NiMARE

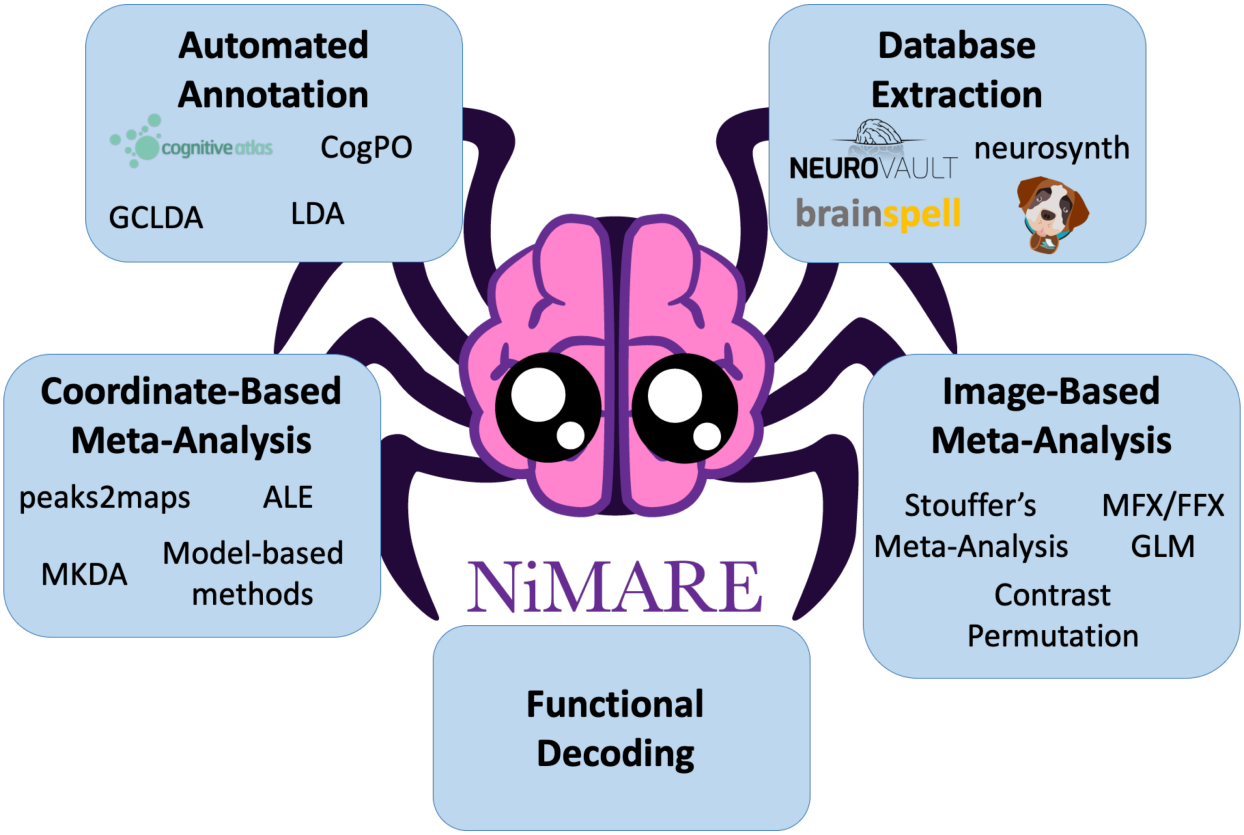
Release 0.0.2+0.g74752b3.dirty

May 12, 2020

Contents:

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.



CHAPTER 1

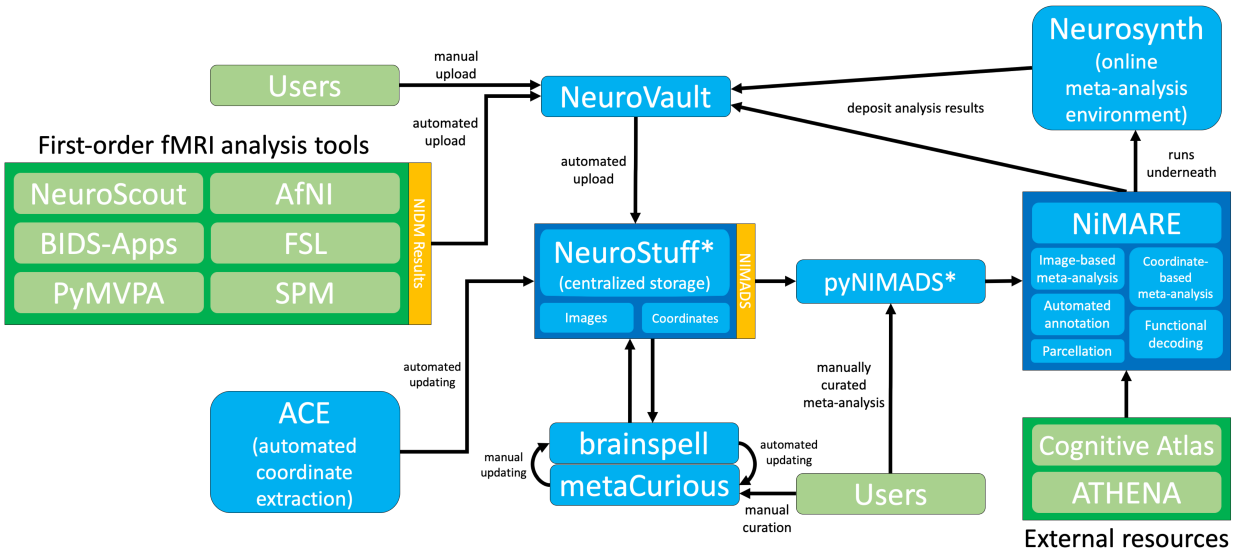
About NiMARE

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 by 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

NiMARE can be installed from pip. To install the latest official release:

```
pip install nimare
```

If you want to use the most up-to-date version, you can install from master:

```
pip install git+https://github.com/neurostuff/NiMARE.git
```

NiMARE requires Python >=3.6 and the following packages:

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

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.

CHAPTER 3

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.

CHAPTER 4

Examples

4.1 Working with datasets

4.1.1 Download and convert the Neurosynth database

Download and convert the Neurosynth database (with abstracts) for analysis with NiMARE.

Note: This will likely change as we work to shift database querying to a remote database, rather than handling it locally with NiMARE.

Start with the necessary imports

```
import os

from neurosynth.base.dataset import download

import nimare
```

Download Neurosynth

```
out_dir = os.path.abspath('../example_data/')
if not os.path.isdir(out_dir):
    os.mkdir(out_dir)

if not os.path.isfile(os.path.join(out_dir, 'database.txt')):
    download(out_dir, unpack=True)
```

Convert Neurosynth database to NiMARE dataset file

```
dset = nimare.io.convert_neurosynth_to_dataset(
    os.path.join(out_dir, 'database.txt'),
    os.path.join(out_dir, 'features.txt'))
dset.save(os.path.join(out_dir, 'neurosynth_dataset.pkl.gz'))
```

Add article abstracts to dataset

```
dset = nimare.extract.download_abstracts(dset, 'tsalo006@fiu.edu')
dset.save(os.path.join(out_dir, 'neurosynth_nimare_with_abstracts.pkl.gz'))
```

Total running time of the script: (0 minutes 0.000 seconds)

4.2 Performing meta-analyses

4.2.1 Generate modeled activation maps with peaks2maps

Start with the necessary imports

```
import os

import numpy as np
import matplotlib.pyplot as plt
from nilearn.plotting import plot_glass_brain

import nimare
from nimare.tests.utils import get_test_data_path
```

Load Dataset

```
dset_file = os.path.join(get_test_data_path(), 'nidm_pain_dset.json')
dset = nimare.dataset.Dataset(dset_file)
```

Run peaks2maps

```
k = nimare.meta.cbma.kernel.Peaks2MapsKernel()
imgs = k.transform(dset, masked=True)
```

Plot modeled activation maps

```
for img in imgs:
    display = plot_glass_brain(img, display_mode='lyrz',
                               plot_abs=False, colorbar=True,
                               vmax=1, threshold=0)
```

Total running time of the script: (0 minutes 0.000 seconds)

4.2.2 Generate modeled activation maps

Start with the necessary imports

```
import os

import numpy as np
import matplotlib.pyplot as plt
from nilearn.plotting import plot_stat_map

import nimare
from nimare.tests.utils import get_test_data_path
```

Load Dataset

```
dset_file = os.path.join(get_test_data_path(), 'nidm_pain_dset.json')
dset = nimare.dataset.Dataset(dset_file)
```

MKDA kernel maps

```
kernel = nimare.meta.cbma.MKDAKernel(r=8)
mkda_r08 = kernel.transform(dset)
kernel = nimare.meta.cbma.MKDAKernel(r=9)
mkda_r09 = kernel.transform(dset)
kernel = nimare.meta.cbma.MKDAKernel(r=10)
mkda_r10 = kernel.transform(dset)
kernel = nimare.meta.cbma.MKDAKernel(r=11)
mkda_r11 = kernel.transform(dset)

fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(10, 17.5))
plot_stat_map(mkda_r08[2], cut_coords=[-2, -10, -4],
              title='r=8mm', vmax=2, axes=axes[0],
              draw_cross=False)
plot_stat_map(mkda_r09[2], cut_coords=[-2, -10, -4],
              title='r=9mm', vmax=2, axes=axes[1],
              draw_cross=False)
plot_stat_map(mkda_r10[2], cut_coords=[-2, -10, -4],
              title='r=10mm', vmax=2, axes=axes[2],
              draw_cross=False)
plot_stat_map(mkda_r11[2], cut_coords=[-2, -10, -4],
              title='r=11mm', vmax=2, axes=axes[3],
              draw_cross=False)
fig.show()
```

Show different kernel types together

```
kernel = nimare.meta.cbma.MKDAKernel(r=10)
mkda_res = kernel.transform(dset)
kernel = nimare.meta.cbma.KDAKernel(r=10)
kda_res = kernel.transform(dset)
kernel = nimare.meta.cbma.ALEKernel(n=20)
```

(continues on next page)

(continued from previous page)

```
ale_res = kernel.transform(dset)
max_conv = np.max(kda_res[2].get_data())
plot_stat_map(ale_res[2], cut_coords=[-2, -10, -4], title='ALE')
plot_stat_map(mkda_res[2], cut_coords=[-2, -10, -4], title='MKDA', vmax=max_conv)
plot_stat_map(kda_res[2], cut_coords=[-2, -10, -4], title='KDA', vmax=max_conv)
```

Total running time of the script: (0 minutes 0.000 seconds)

4.2.3 Run image-based meta-analyses on 21 pain studies

Collection of NIDM-Results packs downloaded from Neurovault collection 1425, uploaded by Dr. Camille Maumet.

Note: This will likely change as we work to shift database querying to a remote database, rather than handling it locally with NiMARE.

Start with the necessary imports

```
import os

import numpy as np
import pandas as pd
import nibabel as nib
from nilearn.masking import apply_mask, unmask
from nilearn.plotting import plot_stat_map

import nimare
from nimare.tests.utils import get_test_data_path
from nimare.meta.esma import fishers
from nimare.meta.ibma import (Fishers, Stouffers, WeightedStouffers,
                              RFX_GLM, FFX_GLM, ffx_glm)
```

Download data

```
dset_dir = nimare.extract.download_nidm_pain()
```

Load Dataset

```
dset_file = os.path.join(get_test_data_path(), 'nidm_pain_dset.json')
dset = nimare.dataset.Dataset(dset_file)
dset.update_path(dset_dir)

mask_img = dset.masker.mask_img

logp_thresh = -np.log(.05)
```

Fisher's (using functions)

Get images for analysis

```
files = dset.get_images(imtype='z')
files = [f for f in files if f]
z_imgs = [nib.load(f) for f in files]
z_data = apply_mask(z_imgs, mask_img)
print('{0} studies found.'.format(z_data.shape[0]))

result = fishers(z_data, mask_img)
fishers_result = unmask(result['z'], mask_img)
plot_stat_map(fishers_result, cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

Fisher's (using Estimators)

Here is the object-oriented approach

```
meta = Fishers()
meta.fit(dset)
plot_stat_map(meta.results.get_map('z'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

Stouffer's with fixed-effects inference

```
meta = Stouffers(inference='ffx', null='theoretical', n_iters=None)
meta.fit(dset)
plot_stat_map(meta.results.get_map('z'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

Stouffer's with random-effects inference using theoretical null distribution

```
meta = Stouffers(inference='rfx', null='theoretical', n_iters=None)
meta.fit(dset)
plot_stat_map(meta.results.get_map('z'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

Stouffer's with random-effects inference using empirical null distribution

```
meta = Stouffers(inference='rfx', null='empirical', n_iters=1000)
meta.fit(dset)
plot_stat_map(meta.results.get_map('z'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

Weighted Stouffer's

```
meta = WeightedStouffers()
meta.fit(dset)
plot_stat_map(meta.results.get_map('z'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

RFX GLM with theoretical null distribution

```
meta = RFX_GLM(null='theoretical', n_iters=None)
meta.fit(dset)
plot_stat_map(meta.results.get_map('z'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

RFX GLM with empirical null distribution

```
meta = RFX_GLM(null='empirical', n_iters=1000)
meta.fit(dset)
plot_stat_map(meta.results.get_map('z'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

Total running time of the script: (0 minutes 0.000 seconds)

4.2.4 Run coordinate-based meta-analyses on 21 pain studies

Collection of NIDM-Results packs downloaded from Neurovault collection 1425, uploaded by Dr. Camille Maumet.

Note: This will likely change as we work to shift database querying to a remote database, rather than handling it locally with NiMARE.

Start with the necessary imports

```
import os
import json
from glob import glob

import numpy as np
import pandas as pd
import nibabel as nib
from scipy.stats import t
from nilearn.masking import apply_mask
from nilearn.plotting import plot_stat_map

import nimare
from nimare.tests.utils import get_test_data_path
```

Load Dataset

```
dset_file = os.path.join(get_test_data_path(), 'nidm_pain_dset.json')
dset = nimare.dataset.Dataset(dset_file)

mask_img = dset.masker.mask_img
```

MKDA density analysis

```
mkda = nimare.meta.cbma.MKDADensity(kernel__r=10)
mkda.fit(dset)
corr = nimare.correct.FWECorrector(method='permutation', n_iters=10, n_cores=1)
cres = corr.transform(mkda.results)
plot_stat_map(cres.get_map('logp_level-voxel_corr-FWE_method-permutation'),
              cut_coords=[0, 0, -8], draw_cross=False, cmap='RdBu_r')
```

MKDA Chi2 with FDR correction

```
mkda = nimare.meta.cbma.MKDAChi2(kernel__r=10)
dset1 = dset.slice(dset.ids)
dset2 = dset.slice(dset.ids)
mkda.fit(dset1, dset2)
corr = nimare.correct.FDRCorrector(method='fdr_bh', alpha=0.001)
cres = corr.transform(mkda.results)
plot_stat_map(cres.get_map('consistency_z_FDR_corr-FDR_method-fdr_bh'),
              threshold=1.65, cut_coords=[0, 0, -8], draw_cross=False,
              cmap='RdBu_r')
```

MKDA Chi2 with FWE correction

```
corr = nimare.correct.FWECorrector(method='permutation', n_iters=10, n_cores=1)
cres = corr.transform(mkda.results)
plot_stat_map(cres.get_map('consistency_z'), threshold=1.65,
              cut_coords=[0, 0, -8], draw_cross=False, cmap='RdBu_r')
```

KDA

```
kda = nimare.meta.cbma.KDA(kernel__r=10)
kda.fit(dset)
corr = nimare.correct.FWECorrector(method='permutation', n_iters=10, n_cores=1)
cres = corr.transform(kda.results)
plot_stat_map(cres.get_map('logp_level-voxel_corr-FWE_method-permutation'),
              cut_coords=[0, 0, -8], draw_cross=False, cmap='RdBu_r')
```

ALE

```
ale = nimare.meta.cbma.ALE()
ale.fit(dset)
corr = nimare.correct.FWECorrector(method='permutation', n_iters=10, n_cores=1)
cres = corr.transform(ale.results)
plot_stat_map(cres.get_map('logp_level-cluster_corr-FWE_method-permutation'),
              cut_coords=[0, 0, -8], draw_cross=False, cmap='RdBu_r')
```

SCALE

```
ijk = np.vstack(np.where(mask_img.get_data())).T
scale = nimare.meta.cbma.SCALE(ijk=ijk, n_iters=10, n_cores=1)
scale.fit(dset)
plot_stat_map(scale.results.get_map('z_vthresh'), cut_coords=[0, 0, -8],
              draw_cross=False, cmap='RdBu_r')
```

Total running time of the script: (0 minutes 0.000 seconds)

4.3 Automated annotation

4.3.1 Train an LDA model and use it

This example trains a latent Dirichlet allocation with MALLET using abstracts from Neurosynth.

Start with the necessary imports

```
import os

import nimare
from nimare import annotate
from nimare.tests.utils import get_test_data_path
```

Load dataset with abstracts

```
dset = nimare.dataset.Dataset.load(
    os.path.join(get_test_data_path(), 'neurosynth_laird_studies.pkl.gz'))
```

Download MALLET

LDAModel will do this automatically.

```
mallet_dir = nimare.extract.download_mallet()
```

Run model

Five iterations will take ~10 minutes

```
model = annotate.topic.LDAModel(dset.texts, text_column='abstract', n_iters=5)
model.fit()
model.save('lda_model.pkl.gz')
```

Total running time of the script: (0 minutes 0.000 seconds)

4.3.2 Train a GCLDA model and use it

This example trains a generalized corresponded latent Dirichlet allocation using abstracts from Neurosynth and then uses it for decoding.

Start with the necessary imports

```
import os

import numpy as np
import nibabel as nib

import nimare
from nimare import annotate, decode
from nimare.tests.utils import get_test_data_path
```

Load dataset with abstracts

```
dset = nimare.dataset.Dataset.load(
    os.path.join(get_test_data_path(), 'neurosynth_laird_studies.pkl.gz'))
```

Generate term counts

```
counts_df = annotate.text.generate_counts(
    dset.texts, text_column='abstract', tfidf=False, max_df=0.99, min_df=0)
```

Run model

Five iterations will take ~10 minutes

```
model = annotate.topic.GCLDAModel(
    counts_df, dset.coordinates, mask=dset.masker.mask_img)
model.fit(n_iters=5, loglikely_freq=5)
model.save('gclda_model.pkl.gz')
```

Decode an ROI image

Make an ROI from a single voxel

```
arr = np.zeros(dset.masker.mask_img.shape, int)
arr[40:44, 45:49, 40:44] = 1
mask_img = nib.Nifti1Image(arr, dset.masker.mask_img.affine)

# Run the decoder
decoded_df, _ = decode.discrete.gclda_decode_roi(model, mask_img)
decoded_df.sort_values(by='Weight', ascending=False).head(10)
```

Total running time of the script: (0 minutes 0.000 seconds)

CHAPTER 5

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.

5.1 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.

5.2 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.

5.3 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.

5.4 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.

5.5 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.

5.6 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!

5.7 Thank you!

You're awesome.

- NOTE: These guidelines are based on contributing guidelines from the STEMMRoleModels project.

CHAPTER 6

NiMARE Developer Guide

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

6.1 Coding Style

NiMARE code should follow PEP8 recommendations. Additionally, we have modeled NiMARE's code on scikit-learn.

CHAPTER 7

API

7.1 nimare.dataset: Dataset IO

Classes for representing datasets of images and/or coordinates.

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

7.1.1 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

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

Continued on next page

Table 2 – continued from previous page

get_images(self[, ids, imtype])	Get images of a certain type for a subset of studies in the dataset.
get_labels(self[, ids])	Extract list of labels for which studies in Dataset have annotations.
get_metadata(self[, ids, field])	Get metadata from Dataset.
get_params(self[, deep])	Get parameters for this estimator.
get_studies_by_coordinate(self, xyz[, r])	Extract list of studies with at least one focus within radius r of requested coordinates.
get_studies_by_label(self[, labels, ...])	Extract list of studies with a given label.
get_studies_by_mask(self, mask)	Extract list of studies with at least one coordinate in mask.
get_texts(self[, ids, text_type])	Extract list of texts of a given type for selected IDs.
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.
slice(self, ids)	Return a reduced dataset with only requested IDs.
update_path(self, new_path)	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** – Redcued 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.

7.2 nimare.meta: Meta-analytic algorithms

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

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

Continued on next page

Table 3 – continued from previous page

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).
nimare.meta.cbma.ale	CBMA methods from the activation likelihood estimation (ALE) family
nimare.meta.cbma.mkda	CBMA methods from the multilevel kernel density analysis (MKDA) family
nimare.meta.cbma.model	Model-based coordinate-based meta-analysis estimators
nimare.meta.base	

7.2.1 nimare.meta.esma

Effect-size meta-analysis functions

Functions

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

7.2.2 nimare.meta.ibma

Image-based meta-analysis estimators

Classes

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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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.

Continued on next page

Table 20 – continued from previous page

set_params(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.results.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

7.2.3 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

ALEKernel([fwhm, n])	Generate ALE modeled activation images from coordinates and sample size.
KDAKernel([r, value])	Generate KDA modeled activation images from coordinates.
KernelTransformer()	Base class for modeled activation-generating methods.
MKDAKernel([r, value])	Generate MKDA modeled activation images from coordinates.
Peaks2MapsKernel([resample_to_mask])	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

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.
transform(self, dataset[, mask, masked])	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

get_params(self[, deep])	Get parameters for this estimator.
load(filename[, compressed])	Load a pickled class instance from file.

Continued on next page

Table 23 – continued from previous page

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

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

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.
transform(self, dataset)	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

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.
transform(self, dataset[, mask, masked])	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

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.
transform(self, dataset[, mask, masked])	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

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.
transform(self, dataset[, mask, masked])	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

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

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.
transform(self, dataset[, mask, masked])	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

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.
transform(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

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.
transform(self, dataset[, mask, masked])	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

7.2.4 nimare.meta.cbma.ale

CBMA methods from the activation likelihood estimation (ALE) family

Classes

ALE([kernel_estimator])	Activation likelihood estimation
ALESubtraction([n_iters])	ALE subtraction analysis.
SCALE([voxel_thresh, n_iters, n_cores, ijk, ...])	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].

Available correction methods: *ALE.correct_fwe_permutation*

References

Methods

correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset)	Fit Estimator to Dataset.
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.

correct_fwe_permutation(*self*, *result*, *voxel_thresh=0.001*, *n_iters=10000*, *n_cores=-1*)

Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from an ALE meta-analysis.
- **voxel_thresh** (float, optional) – Cluster-defining uncorrected p-value threshold. Default is 0.001.
- **n_iters** (int, optional) – Number of iterations to build vFWE and cFWE null distributions. Default is 10000.

- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns **images** – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'z_vthresh', 'p_level-voxel', 'z_level-voxel', and 'logp_level-cluster'.

Return type dict

Notes

This method also adds the following arrays to the Estimator's null distributions attribute (null_distributions): 'fwe_level-voxel' and 'fwe_level-cluster'.

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = ALE()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', voxel_thresh=0.001,
                             n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

fit (*self*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, ale1, ale2[, image1, image2, ...])	Run a subtraction analysis comparing two groups of experiments from separate meta-analyses.
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*, *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

- **ale1/ale2** (*nimare.meta.cbma.ale.ALE*) – Fitted ALE models for datasets to compare.
- **image1/image2** (*img_like or array_like*) – 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.results.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 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 columnns.
- **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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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].

Available correction methods: *ALE.correct_fwe_permutation*

References

Methods

correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset)	Fit Estimator to Dataset.
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.

correct_fwe_permutation (*self*, *result*, *voxel_thresh=0.001*, *n_iters=10000*, *n_cores=-1*)
Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from an ALE meta-analysis.
- **voxel_thresh** (float, optional) – Cluster-defining uncorrected p-value threshold.

Default is 0.001.

- **n_iters** (int, optional) – Number of iterations to build vFWE and cFWE null distributions. Default is 10000.
- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns **images** – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'z_vthresh', 'p_level-voxel', 'z_level-voxel', and 'logp_level-cluster'.

Return type dict

Notes

This method also adds the following arrays to the Estimator's null distributions attribute (null_distributions): 'fwe_level-voxel' and 'fwe_level-cluster'.

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = ALE()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', voxel_thresh=0.001,
                             n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

fit (*self*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, ale1, ale2[, image1, image2, ...])	Run a subtraction analysis comparing two groups of experiments from separate meta-analyses.
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*, *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

- **ale1/ale2** (*nimare.meta.cbma.ale.ALE*) – Fitted ALE models for datasets to compare.
- **image1/image2** (*img_like or array_like*) – 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.results.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 columnns.
- **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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

7.2.5 nimare.meta.cbma.mkda

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

Classes

KDA([kernel_estimator])	Kernel density analysis.
MKDAChi2([prior, kernel_estimator])	Multilevel kernel density analysis- Chi-square analysis [Rb50f9c63f995-1].
MKDADensity([kernel_estimator])	Multilevel kernel density analysis- Density analysis [R8ac5f6c30bba-1].

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].

Available correction methods: *KDA.correct_fwe_permutation*

References

Methods

correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset)	Fit Estimator to Dataset.

Continued on next page

Table 39 – continued from previous page

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.

correct_fwe_permutation (*self*, *result*, *n_iters=10000*, *n_cores=-1*)

Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **n_iters** (int, optional) – Number of iterations to build the vFWE null distribution. Default is 10000.
- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns **images** – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'logp_level-voxel'.

Return type dict

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = KDA()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

fit (*self*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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.

Notes

Available correction methods: *MKDAChi2.correct_fwe_permutation*, *MKDAChi2.correct_fdr_bh*

References

Methods

correct_fdr_bh(self, result[, alpha])	Perform FDR correction using the Benjamini-Hochberg method.
correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset, dataset2)	Fit Estimator to datasets.
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.

correct_fdr_bh(*self*, *result*, *alpha=0.05*)

Perform FDR correction using the Benjamini-Hochberg method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **alpha** (float, optional) – Alpha. Default is 0.05.

Returns images – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'consistency_z_FDR' and 'specificity_z_FDR'.

Return type dict

See also:

nimare.correct.FDRCorrector() The Corrector from which to call this method.

Examples

```
>>> meta = MKDAChi2()
>>> result = meta.fit(dset)
>>> corrector = FDRCorrector(method='bh', alpha=0.05)
>>> cresult = corrector.transform(result)
```

correct_fwe_permutation(*self*, *result*, *n_iters=5000*, *n_cores=-1*)

Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **n_iters** (int, optional) – Number of iterations to build the vFWE null distribution. Default is 5000.
- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns images – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'consistency_p_FWE', 'consistency_z_FWE', 'specificity_p_FWE', and 'specificity_z_FWE'.

Return type dict

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = MKDAChi2()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

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

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

Returns Results of Estimator fitting.

Return type *nimare.results.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.

Notes

Available correction methods: *MKDADensity.correct_fwe_permutation*

References

Methods

correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset)	Fit Estimator to Dataset.
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.

correct_fwe_permutation(*self*, *result*, *voxel_thresh=0.01*, *n_iters=1000*, *n_cores=-1*)

Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **voxel_thresh** (float, optional) – Cluster-defining OF-value threshold. Default is 0.01.
- **n_iters** (int, optional) – Number of iterations to build the vFWE and cFWE null distributions. Default is 1000.
- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns images – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'vthresh', 'logp_level-cluster', and 'logp_level-voxel'.

Return type dict

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = MKDADensity()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', voxel_thresh=0.01,
                             n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

fit (*self*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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].

Available correction methods: *KDA.correct_fwe_permutation*

References

Methods

correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset)	Fit Estimator to Dataset.
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.

correct_fwe_permutation(*self*, *result*, *n_iters=10000*, *n_cores=-1*)

Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **n_iters** (int, optional) – Number of iterations to build the vFWE null distribution. Default is 10000.
- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns **images** – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'logp_level-voxel'.

Return type dict

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = KDA()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

fit (*self*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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'>*, ***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.

Notes

Available correction methods: *MKDAChi2.correct_fwe_permutation*, *MKDAChi2.correct_fdr_bh*

References

Methods

correct_fdr_bh(self, result[, alpha])	Perform FDR correction using the Benjamini-Hochberg method.
correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset, dataset2)	Fit Estimator to datasets.
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.

correct_fdr_bh(*self*, *result*, *alpha=0.05*)

Perform FDR correction using the Benjamini-Hochberg method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **alpha** (float, optional) – Alpha. Default is 0.05.

Returns images – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'consistency_z_FDR' and 'specificity_z_FDR'.

Return type dict

See also:

nimare.correct.FDRCorrector() The Corrector from which to call this method.

Examples

```
>>> meta = MKDAChi2()
>>> result = meta.fit(dset)
>>> corrector = FDRCorrector(method='bh', alpha=0.05)
>>> cresult = corrector.transform(result)
```

correct_fwe_permutation (*self*, *result*, *n_iters=5000*, *n_cores=-1*)

Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **n_iters** (int, optional) – Number of iterations to build the vFWE null distribution. Default is 5000.
- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns **images** – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'consistency_p_FWE', 'consistency_z_FWE', 'specificity_p_FWE', and 'specificity_z_FWE'.

Return type dict

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = MKDAChi2()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

fit (*self*, *dataset*, *dataset2*)

Fit Estimator to datasets.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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.

Notes

Available correction methods: *MKDADensity.correct_fwe_permutation*

References

Methods

correct_fwe_permutation(self, result[, ...])	Perform FWE correction using the max-value permutation method.
fit(self, dataset)	Fit Estimator to Dataset.
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.

correct_fwe_permutation(*self*, *result*, *voxel_thresh=0.01*, *n_iters=1000*, *n_cores=-1*)

Perform FWE correction using the max-value permutation method. Only call this method from within a Corrector.

Parameters

- **result** (*nimare.results.MetaResult*) – Result object from a KDA meta-analysis.
- **voxel_thresh** (float, optional) – Cluster-defining OF-value threshold. Default is 0.01.
- **n_iters** (int, optional) – Number of iterations to build the vFWE and cFWE null distributions. Default is 1000.
- **n_cores** (int, optional) – Number of cores to use for parallelization. If <=0, defaults to using all available cores. Default is -1.

Returns **images** – Dictionary of 1D arrays corresponding to masked images generated by the correction procedure. The following arrays are generated by this method: 'vthresh', 'logp_level-cluster', and 'logp_level-voxel'.

Return type dict

See also:

nimare.correct.FWECorrector() The Corrector from which to call this method.

Examples

```
>>> meta = MKDADensity()
>>> result = meta.fit(dset)
>>> corrector = FWECorrector(method='permutation', voxel_thresh=0.01,
                             n_iters=5, n_cores=1)
>>> cresult = corrector.transform(result)
```

fit(*self*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

7.2.6 nimare.meta.cbma.model

Model-based coordinate-based meta-analysis estimators

Classes

BHICP()	Bayesian hierarchical cluster process model [Rbb0a73000b2f-1].
HPGRF()	Hierarchical Poisson/Gamma random field model [Rb6084ac49a63-1].
SBLFR()	Spatial Bayesian latent factor regression model [R270830ed9ee2-1].
SBR()	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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)

Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

7.3 nimare.results: Meta-analytic results

Base classes for datasets.

nimare.results	Base classes for datasets.
nimare.results.MetaResult(estimator, mask[, ...])	Base class for meta-analytic results.

7.3.1 nimare.results.MetaResult

class MetaResult(*estimator*, *mask*, *maps=None*)
Base class for meta-analytic results.

Methods

copy(self)	Returns copy of result object.
get_map(self, name[, return_type])	Get stored map as image or array.
save_maps(self[, output_dir, prefix, ...])	Save results to files.

copy(*self*)
Returns copy of result object.

get_map(*self*, *name*, *return_type='image'*)
Get stored map as image or array.

save_maps(*self*, *output_dir='.'*, *prefix=''*, *prefix_sep='_'*, *names=None*)
Save results to files.

Parameters

- **output_dir** (str, optional) – Output directory in which to save results. If the directory doesn't exist, it will be created. Default is current directory.
- **prefix** (str, optional) – Prefix to prepent to output file names. Default is none.
- **prefix_sep** (str, optional) – Separator to add between prefix and default file names. Default is _.

7.4 nimare.correct: Multiple comparisons correction

Multiple comparisons correction

nimare.correct	Multiple comparisons correction
nimare.correct.FWECorrector([method])	Perform family-wise error rate correction on a meta-analysis.
nimare.correct.FDRCorrector([alpha, method])	Perform false discovery rate correction on a meta-analysis.

7.4.1 nimare.correct.FWECorrector

class FWECorrector(*method='bonferroni'*, ***kwargs*)

Perform family-wise error rate correction on a meta-analysis.

Parameters

- **method** (str) – The FWE correction to use. Available internal methods are 'bonferroni'. Additional methods may be implemented within the provided Estimator.
- ****kwargs** – Keyword arguments to be used by the FWE correction implementation.

Notes

This corrector supports a small number of internal FDR correction methods, but can also use special methods implemented within individual Estimators. To determine what methods are available for the Estimator you're using, check the Estimator's documentation. Estimators have special methods following the naming convention correct_[correction-type]_[method] (e.g., ALE.correct_fwe_permutation).

Methods

transform(self, result)	Apply the multiple comparisons correction method to a MetaResult object.

transform(*self*, *result*)

Apply the multiple comparisons correction method to a MetaResult object.

Parameters **result** (*nimare.results.MetaResult*) – MetaResult generated by an Estimator to be corrected for multiple comparisons.

Returns **result** – MetaResult with new corrected maps added.

Return type *nimare.results.MetaResult*

7.4.2 nimare.correct.FDRCorrector

class FDRCorrector(*alpha=0.05*, *method='indep'*, ***kwargs*)
Perform false discovery rate correction on a meta-analysis.

Parameters

- **alpha** (float) – The FDR correction rate to use.
- **method** (str) – The FDR correction to use. Either 'indep' (for independent or positively correlated values) or 'negcorr' (for general or negatively correlated tests).

Notes

This corrector supports a small number of internal FDR correction methods, but can also use special methods implemented within individual Estimators. To determine what methods are available for the Estimator you're using, check the Estimator's documentation. Estimators have special methods following the naming convention correct_[correction-type]_[method] (e.g., MKDAChi2.correct_fdr_bh).

Methods

transform(self, result)	Apply the multiple comparisons correction method to a MetaResult object.

transform(*self*, *result*)
Apply the multiple comparisons correction method to a MetaResult object.

Parameters **result** (*nimare.results.MetaResult*) – MetaResult generated by an Estimator to be corrected for multiple comparisons.

Returns **result** – MetaResult with new corrected maps added.

Return type *nimare.results.MetaResult*

7.5 nimare.annotate: Automated annotation

Automated annotation tools

nimare.annotate	Automated annotation tools
nimare.annotate.ontology	Automated annotation tools for existing ontologies.
nimare.annotate.topic	Automated annotation with text-derived topic models.
nimare.annotate.vector	Automated annotation with text-derived vector models.
nimare.annotate.text	Text extraction tools.

7.5.1 nimare.annotate.ontology

Automated annotation tools for existing ontologies.

extract_cogpo()
Predict Cognitive Paradigm Ontology [1] labels with ATHENA classifiers [2].

Warning: This function is not yet implemented.

References

extract_cogat(*text_df*, *id_df=None*, *text_column='abstract'*)
Extract Cognitive Atlas [1] terms and count instances using regular expressions.

Parameters

- **text_df** ((D x 2) pandas.DataFrame) – Pandas dataframe with two columns: 'id' and the 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.
- **text_column** (str, optional) – Name of column in text_df that contains text. Default is 'abstract'.

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

transform(self, text[, convert_uk])	Replace terms in text with unique Cognitive Atlas identifiers.

transform(*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

7.5.2 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

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	

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 (>=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.

p_topic_g_voxel_
Probability of each topic (T) give a voxel (V)

Type (V x T) numpy.ndarray

p_voxel_g_topic_
Probability of each voxel (V) given a topic (T)

Type (V x T) numpy.ndarray

p_topic_g_word_
Probability of each topic (T) given a word (W)

Type (W x T) numpy.ndarray

p_word_g_topic_
Probability of each word (W) given a topic (T)

Type (W x T) numpy.ndarray

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

compute_log_likelihood(self[, model, ...])	Compute log-likelihood [1] of a model object given current model.
fit(self[, n_iters, loglikely_freq, verbose])	Run multiple iterations.
get_params(self[, deep])	Get parameters for this estimator.
get_probs(self)	Get conditional probability of selecting each voxel in the brain mask given each topic.
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.

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(topic|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(voxel|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(topic|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(word|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*, *text_column='abstract'*, *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_column) containing article text.
- **text_column** (str, optional) – Name of column in text_df that contains text. Default is 'abstract'.
- **n_topics** (int, optional) – Number of topics to generate. Default=50.
- **n_iters** (int, optional) – Number of iterations to run in training topic model. Default=1000.
- **alpha** (float or 'auto', optional) – The Dirichlet prior on the per-document topic distributions. Default: auto, which calculates 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).

commands_
List of MALLET commands called to fit model.

Type list of str

References

Methods

fit(self)	Fit LDA model to corpus.
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*)
Fit LDA model to corpus.

p_topic_g_doc_
Probability of each topic given a document
Type numpy.ndarray

p_word_g_topic_
Probability of each word given a topic
Type numpy.ndarray

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

7.5.3 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

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.

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

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.

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

7.5.4 nimare.annotate.text

Text extraction tools.

Functions

generate_cooccurrence(text_df[, ...])	Build co-occurrence matrix from documents.
generate_counts(text_df[, text_column, ...])	Generate tf-idf weights for unigrams/bigrams derived from textual data.

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*, *min_df=50*, *max_df=0.5*)
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

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*, *min_df=50*, *max_df=0.5*)

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

7.6 nimare.decode: Functional characterization analysis

Functional decoding tools

nimare.decode	Functional decoding tools
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.
nimare.decode.continuous	Methods for decoding unthresholded brain maps into text.
nimare.decode.encode	Methods for encoding text into brain maps.

7.6.1 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

brainmap_decode(coordinates, annotations, ids)	Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the BrainMap method [1].
gclda_decode_roi(model, roi[, topic_priors, ...])	Perform image-to-text decoding for discrete image inputs (e.g., regions of interest, significant clusters) according to the method described in [1].
neurosynth_decode(coordinates, annotations, ids)	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 (p_topic_g_voxel)
tau_t	Topic weight vector (topic_weights)
$p(w\|t)$	Probability of word type given topic (p_word_g_topic)

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 (`p_topic_g_voxel`)
tau_t	Topic weight vector (`topic_weights`)
$p(w\|t)$	Probability of word type given topic (`p_word_g_topic`)

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

7.6.2 nimare.decode.continuous

Methods for decoding unthresholded brain maps into text.

Functions

corr_decode(dataset, img[, features, ...])

param dataset A dataset with coordinates.

Continued on next page

Table 69 – continued from previous page

corr_dist_decode(dataset, img[, features, ...])	Builds feature-specific distributions of correlations with input image for image-based meta-analytic functional decoding.
gclda_decode_map(model, image[, ...])	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(*dataset*, *img*, *features=None*, *frequency_threshold=0.001*, *meta_estimator=None*, *target_image='specificity_z'*)

Parameters

- **dataset** (*nimare.dataset.Dataset*) – A dataset with coordinates.
- **img** (nibabel.Nifti1.Nifti1Image) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **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(*dataset*, *img*, *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

- **dataset** (*nimare.dataset.Dataset*) – A dataset with images.
- **img** (nibabel.Nifti1.Nifti1Image) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **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 (`p_topic_g_voxel`)
tau_t	Topic weight vector (`topic_weights`)
$p(w\|t)$	Probability of word type given topic (`p_word_g_topic`)
ω	1d array from input image (`input_values`)

1. Compute $p(t|v)$ (`p_topic_g_voxel`).
   - From `gclda.model.Model.get_spatial_probs()`
2. Squeeze input image to 1d array ω (`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(*dataset*, *img*, *features=None*, *frequency_threshold=0.001*, *meta_estimator=None*, *target_image='specificity_z'*)

Parameters

- **dataset** (*nimare.dataset.Dataset*) – A dataset with coordinates.
- **img** (nibabel.Nifti1.Nifti1Image) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **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(*dataset*, *img*, *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

- **dataset** (*nimare.dataset.Dataset*) – A dataset with images.
- **img** (nibabel.Nifti1.Nifti1Image) – Input image to decode. Must have same affine/dimensions as dataset mask.
- **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 (p_topic_g_voxel)
tau_t	Topic weight vector (topic_weights)
$p(w\|t)$	Probability of word type given topic (p_word_g_topic)
ω	1d array from input image (input_values)

1. Compute $p(t|v)$ (p_topic_g_voxel).
   - From gclda.model.Model.get_spatial_probs()
2. Squeeze input image to 1d array ω (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

7.6.3 nimare.decode.encode

Methods for encoding text into brain maps.

Functions

gclda_encode(model, text[, out_file, ...])	Perform text-to-image encoding according to the method described in [1].
text2brain()	Perform text-to-image encoding with the text2brain model [1].

nimare.decode.encode.gclda_encode

gclda_encode (*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 voxel given topic (`p_voxel_g_topic_`)
tau$_t$	Topic weight vector (`topic_weights`)
$p(w\|t)$	Probability of word type given topic (`p_word_g_topic`)
ω	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

`gclda_encode` (*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 voxel given topic (p_voxel_g_topic_)
tau_t	Topic weight vector (topic_weights)
$p(w\|t)$	Probability of word type given topic (p_word_g_topic)
ω	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

text2brain()
Perform text-to-image encoding with the text2brain model [1].

> **Warning:** This method is not yet implemented.

References

7.7 nimare.parcellate: Meta-analytic parcellation

Meta-analytic parcellation tools

nimare.parcellate	Meta-analytic parcellation tools
nimare.parcellate.cbp	Coactivation-based parcellation
nimare.parcellate.mamp	Meta-analytic activation modeling-based parcellation (MAMP).
nimare.parcellate.mapbot	Meta-analytic parcellation based on text (MAPBOT).

7.7.1 nimare.parcellate.cbp

Coactivation-based parcellation

Classes

CoordCBP(dataset, ids)	Coordinate-based coactivation-based parcellation [R759c29a4323f-1].
ImCBP(dataset, ids)	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 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 not yet implemented.

References

Methods

fit(self, target_mask[, method, r, n_exps, ...])	Run CBP parcellation.
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*, *method='min_distance'*, *r=5*, *n_exps=50*, *n_parcels=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_parcels** (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

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_parcels])	**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_parcels=2*)

Parameters

- **target_mask** (*img_like*) – Image with binary mask for region of interest to be parcellated.
- **n_parcels** (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 not yet implemented.

References

Methods

fit(self, target_mask[, method, r, n_exps, ...])	Run CBP parcellation.
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*, *method='min_distance'*, *r=5*, *n_exps=50*, *n_parcels=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_parcels** (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 ImCBP (*dataset*, *ids*)
Image-based coactivation-based parcellation

> **Warning:** This method is not yet implemented.

Methods

fit(self, target_mask[, n_parcels])	**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_parcels=2*)

Parameters

- **target_mask** (*img_like*) – Image with binary mask for region of interest to be parcellated.
- **n_parcels** (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

7.7.2 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

fit(self, target_mask[, n_parcels, ...])	Run MAMP parcellation.
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_parcels=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_parcels** (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 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

fit(self, target_mask[, n_parcels, ...])	Run MAMP parcellation.
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_parcels=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_parcels** (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

7.7.3 nimare.parcellate.mapbot

Meta-analytic parcellation based on text (MAPBOT).

Classes

MAPBOT(tfidf_df, coordinates_df, mask)	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_voxels X n_terms frequency matrix F.
3. Compute n_voxels X n_voxels value matrix V: - D = (F.T * F) * ones(F) - V = F * D^-.5
4. Perform non-negative matrix factorization on value matrix.

Warning: This method is not yet implemented.

References

Methods

fit(self, target_mask[, method, r, n_exps, ...])	Run MAPBOT parcellation.
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*, *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) * ones(F) - V = F * D^-.5
4. Perform non-negative matrix factorization on value matrix.

Warning: This method is not yet implemented.

References

Methods

fit(self, target_mask[, method, r, n_exps, ...])	Run MAPBOT parcellation.
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*, *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

7.8 nimare.io: Input/Output

Input/Output operations.

nimare.io	Input/Output operations.
nimare.io.convert_neurosynth_to_dict(text_file)	Convert Neurosynth database files to a dictionary.
nimare.io.convert_neurosynth_to_json(...[, ...])	Convert Neurosynth dataset text file to a NiMARE json file.
nimare.io.convert_neurosynth_to_dataset(...)	Convert Neurosynth database files into dictionary and create NiMARE Dataset with dictionary.
nimare.io.convert_sleuth_to_dict(text_file)	Convert Sleuth text file to a dictionary.
nimare.io.convert_sleuth_to_json(text_file, ...)	Convert Sleuth output text file into json.
nimare.io.convert_sleuth_to_dataset(text_file)	Convert Sleuth output text file into dictionary and create NiMARE Dataset with dictionary.

7.8.1 nimare.io.convert_neurosynth_to_dict

convert_neurosynth_to_dict(*text_file*, *annotations_file=None*)

Convert Neurosynth database files to a dictionary.

Parameters

- **text_file** (str) – Text file with Neurosynth's coordinates. Normally named "database.txt".
- **annotations_file** (str or None, optional) – Optional file with Neurosynth's annotations. Normally named "features.txt". Default is None.

Returns **dict_** – NiMARE-organized dictionary containing experiment information from text files.

Return type dict

7.8.2 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.

7.8.3 nimare.io.convert_neurosynth_to_dataset

convert_neurosynth_to_dataset(*text_file*, *annotations_file=None*, *target='mni152_2mm'*)
Convert Neurosynth database files into dictionary and create NiMARE Dataset with dictionary.

Parameters

- **text_file** (str) – Text file with Neurosynth's coordinates. Normally named "database.txt".
- **target** (*{'mni152_2mm', 'ale_2mm'}, optional*) – Target template space for coordinates. Default is 'mni152_2mm'.
- **annotations_file** (str or None, optional) – Optional file with Neurosynth's annotations. Normally named "features.txt". Default is None.

Returns Dataset object containing experiment information from text_file.

Return type *nimare.dataset.Dataset*

7.8.4 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 NiMARE-organized dictionary containing experiment information from text file.

Return type dict

7.8.5 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.

7.8.6 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*

7.9 nimare.extract: Dataset and model fetching

Dataset and trained model downloading functions

nimare.extract	Dataset and trained model downloading functions
nimare.extract.download_nidm_pain([...])	Download NIDM Results for 21 pain studies from NeuroVault for tests.
nimare.extract.download_mallet([data_dir, ...])	Download the MALLET toolbox for LDA topic modeling.
nimare.extract.download_cognitive_atlas([...])	Download Cognitive Atlas ontology and combine Concepts, Tasks, and Disorders to create ID and relationship DataFrames.
nimare.extract.download_abstracts(dataset, email)	Download the abstracts for a list of PubMed IDs.
nimare.extract.download_peaks2maps_model([...])	Download the trained Peaks2Maps model from OHBM 2018.

7.9.1 nimare.extract.download_nidm_pain

download_nidm_pain(*data_dir=None*, *overwrite=False*, *verbose=1*)

Download NIDM Results for 21 pain studies from NeuroVault for tests.

Parameters

- **data_dir** (str, optional) – Location in which to place the studies. Default is None, which uses the package's default path for downloaded data.
- **overwrite** (bool, optional) – Whether to overwrite existing files or not. Default is False.
- **verbose** (int, optional) – Default is 1.

Returns **data_dir** – Updated data directory pointing to dataset files.

Return type str

7.9.2 nimare.extract.download_mallet

download_mallet(*data_dir=None*, *overwrite=False*, *verbose=1*)

Download the MALLET toolbox for LDA topic modeling.

Parameters

- **data_dir** (str, optional) – Location in which to place MALLET. Default is None, which uses the package's default path for downloaded data.
- **overwrite** (bool, optional) – Whether to overwrite existing files or not. Default is False.
- **verbose** (int, optional) – Default is 1.

Returns **data_dir** – Updated data directory pointing to MALLET files.

Return type str

7.9.3 nimare.extract.download_cognitive_atlas

download_cognitive_atlas(*data_dir=None*, *overwrite=False*, *verbose=1*)

Download Cognitive Atlas ontology and combine Concepts, Tasks, and Disorders to create ID and relationship DataFrames.

Parameters

- **data_dir** (str, optional) – Location in which to place Cognitive Atlas files. Default is None, which uses the package's default path for downloaded data.
- **overwrite** (bool, optional) – Whether to overwrite existing files or not. Default is False.
- **verbose** (int, optional) – Default is 1.

Returns **out_dict** – Dictionary with two keys: 'ids' and 'relationships'. Each points to a csv file. The 'ids' file contains CogAt identifiers, canonical names, and aliases, sorted by alias length (number of characters). The 'relationships' file contains associations between CogAt items, with three columns: input, output, and rel_type (relationship type).

Return type dict

7.9.4 nimare.extract.download_abstracts

download_abstracts(*dataset*, *email*)

Download the abstracts for a list of PubMed IDs. Uses the BioPython package.

Parameters

- **dataset** (*nimare.dataset.Dataset*) – A Dataset object where IDs are in the form PMID-EXPID
- **email** (str) – Email address to use to call the PubMed API

Returns **dataset**

Return type *nimare.dataset.Dataset*

7.9.5 nimare.extract.download_peaks2maps_model

download_peaks2maps_model(*data_dir=None*, *overwrite=False*, *verbose=1*)

Download the trained Peaks2Maps model from OHBM 2018.

7.10 nimare.stats: Statistical functions

Various statistical helper functions

nimare.stats	Various statistical helper functions
nimare.stats.one_way(data, n)	One-way chi-square test of independence.
nimare.stats.two_way(cells)	Two-way chi-square test of independence.
nimare.stats.pearson(x, y)	Correlates row vector x with each row vector in 2D array y.
nimare.stats.null_to_p(test_value, null_array)	Return two-sided p-value for test value against null array.

Continued on next page

Table 85 – continued from previous page

nimare.stats.p_to_z(p[, tail])	Convert p-values to z-values.
nimare.stats.t_to_z(t_values, dof)	From Vanessa Sochat's TtoZ package.
nimare.stats.fdr(p[, q])	Determine FDR threshold given a p value array and desired false discovery rate q.

7.10.1 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

7.10.2 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.

7.10.3 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*) –

7.10.4 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.

7.10.5 nimare.stats.p_to_z

p_to_z(*p*, *tail='two'*)

Convert p-values to z-values.

7.10.6 nimare.stats.t_to_z

t_to_z(*t_values*, *dof*)

From Vanessa Sochat's TtoZ package.

7.10.7 nimare.stats.fdr

fdr(*p*, *q=0.05*)
Determine FDR threshold given a p value array and desired false discovery rate q.

7.11 nimare.utils: Utility functions and submodules

Utilities

nimare.utils	Utilities
nimare.utils.get_template([space, mask])	Load template file.
nimare.utils.listify(obj)	Wraps all non-list or tuple objects in a list; provides a simple way to accept flexible arguments.
nimare.utils.round2(ndarray)	Numpy rounds X.5 values to nearest even integer.
nimare.utils.vox2mm(ijk, affine)	Convert matrix subscripts to coordinates.
nimare.utils.mm2vox(xyz, affine)	Convert coordinates to matrix subscripts.
nimare.utils.tal2mni(coords)	Python version of BrainMap's tal2icbm_other.m.
nimare.utils.mni2tal(coords)	Python version of BrainMap's icbm_other2tal.m.
nimare.utils.get_resource_path()	Returns the path to general resources, terminated with separator.

7.11.1 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

7.11.2 nimare.utils.listify

listify(*obj*)
Wraps all non-list or tuple objects in a list; provides a simple way to accept flexible arguments.

7.11.3 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.

7.11.4 nimare.utils.vox2mm

vox2mm(*ijk*, *affine*)
Convert matrix subscripts to coordinates. From here: http://blog.chrisgorgolewski.org/2014/12/how-to-convert-between-voxel-and-mm.html

7.11.5 nimare.utils.mm2vox

mm2vox(*xyz*, *affine*)
Convert coordinates to matrix subscripts. From here: http://blog.chrisgorgolewski.org/2014/12/how-to-convert-between-voxel-and-mm.html

7.11.6 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 FORMAT outpoints = 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

7.11.7 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

7.11.8 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.

7.12 nimare.workflows: Common workflows

Common meta-analytic workflows

nimare.workflows	Common meta-analytic workflows
nimare.workflows.ale	Workflow for running an ALE meta-analysis from a Sleuth text file.
nimare.workflows.conperm	Workflow for running a contrast permutation meta-analysis on a set of images.

Continued on next page

Table 87 – continued from previous page

nimare.workflows.macm	Perform MACM with ALE algorithm.
nimare.workflows.peaks2maps	Workflow for contrast permutation meta-analysis on images constructed from coordinates using the Peaks2Maps kernel.
nimare.workflows.scale	Workflow for running a SCALE meta-analysis from a Sleuth text file.

7.12.1 nimare.workflows.ale

Workflow for running an ALE meta-analysis from a Sleuth text file.

Functions

ale_sleuth_workflow(sleuth_file[, ...])	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.

7.12.2 nimare.workflows.conperm

Workflow for running a contrast permutation meta-analysis on a set of images.

Functions

conperm_workflow(contrast_images[, ...])	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.

7.12.3 nimare.workflows.macm

Perform MACM with ALE algorithm.

Functions

macm_workflow(dataset_file, mask_file[, ...])	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.

7.12.4 nimare.workflows.peaks2maps

Workflow for contrast permutation meta-analysis on images constructed from coordinates using the Peaks2Maps kernel.

Functions

peaks2maps_workflow(sleuth_file[, ...])	

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*)

7.12.5 nimare.workflows.scale

Workflow for running a SCALE meta-analysis from a Sleuth text file.

Functions

scale_workflow(dataset_file, baseline[, ...])	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.

7.13 nimare.base: Base classes

Base classes for datasets.

nimare.base	Base classes for datasets.
nimare.base.NiMAREBase()	Base class for NiMARE.
nimare.base.Estimator()	Estimators take in Datasets and return MetaResults
nimare.base.Transformer()	Transformers take in Datasets and return Datasets

7.13.1 nimare.base.NiMAREBase

class NiMAREBase
Base class for NiMARE.

Methods

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.

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

7.13.2 nimare.base.Estimator

class Estimator
Estimators take in Datasets and return MetaResults

Methods

fit(self, dataset)	Fit Estimator to Dataset.
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*, *dataset*)
Fit Estimator to Dataset.

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

Returns Results of Estimator fitting.

Return type *nimare.results.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

7.13.3 nimare.base.Transformer

class Transformer
Transformers take in Datasets and return Datasets

Initialize with hyperparameters.

Methods

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.
transform(self, dataset)	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.

CHAPTER 8

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

[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

[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

[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

[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

[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

Python Module Index

n

Index

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W