class in *nimare.meta.cbma.mkda*), 70, 73  
`MKDADensity` (class in *nimare.meta.cbma.mkda*), 71, 75  
`MKDABKernel` (class in *nimare.meta.cbma.kernel*), 51, 55  
`mm2vox()` (in module *nimare.utils*), 126  
`mni2tal()` (in module *nimare.utils*), 127

## N

`neurosynth_decode()` (in module *nimare.decode.discrete*), 100, 101  
`nimare.annotate` (module), 87  
`nimare.annotate.ontology` (module), 87  
`nimare.annotate.text` (module), 96

`nimare.annotate.topic` (module), 89  
`nimare.annotate.vector` (module), 94  
`nimare.base` (module), 129  
`nimare.correct` (module), 86  
`nimare.dataset` (module), 21  
`nimare.decode` (module), 98  
`nimare.decode.continuous` (module), 101  
`nimare.decode.discrete` (module), 98  
`nimare.decode.encode` (module), 106  
`nimare.io` (module), 123  
`nimare.meta` (module), 24  
`nimare.meta.cbma.ale` (module), 59  
`nimare.meta.cbma.kernel` (module), 47  
`nimare.meta.cbma.mkda` (module), 68  
`nimare.meta.cbma.model` (module), 76  
`nimare.meta.esma` (module), 25  
`nimare.meta.ibma` (module), 28  
`nimare.parcellate` (module), 109  
`nimare.parcellate.cbp` (module), 109  
`nimare.parcellate.mamp` (module), 115  
`nimare.parcellate.mapbot` (module), 119  
`nimare.stats` (module), 124  
`nimare.utils` (module), 125  
`nimare.workflows` (module), 127  
`nimare.workflows.ale` (module), 127  
`nimare.workflows.conperm` (module), 128  
`nimare.workflows.macm` (module), 128  
`nimare.workflows.peaks2maps` (module), 128  
`nimare.workflows.scale` (module), 129  
`null_to_p()` (in module *nimare.stats*), 125

## O

`one_way()` (in module *nimare.stats*), 124  
`ontology` (*CogAtLemmatizer* attribute), 88

## P

`p_to_z()` (in module *nimare.stats*), 125  
`peaks2maps_workflow()` (in module *nimare.workflows.peaks2maps*), 128  
`Peaks2MapsKernel` (class in *nimare.meta.cbma.kernel*), 53, 58  
`pearson()` (in module *nimare.stats*), 125

## R

`regex` (*CogAtLemmatizer* attribute), 88  
`RFX_GLM` (class in *nimare.meta.ibma*), 33, 42  
`rfx_glm()` (in module *nimare.meta.esma*), 25, 27  
`round2()` (in module *nimare.utils*), 126

## S

`save()` (*ALE* method), 60, 65  
`save()` (*ALEKernel* method), 48, 55  
`save()` (*ALESubtraction* method), 62, 66

save() (*BHICP method*), 77, 82  
 save() (*BoltzmannModel method*), 89  
 save() (*CoordCBP method*), 110, 114  
 save() (*Dataset method*), 24  
 save() (*FFX\_GLM method*), 29, 39  
 save() (*Fishers method*), 30, 40  
 save() (*GCLDAModel method*), 92  
 save() (*HPGRF method*), 78, 83  
 save() (*IBMAEstimator method*), 31, 41  
 save() (*ImCBP method*), 112, 115  
 save() (*KDA method*), 69, 73  
 save() (*KDAKernel method*), 50, 57  
 save() (*KernelTransformer method*), 51  
 save() (*LDAModel method*), 93  
 save() (*MAMP method*), 117, 119  
 save() (*MAPBOT method*), 121, 122  
 save() (*MFX\_GLM method*), 32, 42  
 save() (*MKDACHi2 method*), 71, 74  
 save() (*MKDADensity method*), 72, 76  
 save() (*MKDABKernel method*), 52, 56  
 save() (*Peaks2MapsKernel method*), 53, 59  
 save() (*RFX\_GLM method*), 34, 43  
 save() (*SBLFR method*), 80, 84  
 save() (*SBR method*), 81, 86  
 save() (*SCALE method*), 63, 68  
 save() (*Stouffers method*), 35, 45  
 save() (*Text2BrainModel method*), 95  
 save() (*WeightedStouffers method*), 36, 46  
 save() (*Word2BrainModel method*), 94  
 SBLFR (*class in nimare.meta.cbma.model*), 79, 84  
 SBR (*class in nimare.meta.cbma.model*), 80, 85  
 SCALE (*class in nimare.meta.cbma.ale*), 62, 66  
 scale\_workflow() (*in module nimare.workflows.scale*), 129  
 set\_params() (*ALE method*), 61, 65  
 set\_params() (*ALEKernel method*), 49, 55  
 set\_params() (*ALESubtraction method*), 62, 66  
 set\_params() (*BHICP method*), 77, 82  
 set\_params() (*BoltzmannModel method*), 89  
 set\_params() (*CoordCBP method*), 111, 114  
 set\_params() (*Dataset method*), 24  
 set\_params() (*FFX\_GLM method*), 29, 39  
 set\_params() (*Fishers method*), 30, 40  
 set\_params() (*GCLDAModel method*), 92  
 set\_params() (*HPGRF method*), 79, 83  
 set\_params() (*IBMAEstimator method*), 31, 41  
 set\_params() (*ImCBP method*), 112, 115  
 set\_params() (*KDA method*), 69, 73  
 set\_params() (*KDAKernel method*), 50, 58  
 set\_params() (*KernelTransformer method*), 51  
 set\_params() (*LDAModel method*), 93  
 set\_params() (*MAMP method*), 117, 119  
 set\_params() (*MAPBOT method*), 121, 122  
 set\_params() (*MFX\_GLM method*), 33, 42

set\_params() (*MKDACHi2 method*), 71, 75  
 set\_params() (*MKDADensity method*), 72, 76  
 set\_params() (*MKDABKernel method*), 52, 56  
 set\_params() (*Peaks2MapsKernel method*), 54, 59  
 set\_params() (*RFX\_GLM method*), 34, 44  
 set\_params() (*SBLFR method*), 80, 85  
 set\_params() (*SBR method*), 81, 86  
 set\_params() (*SCALE method*), 64, 68  
 set\_params() (*Stouffers method*), 35, 45  
 set\_params() (*Text2BrainModel method*), 95  
 set\_params() (*WeightedStouffers method*), 36, 46  
 set\_params() (*Word2BrainModel method*), 94  
 slice() (*Dataset method*), 24  
 Stouffers (*class in nimare.meta.ibma*), 34, 44  
 stouffers() (*in module nimare.meta.esma*), 26, 27

## T

t\_to\_z() (*in module nimare.stats*), 125  
 tal2mni() (*in module nimare.utils*), 126  
 text2brain() (*in module nimare.decode.encode*), 107, 109  
 Text2BrainModel (*class in nimare.annotate.vector*), 95  
 transform() (*ALEKernel method*), 49, 55  
 transform() (*KDAKernel method*), 50, 58  
 transform() (*KernelTransformer method*), 51  
 transform() (*MKDABKernel method*), 52, 56  
 transform() (*Peaks2MapsKernel method*), 54, 59  
 two\_way() (*in module nimare.stats*), 124

## U

uk\_to\_us() (*in module nimare.annotate.text*), 97, 98  
 update\_path() (*Dataset method*), 24

## V

vox2mm() (*in module nimare.utils*), 126

## W

weighted\_stouffers() (*in module nimare.meta.esma*), 26, 27  
 WeightedStouffers (*class in nimare.meta.ibma*), 35, 45  
 Word2BrainModel (*class in nimare.annotate.vector*), 94