An Open Publishing Platform of the Organization for Human Brain Mapping
APERTURE NEURO

JUPYTER BOOK

NiMARE: Neuroimaging Meta-Analysis Research Environment

Taylor Salo, Tal Yarkoni, Thomas E. Nichols, Jean-Baptiste Poline, Murat Bilgel, Katherine L. Bottenhorn, Simon B. Eickhoff, Dorota Jarecka, James D. Kent, Adam Kimbler, Dylan M. Nielson, Kendra M. Oudyk, Julio A. Peraza, Alexandre Pérez, Puck C. Reeders, Julio A. Yanes, Angela R. Laird

ABSTRACT

Corresponding author: Dr. Taylor Salo, Florida International University, FL, USA, email: tsalo006@fiu.edu
Date Received: February 28, 2022
Date Accepted: September 04, 2022

CONTENTS

This is an open access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits authors to copy and redistribute the material in any medium or format, remix, transform and build upon material, for any purpose, even commercially.

OPEN ACCESS

We present NiMARE (Neuroimaging Meta-Analysis Research Environment; RRID:SCR_0173981), a Python library for neuroimaging meta-analyses and meta-analysis-related analyses. NiMARE is an open source, collaboratively-developed package that implements a range of meta- analytic algorithms, including coordinate- and image-based meta-analyses, automated annotation, functional decoding, and meta-analytic coactivation modeling. By consolidating meta-analytic methods under a common library and syntax, NiMARE makes it straightforward for users to employ the appropriate approach for a given analysis. In this paper, we describe NiMARE's architecture and the methods implemented in the library. Additionally, we provide example code and results for each of the available tools in the library.

INTRODUCTION

We introduce NiMARE (Neuroimaging Meta-Analysis Research Environment), a Python package for analyzing meta-analytic neuroimaging data. NiMARE is a new library developed as a component in a burgeoning open-source meta-analytic ecosystem for neuroimaging data, which currently includes Neurosynth, NeuroVault, NeuroQuery, and PyMARE.

While several libraries already exist for neuroimaging meta-analysis, these libraries are generally algorithm-specific, and are provided in a range of very different user interfaces, languages, and licenses. This variability may prevent meta-analysts from using the most appropriate algorithm for a given analysis. Further, having multiple meta-analysis algorithms available in one library facilitates

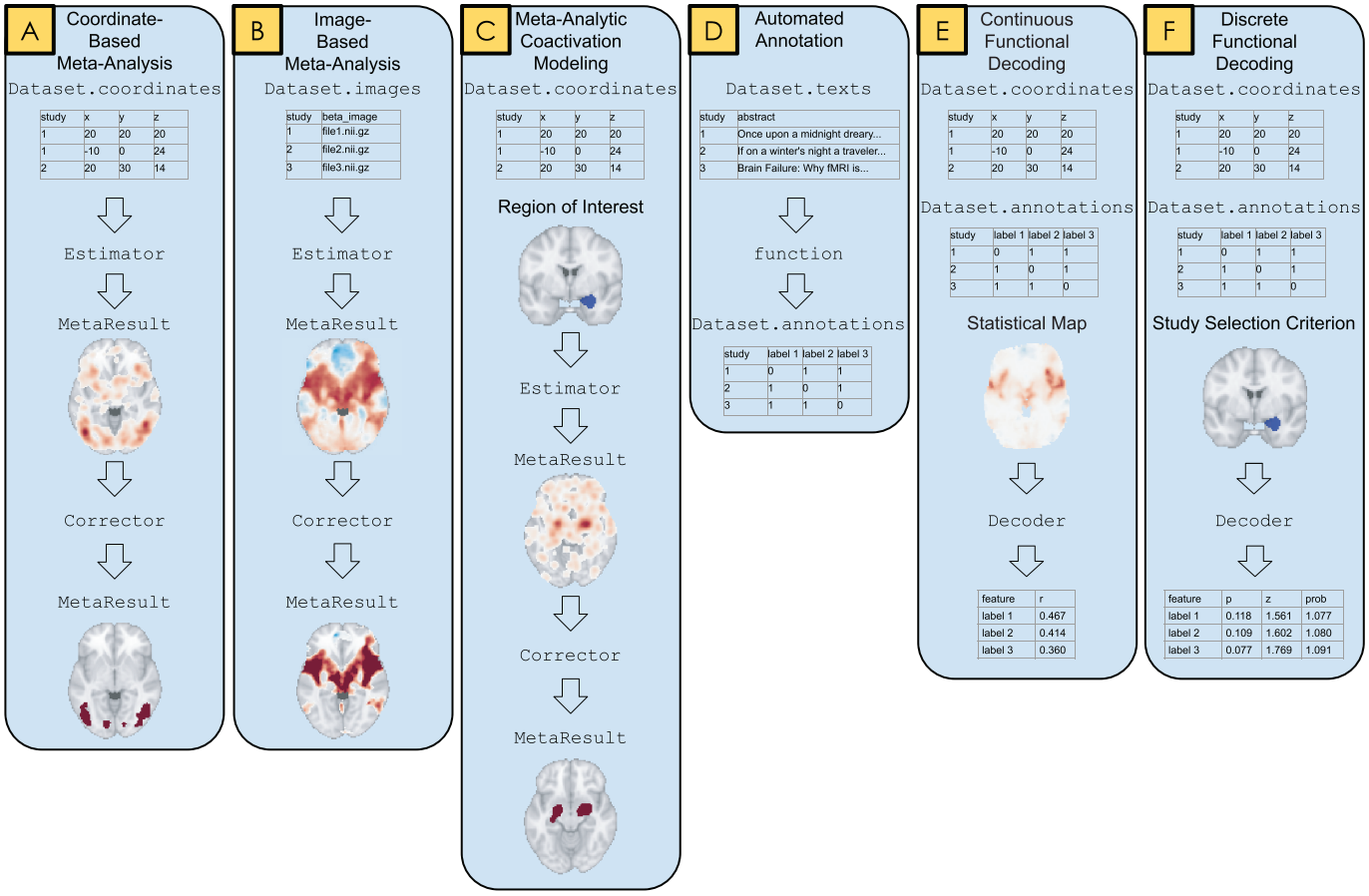

Fig. 1. A graphical representation of tools and methods implemented in NiMARE. This diagram outlines six of the most common use-cases for NiMARE. **(A)** Coordinate-Based Meta-Analysis (CBMA) is performed by creating a NiMARE Dataset with coordinate information stored in the Dataset.coordinates attribute, which is then used in a CBMA Estimator. This produces a MetaResult object with statistical maps, which can then be used in a Corrector object for multiple comparisons correction. Once the Corrector has been fitted, it will produce a corrected version of the MetaResult object, containing updated statistical maps. **(B)** Image-Based Meta-Analysis (IBMA) operates similarly to CBMA, except that IBMA Estimators use statistical maps stored in the Dataset.images attribute. **(C)** Meta-Analytic Coactivation Modeling (MACM) uses a region of interest to select coordinate-based studies within a Dataset, after which the standard CBMA workflow is performed. **(D)** Automated Annotation infers labels from textual (and sometimes other) data associated with the Dataset, as stored in the Dataset.texts attribute. The annotation functions produce labels which may be integrated into the Dataset as the Dataset.annotations attribute. **(E)** Functional decoding of continuous statistical maps operates similarly to discrete decoding, in that the input Dataset must have both coordinates and annotations attributes. The Dataset, along with an unthresholded statistical map to decode, is provided to the Decoder object, which then outputs measures of similarity or associativeness with each label. **(F)** Functional decoding of discrete inputs applies a selection criterion to a Dataset with both coordinates and annotations attributes, using a Decoder object. The decoding algorithm will output measures of similarity or associativeness with each label in the annotations.

 CC By 4.0: © Taylor Salo et al.

direct comparisons of methods. With NiMARE, we consolidate meta-analytic algorithms from a range of libraries and publications, and provide a common Python syntax and well documented application program interfaces. Additionally, NiMARE is a collaboratively-developed open source package, enabling researchers to contribute new methods not included in the current version.

In this paper, we describe NiMARE's aims, architecture and the functionality it supports—including tools for database extraction, automated annotation, meta-analysis, meta-analytic coactivation modeling, and functional decoding. The text is accompanied by extensive code samples and results (also available online in the form of Python scripts; https://github.com/NBCLab/nimare-paper with additional documentation in https://github.com/neurodatascience/meta_analysis_notebook), ensuring that users can follow along interactively.

NIMARE OVERVIEW

NiMARE is designed to be modular and object-oriented, with an interface that mimics popular Python libraries, including scikit-learn and nilearn. This standardized interface allows users to employ a wide range of meta-analytic algorithms without having to familiarize themselves with the idiosyncrasies of algorithm-specific tools. This lets users use whatever method is most appropriate for a given research question with minimal mental overhead from switching methods. Additionally, NiMARE emphasizes citability, with references in the documentation and citable boilerplate text that can be copied directly into manuscripts, in order to ensure that the original algorithm developers are appropriately recognized.

NiMARE works with Python versions 3.6 and higher, and can easily be installed with pip. Its source code is housed and version controlled in a GitHub repository at https://github.com/neurostuff/NiMARE.

NiMARE is under continued active development, and we anticipate that the user-facing API (application programming interface) may change over time. Our emphasis in this paper is thus primarily on reviewing the functionality implemented in the package and illustrating the general interface, and not on providing a detailed and static user guide that will be found within the package documentation.

Tools in NiMARE are organized into several modules, including **nimare.meta, nimare.correct, nimare.annotate, nimare.decode,** and **nimare.workflows**. In addition to these primary modules, there are several secondary modules for data wrangling and internal helper functions, including **nimare.io, nimare.dataset, nimare.extract, nimare.stats, nimare.utils,** and **nimare.base.** These modules are summarized in Application Programming Interface, as well as in Table 1.

Application programming interface

One of the principal goals of NiMARE is to implement a range of methods with a set of shared interfaces, to enable users to employ the most appropriate algorithm for a given question without introducing a steep learning curve. This approach is modeled on the widely-used scikit-learn package,[2,3] which implements a large number of machine learning algorithms - all with simple, consistent interfaces. Regardless of the algorithm employed, data should be in the same format and the same class methods should be called to fit and/or generate predictions from the model.

To this end, we have adopted an object-oriented approach to NiMARE's core API that organizes tools based on the type of inputs and outputs they operate over. The key data structure is the **Dataset** class, which stores a range of neuroimaging data amenable to various forms of meta-analysis. There are two main types of tools that operate on a Dataset class. **Transformer** classes, as their name suggests, perform some transformation on a Dataset- i.e., they take a Dataset instance as input, and return a modified version of that Dataset instance as output (for example, with newly generated maps stored within the object). **Estimator** classes apply a meta-analytic algorithm to a Dataset and return a set of statistical images stored in a MetaResult container class. The key methods supported by each of these base classes, as well as the main arguments to those methods, are consistent throughout the hierarchy (e.g., all Transformer classes must implement a transform() method), minimizing the learning curve and ensuring a high degree of predictability for users.

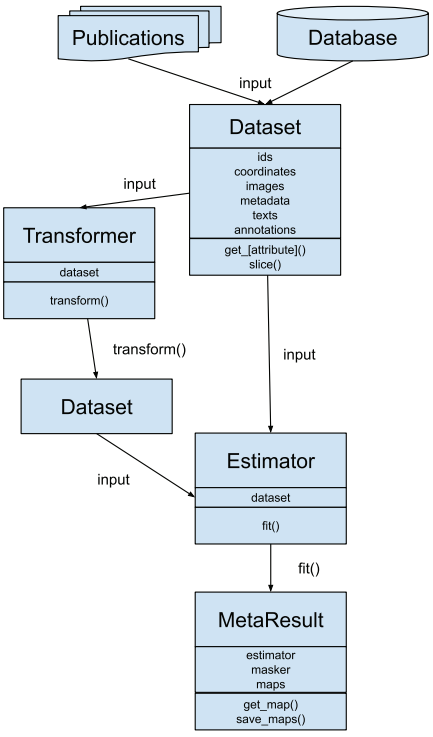

Fig. 2. A schematic figure of Datasets, Estimators, Transformers, and MetaResults in NiMARE.

 CC By 4.0: © Taylor Salo et al.

Package organization

At present, the package is organized into 14 distinct modules. **nimare.dataset** defines the Dataset class. **nimare.meta** includes Estimators for coordinate- and image-based meta-analysis methods. **nimare.results** defines the MetaResult class, which stores statistical maps produced by meta-analyses. **nimare.correct** implements Corrector classes for family-wise error (FWE) and false discovery rate (FDR) multiple comparisons correction. **nimare.annotate** implements a range of automated annotation methods, including latent Dirichlet allocation (LDA) and generalized correspondence latent Dirichlet allocation (GCLDA). **nimare.decode** implements a number of meta-analytic functional decoding and encoding algorithms. **nimare.io** provides functions for converting alternative meta-analytic dataset structure, such as Sleuth text files or Neurosynth Datasets, to NiMARE format. **nimare.transforms** implements a range of spatial and data type transformations, including a function to generate new images in the Dataset from existing image types. **nimare.extract** provides methods for fetching Datasets and models across the internet. **nimare.generate** includes functions for generating data for internal testing and validation. **nimare.base** defines a number of base classes used throughout the rest of the package. Finally, **nimare.stats** and **nimare.utils** are modules for statistical and generic utility functions, respectively. These modules are summarized in Table 1.

Dependencies

NiMARE depends on the standard SciPy stack, as well as a small number of widely-used packages. Dependencies from the SciPy stack include scipy,[4] numpy,[5,6] pandas,[7] and scikit-learn.[2,3] Additional requirements include fuzzywuzzy, nibabel,[8] nilearn[9], statsmodels,[10] and tqdm.[11]

DOWNLOAD THE DATA

```
# First, import the necessary modules and functions
import os

from repo2data.repo2data import Repo2Data

# Install the data if running locally, or points to cached data if running on neurolibre
DATA_REQ_FILE = os.path.abspath("../binder/data_requirement.json")
repo2data = Repo2Data(DATA_REQ_FILE)
data_path = repo2data.install()
data_path = os.path.join(data_path[0], "data")
print(f"Data are located at {data_path}")
```

Table 1. Summaries of modules in NiMARE.

Module	Description
dataset	This module stores the Dataset class, which contains NiMARE Datasets.
meta	This module contains Estimators for image- and coordinate-based meta-analysis algorithms, as well as KernelTransformers, which are used in conjunction with coordinate-based methods.
results	This module stores the MetaResult class, which in turn is used to manage statistical maps produced by meta-analytic algorithms.
correct	This module contains classes for multiple comparisons correction, including FWECorrector (family-wise error rate correction) and FDRCorrector (FDR correction).
annotate	This module includes a range of tools for automated annotation of studies. Methods in this module include: topic models, such as LDA and GCLDA; ontology-based annotation, such as Cognitive Atlas term extract from text; and general text-based feature extraction, such as count or tf-idf extraction from text.
decode	This module includes a number of methods for functional characterization analysis, also known as functional decoding. Methods in this module are divided into three groups: discrete, for decoding regions of interest or subsets of the Dataset; continuous, for decoding unthresholded statistical maps; and encoding, for simulating statistical maps from labels.
io	This module contains functions for converting common file types, such as Neurosynth- or Sleuth-format files, into NiMARE- compatible formats, such as Dataset objects.
transforms	This module contains classes and functions for converting between common data types. Two important classes in this module are the ImageTransformer, which uses available images and metadata to produce new images in a Dataset, and the ImagesToCoordinates, which extracts peak coordinates from images in the Dataset, so that image-based studies can be used for coordinate-based meta-analyses.
extract	This module contains functions for downloading external resources, such as the Neurosynth Dataset and the Cognitive Atlas ontology.
stats	This module contains miscellaneous statistical methods used throughout the rest of the library.
generate	This module contains functions for generating useful data for internal testing and validation.
utils	This module contains miscellaneous utility functions used throughout the rest of the library.
workflows	This module contains a number of common workflows that can be run from the command line, such as an activation likelihood estimation (ALE) meta-analysis or a contrast-permutation image-based meta-analysis. All of the workflow functions additionally generate boilerplate text that can be included in manuscript methods sections.
base	This module defines a number of base classes used throughout the rest of the library.

 CC By 4.0: © Taylor Salo et al.

We will also create a directory in which to save files that are generated within the book.

```
os.makedirs("../outputs/", exist_ok=True)
print(f"Files generated by the book will be saved to {os.path.
abspath('../outputs/')}")
```

```
Files generated by the book will be saved to /Users/taylor/
Documents/nbc/nimarepaper/ outputs
```

EXTERNAL META-ANALYTIC RESOURCES

Large-scale meta-analytic databases have made systematic meta-analyses of the neuroimaging literature possible. These databases combine results from neuroimaging studies, whether represented as coordinates of peak activations or unthresholded statistical images, with important study metadata, such as information about the samples acquired, stimuli used, analyses performed, and mental constructs putatively manipulated. The two most popular coordinate-based meta-analytic databases are BrainMap and Neurosynth, while the most popular image-based database is NeuroVault.

The studies archived in these databases may be either manually or automatically annotated—often with reference to a formal ontology or controlled vocabulary. Ontologies for cognitive neuroscience define what mental states or processes are postulated to be manipulated or measured in experiments, and may also include details of said experiments (e.g., the cognitive tasks employed), relationships between concepts (e.g., verbal working memory is a kind of working memory), and various other metadata that can be standardized and represented in a machine-readable form.[12–14] Some of these ontologies are very well-defined, such as expert-generated taxonomies designed specifically to describe only certain aspects of experiments and the relationships between elements within the taxonomy, while others are more loosely defined, in some cases simply building a vocabulary based on which terms are commonly used in cognitive neuroscience articles.

BrainMap

BrainMap [15–17] relies on expert annotators to label individual comparisons within studies according to its internally developed ontology, the BrainMap Taxonomy.[15] While this approach is likely to be less noisy than an automated annotation method using article text or imaging results to predict content, it is also subject to a number of limitations. First, there are simply not enough annotators to keep up with the ever-expanding literature. Second, any development of the underlying ontology has the potential to leave the database outdated. For example, if a new label is added to the BrainMap Taxonomy, then each study in the full BrainMap database needs to be evaluated for that label before that label can be properly integrated into the database. Finally, a manually annotated database like BrainMap will be biased by which subdomains within the literature are annotated. While outside contributors can add and annotate studies to the database, the main source of annotations has been researchers associated with the BrainMap project.

While BrainMap is a semi-closed resource (i.e., a collaboration agreement is required to access the full database), registered users may search the database using the Sleuth search tool, in order to collect samples for meta-analyses. Sleuth can export these study collections as text files with coordinates. NiMARE provides a function to import data from Sleuth text files into the NiMARE Dataset format.

The function **convert_sleuth_to_dataset()** can be used to convert text files exported from Sleuth into NiMARE Datasets. Here, we convert two files from a previous publication by NiMARE contributors[18] into two separate Datasets.

```
from nimare import io

sleuth_dset1 = io.convert_sleuth_to_dataset(
    os.path.join(data_path, "contrast-CannabisMinusControl_
    space-talairach_sleuth.txt")
)
sleuth_dset2 = io.convert_sleuth_to_dataset(
    os.path.join(data_path, "contrast-ControlMinusCannabis_
    space-talairach_sleuth.txt")
)
print(sleuth_dset1)
print(sleuth_dset2)

# Save the Datasets to files for future use
sleuth_dset1.save(os.path.join(out_dir, "sleuth_dset1.pkl.gz"))
sleuth_dset2.save(os.path.join(out_dir, "sleuth_dset2.pkl.gz"))
```

```
Dataset(41 experiments, space='ale_2mm') Dataset
(41 experiments, space='ale_2mm')
```

Neurosynth

Neurosynth[19] uses a combination of web scraping and text mining to automatically harvest neuroimaging studies from the literature and to annotate them based on term frequency within article abstracts. As a consequence of its relatively crude automated approach, Neurosynth has its own set of limitations. First, Neurosynth is unable to delineate individual comparisons within studies, and consequently uses the entire paper as its unit of measurement, unlike BrainMap. This risks conflating directly contrasted comparisons (e.g., A>B and B>A), as well as comparisons which have no relation to one another. Second, coordinate extraction and annotation are noisy. Third, annotations automatically performed by Neurosynth are also subject to error, although the reasons behind this are more nuanced and will be discussed later in this paper. Given Neurosynth's limitations, we recommend that it be used for casual, exploratory meta-analyses rather

 CC By 4.0: © Taylor Salo et al.

than for publication-quality analyses. Nevertheless, while individual meta-analyses should not be published from Neurosynth, many derivative analyses have been performed and published (e.g.[20–23]). As evidence of its utility, Neurosynth has been used to define *a priori* regions of interest (e.g.[24–26]) or perform meta-analytic functional decoding (e.g.[27–29],) in many first-order (rather than meta-analytic) fMRI studies.

Here, we show code that would download the Neurosynth database from where it is stored (https://github.com/neurosynth/neurosynth-data) and convert it to a NiMARE Dataset using **fetch_neurosynth()**, for the first step, and **convert_neurosynth_to_dataset()**, for the second.

```
from nimare import extract

# Download the desired version of Neurosynth
from GitHub.
files = extract.fetch_neurosynth(
    data_dir=data_path,
    version="7",
    source="abstract",
    vocab="terms",
    overwrite=False,
)
pprint(files) neurosynth_db = files[0]
```

```
INFO:nimare.extract.utils:Dataset found in ./../data/nimare-paper/
data/neurosynth
```

```
INFO:nimare.extract.extract:Searching for any feature files
matching the following criteria: [('source-abstract', 'vocab-terms',
'data-neurosynth', 'version-7')]
```

```
Downloading data-neurosynth_version-7_coordinates.tsv.gz
```

```
File exists and overwrite is False. Skipping.
Downloading data-neurosynth_version-7_metadata.tsv.gz
```

```
File exists and overwrite is False. Skipping.
Downloading data-neurosynth_version-7_vocab-terms_source-
abstract_type-tfidf_features.npz
```

```
File exists and overwrite is False. Skipping.
Downloading data-neurosynth_version-7_vocab-terms_
vocabulary.txt
```

```
File exists and overwrite is False. Skipping.
[{'coordinates': '/Users/taylor/Documents/nbc/nimare-paper/
data/nimare- paper/data/neurosynth/data-neurosynth_version-7_
coordinates.tsv.gz',
  'features': [{'features': '/Users/taylor/Documents/nbc/nimare-
paper/data/nimare- paper/data/neurosynth/data-neurosynth_
version-7_vocab-terms_source-abstract_type- tfidf_features.npz',
  'vocabulary': '/Users/taylor/Documents/nbc/nimare-paper/
data/nimare- paper/data/neurosynth/data-neurosynth_version-7_
vocab-terms_vocabulary.txt'}],
  'metadata': '/Users/taylor/Documents/nbc/nimare-paper/data/
nimare- paper/data/neurosynth/data-neurosynth_version-7_
metadata.tsv.gz'}]
```

Converting the large Neurosynth and NeuroQuery Datasets to NiMARE **Dataset** objects can be a very memory-intensive process. For the sake of this book, we show how to perform the conversions below, but actually load and use pre-converted Datasets.

```
# Convert the files to a Dataset.
# This may take a while (~10 minutes)
neurosynth_dset = io.convert_neurosynth_to_dataset(
      coordinates_file=neurosynth_db["coordinates"],
      metadata_file=neurosynth_db["metadata"],
      annotations_files=neurosynth_db["features"],
)
print(neurosynth_dset)

# Save the Dataset for later use.
neurosynth_dset.save(os.path.join(out_dir, "neurosynth_dataset.
pkl.gz"))
```

Here, we load a pre-generated version of the Neurosynth Dataset.

```
from nimare import dataset

neurosynth_dset = dataset.Dataset.load(os.path.join(data_path,
"neurosynth_dataset.pkl.gz"))
print(neurosynth_dset)
```

```
Dataset(14371 experiments, space='mni152_2mm')
```

Many of the methods in NiMARE can be very time-consuming or memory-intensive. Therefore, for the sake of ensuring that the analyses in this article may be reproduced by as many people as possible, we will use a reduced version of the Neurosynth Dataset, only containing the first 500 studies, for those methods which may not run easily on the full database.

```
neurosynth_dset_first_500 = neurosynth_dset.slice(neurosynth_
dset.ids[:500]) print(neurosynth_dset)

# Save this Dataset for later use.
neurosynth_dset_first_500.save(os.path.join(out_dir,
"neurosynth_dataset_first500.pkl.gz"))
```

```
Dataset(14371 experiments, space='mni152_2mm')
```

 CC By 4.0: © Taylor Salo et al.

In addition to a large corpus of coordinates, Neurosynth provides term frequencies derived from article abstracts that can be used as annotations.

One additional benefit to Neurosynth is that it has made available the coordinates for a large number of studies for which the study abstracts are also readily available. This has made the Neurosynth database a common resource upon which to build other automated ontologies. Data-driven ontologies which have been developed using the Neurosynth database include the GCLDA30 topic model and Deep Boltzmann machines.[31]

NeuroQuery

A related resource is **NeuroQuery**.[32] NeuroQuery is an online service for large-scale predictive meta-analysis. Unlike Neurosynth, which performs statistical inference and produces statistical maps, NeuroQuery is a supervised learning model and produces a prediction of the brain areas most likely to contain activations. These maps predict locations where studies investigating a given area (determined by the text prompt) are likely to produce activations, but they cannot be used in the same manner as statistical maps from a standard coordinate-based meta-analysis. In addition to this predictive meta-analytic tool, NeuroQuery also provides a new database of coordinates, text annotations, and metadata via an automated extraction approach that improves on Neurosynth's original methods.

While NiMARE does not currently include an interface to NeuroQuery's predictive meta-analytic method, there are functions for downloading the NeuroQuery database and converting it to NiMARE format, much like Neurosynth. The functions for downloading the NeuroQuery database and converting it to a Dataset are **fetch_neuroquery()** and **convert_neurosynth_to_dataset()**, respectively. We are able to use the same function for converting the database to a Dataset for NeuroQuery as Neurosynth because both databases store their data in the same structure.

```
# Download the desired version of
NeuroQuery from GitHub.
files = extract.fetch_neuroquery(
    data_dir=data_path,
    version="1",
    source="combined",
    vocab="neuroquery6308",
    type="tfidf",
    overwrite=False,
)
pprint(files)
neuroquery_db = files[0]
```

```
INFO:nimare.extract.utils:Dataset found in ./../data/nimare-paper/
data/neuroquery
INFO:nimare.extract.extract:Searching for any feature files
matching the following criteria: [('source-combined', 'vocab-
neuroquery6308', 'type-tfidf', 'data-neuroquery', 'version-1')]
Downloading data-neuroquery_version-1_coordinates.tsv.gz
File exists and overwrite is False. Skipping.
Downloading data-neuroquery_version-1_metadata.tsv.gz
File exists and overwrite is False. Skipping.
Downloading data-neuroquery_version-1_vocab-
neuroquery6308_source-combined_typetfidf_
features.npz
File exists and overwrite is False. Skipping.
Downloading data-neuroquery_version-1_vocab-
neuroquery6308_vocabulary.txt
File exists and overwrite is False. Skipping.
[{'coordinates': '/Users/taylor/Documents/nbc/nimare-paper/
data/nimarepaper/data/neuroquery/data-neuroquery_version-1_
coordinates.tsv.gz',
  'features': [{'features': '/Users/taylor/Documents/nbc/nimare-
paper/data/nimarepaper/data/neuroquery/data-neuroquery_
version-1_vocab-neuroquery6308_sourcecombined_type-tfidf_
features.npz',
  'vocabulary': '/Users/taylor/Documents/nbc/nimare-paper/
data/nimarepaper/data/neuroquery/data-neuroquery_version-1_
vocab-neuroquery6308_vocabulary.txt'}],
  'metadata': '/Users/taylor/Documents/nbc/nimare-paper/
data/nimarepaper/data/neuroquery/data-neuroquery_version-1_
metadata.tsv.gz'}]
```

```
# Convert the files to a Dataset.
# This may take a while (~10 minutes)
neuroquery_dset = io.convert_neurosynth_to_dataset(
    coordinates_file=neuroquery_db["coordinates"],
    metadata_file=neuroquery_db["metadata"],
    annotations_files=neuroquery_db["features"],
)
print(neuroquery_dset)

# Save the Dataset for later use.
neuroquery_dset.save(os.path.join(out_dir, "neuroquery_dataset.
pkl.gz"))
```

Here, we load a pre-generated version of the NeuroQuery Dataset.

```
neuroquery_dset = dataset.Dataset.load(os.path.join(data_path,
"neuroquery_dataset.pkl.gz"))
print(neuroquery_dset)
```

```
Dataset(13459 experiments, space='mni152_2mm')
```

 CC By 4.0: © Taylor Salo et al.

NeuroVault

NeuroVault[33] is a public repository of user-uploaded, whole-brain, unthresholded brain maps. Users may associate their image collections with publications, and can annotate individual maps with labels from the Cognitive Atlas, which is the ontology of choice for NeuroVault. NiMARE includes a function, **convert_neurovault_to_dataset()**, with which users can search for images in NeuroVault, download those images, and convert them into a Dataset object.

COORDINATE-BASED META-ANALYSIS

Coordinate-based meta-analysis (CBMA) is currently the most popular method for neuroimaging meta-analysis, given that the majority of fMRI papers currently report their findings as peaks of statistically significant clusters in standard space and do not release unthresholded statistical maps. These peaks indicate where significant results were found in the brain, and thus do not reflect an effect size estimate for each hypothesis test (i.e., each voxel) as one would expect for a typical meta-analysis. As such, standard methods for effect size-based meta-analysis cannot be applied. Over the past two decades, a number of algorithms have been developed to determine whether peaks converge across experiments in order to identify locations of consistent or specific activation associated with a given hypothesis.[34,35]

Kernel-based methods evaluate convergence of coordinates across studies by first convolving foci with a spatial Kernel to produce study-specific modeled activation maps, then combining those modeled activation maps into a sample-wise map, which is compared to a null distribution to evaluate voxel-wise statistical significance. Additionally, for each of the following approaches, except for specific coactivation likelihood estimation (SCALE), voxel- or cluster-level multiple comparisons correction may be performed using Monte Carlo simulations or FDR[36] correction. Basic multiple-comparisons correction methods (e.g., Bonferroni correction) are also supported.

CBMA kernels

CBMA kernels are available as **KernelTransformers** in the **nimare.meta.kernel** module. There are three standard kernels that are currently available: **MKDAKernel**, **KDAKernel**, and **ALEKernel**. Each class may be configured with certain parameters when a new object is initialized. For example, MKDAKernel accepts an r parameter, which determines the radius of the spheres that will be created around each peak coordinate. ALEKernel automatically uses the sample size associated with each experiment in the Dataset to determine the appropriate full-width-at-half-maximum of its Gaussian distribution, as described in Eickhoff et al.[37]; however, users may provide a constant sample_size or fwhm parameter when sample size information is not available within the Dataset metadata.

Here we show how these three kernels can be applied to the same Dataset.

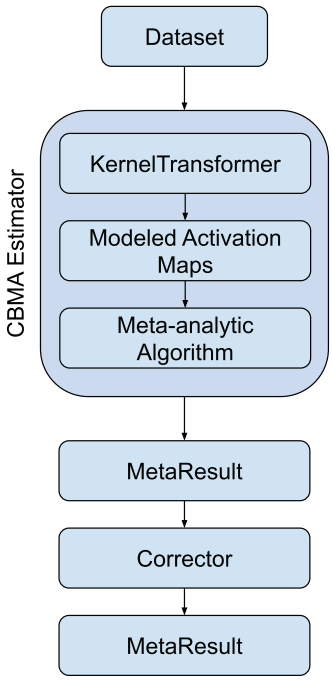

Fig. 3. A flowchart of the typical workflow for coordinate-based meta-analyses in NiMARE.

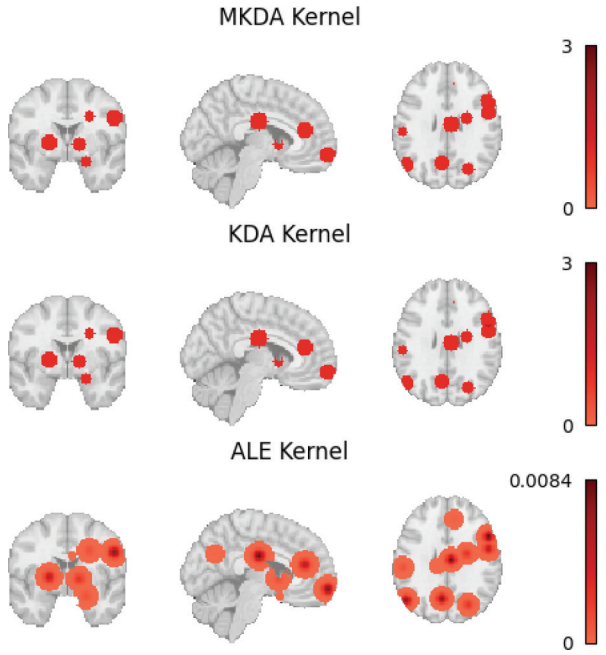

Fig. 4. Modeled activation maps produced by NiMARE's KernelTransformer classes.

 CC By 4.0: © Taylor Salo et al.

```
from nimare.meta import kernel

mkda_kernel = kernel.MKDAKernel(r=10)
mkda_ma_maps = mkda_kernel.transform(sleuth_dset1)
kda_kernel = kernel.KDAKernel(r=10)
kda_ma_maps = kda_kernel.transform(sleuth_dset1)
ale_kernel = kernel.ALEKernel(sample_size=20)
ale_ma_maps = ale_kernel.transform(sleuth_dset1)
```

```
from nimare import dataset, meta

neurosynth_dset_first500 = dataset.Dataset.load(
os.path.join(data_path, "neurosynth_dataset_first500.pkl.gz")
)

# Specify where images for this Dataset
should be located
target_folder = os.path.join(out_dir, "neurosynth_dataset_maps")
os.makedirs(target_folder, exist_ok=True)
neurosynth_dset_first500.update_path(target_folder)

# Initialize a kernel transformer to use
kern = meta.kernel.MKDAKernel(memory_limit="500mb")

# Run the kernel transformer with return_
type set to "dataset" to return an updated
Dataset
# with the MA maps stored as files within
its "images" attribute.
neurosynth_dset_first500 = kern.
transform(neurosynth_dset_first500,
return_type="dataset")
neurosynth_dset_first500.save(
    os.path.join(out_dir, "neurosynth_dataset_first500_with_mkda_
    ma.pkl.gz"),
)
```

```
INFO:nimare.utils:Shared path detected: '/Users/
taylor/Documents/nbc/nimarepaper/ outputs/
neurosynth_dataset_maps/'
```

Multilevel Kernel density analysis

Multilevel Kernel density analysis (MKDA)[38] is a Kernel-based method that convolves each peak from each study with a binary sphere of a set radius. These peak-specific binary maps are then combined into study-specific maps by taking the maximum value for each voxel. Study-specific maps are then averaged across the meta-analytic sample. This averaging is generally weighted by studies' sample sizes, although other covariates may be included, such as weights based on the type of inference (random or fixed effects) employed in the study's analysis. An arbitrary threshold is generally employed to zero-out voxels with very low values, and then a Monte Carlo procedure is used to assess statistical significance, either at the voxel or cluster level.

In NiMARE, the MKDA meta-analyses can be performed with the **MKDADensity** class. This class, like most other CBMA classes in NiMARE, accepts a null_method parameter, which determines how voxel-wise (uncorrected) statistical significance is calculated.

The null_method parameter allows two options: "approximate" or "montecarlo." The "approximate" option builds a histogram-based null distribution of summary-statistic values, which can then be used to determine the associated p-value for **observed** summary-statistic values (i.e., the values in the meta-analytic map). The "montecarlo" option builds a null distribution of summary-statistic values by randomly shuffling the coordinates the Dataset many times, and computing the summary-statistic values for each permutation. In general, the "montecarlo" method is slightly more accurate when there are enough permutations, while the "approximate" method is much faster.

Fitting the CBMA Estimator to a Dataset will produce p-value, z-statistic, and summary-statistic maps, but these are not corrected for multiple comparisons.

When performing a meta-analysis with the goal of statistical inference, you will want to perform multiple comparisons correction with NiMARE's Corrector classes. Please see the multiple comparisons correction chapter for more information.

Here we perform an MKDADensity meta-analysis on one of the Sleuth-based Datasets. We will use the "approximate" null method for speed.

```
from nimare.meta.cbma import mkda

mkdad_meta = mkda.MKDADensity(null_method="approximate")
mkdad_results = mkdad_meta.fit(sleuth_dset1)
```

The MetaResult class

Fitting an Estimator to a Dataset produces a **MetaResult** object. The MetaResult class is a light container holding the different statistical maps produced by the Estimator.

```
print(mkdad_results)
```

```
<nimare.results.MetaResult object at 0x7fdaeb186640>
```

This result is also retained as an attribute in the Estimator.

```
print(mkdad_meta.results)
```

```
<nimare.results.MetaResult object at 0x7fdaeb186640>
```

 CC By 4.0: © Taylor Salo et al.

The maps attribute is a dictionary containing statistical map names and associated numpy arrays.

```
pprint(mkdad_results.maps)
```

```
{'p': array([1., 1., 1., ..., 1., 1., 1.]),
'stat': array([0., 0., 0., ..., 0., 0., 0.]),
'z': array([0., 0., 0., ..., 0., 0., 0.])}
```

These arrays can be transformed into image-like objects using the masker attribute. We can also use the get_map method to get that image object.

```
mkdad_img = mkdad_results.get_map("z", return_type="image")
print(mkdad_img)
```

```
<class 'nibabel.nifti1.Nifti1Image'>
data shape (91, 109, 91)
affine:
[[ -2.   0.   0.  90.]
 [  0.   2.   0. -126.]
 [  0.   0.   2.  -72.]
 [  0.   0.   0.   1.]]
metadata:
<class 'nibabel.nifti1.Nifti1Header'> object, endian='<'
sizeof_hdr      : 348
data_type       : b''
db_name         : b''
extents         : 0
session_error   : 0
regular         : b''
dim_info        : 0
dim             : [  3  91 109  91   1   1   1   1]
intent_p1       : 0.0
intent_p2       : 0.0
intent_p3       : 0.0
intent_code     : none
datatype        : float64
bitpix          : 64
slice_start     : 0
pixdim          : [-1.  2.  2.  2.  1.  1.  1.  1.]
vox_offset      : 0.0
scl_slope       : nan
scl_inter       : nan
slice_end       : 0
slice_code      : unknown
xyzt_units      : 0
cal_max         : 0.0
cal_min         : 0.0
slice_duration  : 0.0
toffset         : 0.0
glmax           : 0
glmin           : 0
descrip         : b''
aux_file        : b''
qform_code      : unknown
sform_code      : aligned
quatern_b       : 0.0
quatern_c       : 1.0
quatern_d       : 0.0
qoffset_x       : 90.0
qoffset_y       : -126.0
qoffset_z       : -72.0
srow_x          : [-2.  0.  0.  90.]
srow_y          : [  0.  2.  0.  -126.]
srow_z          : [  0.  0.  2.  -72.]
intent_name     : b''
magic           : b'n+1'
```

We can save the statistical maps to an output directory as gzipped nifti files, with a prefix. Here, we will save all of the statistical maps with the MKDADensity prefix.

```
mkdad_results.save_maps(output_dir=out_dir,
prefix="MKDADensity")
```

We will also save the Estimator itself, which we will reuse when we get to multiple comparisons correction.

```
mkdad_meta.save(os.path.join(out_dir, "MKDADensity.pkl.gz"))
```

Since this is a Kernel-based algorithm, the Kernel transformer is an optional input to the meta-analytic estimator, and can be controlled in a more fine- grained manner.

```
# These two approaches (initializing the
kernel ahead of time or
# providing the arguments with the kernel__
prefix) are equivalent.
mkda_kernel = kernel.MKDAKernel(r=2)
mkdad_meta = mkda.
MKDADensity(kernel_transformer=mkda_kernel)
mkdad_meta = mkda.MKDADensity(kernel_transformer=kernel.
MKDAKernel, kernel__r=2)

# A completely different kernel could even
be provided, although this is not
# recommended and should only be used for
testing algorithms.
mkdad_meta = mkda.MKDADensity(kernel_transformer=kernel.
KDAKernel)
```

Kernel density analysis

Kernel density analysis (KDA)[39,40] is a precursor algorithm that has been replaced in the field by MKDA. For the sake of completeness, NiMARE also includes a KDA estimator that implements the older KDA algorithm for comparison purposes. The interface is virtually identical, but since there are few if any legitimate uses of KDA (which models studies as fixed rather than random effects), we do not discuss the algorithm further here.

 CC By 4.0: © Taylor Salo et al.

```
kda_meta = mkda.KDA(null_method="approximate")
kda_results = kda_meta.fit(sleuth_dset1)

# Retain the z-statistic map for later use
kda_img = kda_results.get_map("z", return_type="image")
```

Activation likelihood estimation

ALE[41–43] assesses convergence of peaks across studies by first generating a modeled activation map for each study, in which each of the experiment's peaks is convolved with a 3D Gaussian distribution determined by the experiment's sample size, and then by combining these modeled activation maps across studies into an ALE map, which is compared with an empirical null distribution to assess voxel-wise statistical significance.

```
from nimare.meta.cbma import ale

ale_meta = ale.ALE()
ale_results = ale_meta.fit(sleuth_dset1)

# Retain the z-statistic map for later use
ale_img = ale_results.get_map("z", return_type="image")
```

Specific coactivation likelihood estimation

SCALE[44] is an extension of the ALE algorithm developed for meta-analytic coactivation modeling (MACM) analyses. Rather than comparing convergence of foci within the sample to a null distribution derived under the assumption of spatial randomness within the brain, SCALE assesses whether the convergence at each voxel is greater than in the general literature. Each voxel in the brain is assigned a null distribution determined based on the base rate of activation for that voxel across an existing coordinate-based meta-analytic database. This approach allows for the generation of a statistical map for the sample, but no methods for multiple comparisons correction have yet been developed. While this method was developed to support analysis of joint activation or "coactivation" patterns, it is generic and can be applied to any CBMA; see Derivative Analyses.

```
# Here we use the coordinates from
Neurosynth as our measure of coordinate
# base-rates, because we do not have access
to the full BrainMap database.
# However, one assumption of SCALE is that
the Dataset being analyzed comes
# from the same source as the database you
use for calculating base-rates.
xyz = neurosynth_dset.coordinates[["x", "y", "z"]].values
# Typically, you would have >=2500
iterations, but we're using 500 here.
scale_meta = ale.SCALE(n_iters=500, xyz=xyz, memory_
limit="100mb", n_cores=1) scale_results = scale_meta.
fit(sleuth_dset1)

# Retain the z-statistic map for later use
scale_img = scale_results.get_map("z", return_type="image")
```

```
100%      500/500 [03  38<00  00, 2.58it/s]
100%      228483/228483 [02  22<00  00, 3108.27it/s]
```

MKDA Chi-squared Analysis

An alternative to the density-based approaches (i.e., MKDA, KDA, ALE, and SCALE) is the **MKDA Chi-squared** extension.[38] Although still a Kernel-based method in which foci are convolved with a binary sphere and combined within studies, this approach uses voxel-wise Chi-squared tests to assess both consistency (i.e., higher convergence of foci within the meta-analytic sample than expected by chance) and specificity (i.e., higher convergence of foci within the meta-analytic sample than detected in an unrelated dataset) of activation. Such an analysis also requires access to a reference meta-analytic sample or database of studies. For example, to perform a Chi-squared analysis of working memory studies, the researcher will also need a comprehensive set of studies which did not manipulate working memory—ideally one that is matched with the working memory study set on all relevant attributes except the involvement of working memory.

```
mkdac_meta = mkda.MKDAChi2()
mkdac_results = mkdac_meta.fit(sleuth_dset1, sleuth_dset2)

# Retain the specificity analysis's
z-statistic map for later use
mkdac_img = mkdac_results.get_map("z_desc-specificity",
return_type="image")
```

Comparing algorithms

Here, we load the z-statistic map from each of the CBMA estimators we have used throughout this chapter and plot them all side by side.

```
meta_results = {
    "MKDA Density": mkdad_img,
    "MKDA Chi-Squared": mkdac_img,
    "KDA": kda_img,
    "ALE": ale_img,
    "SCALE": scale_img,
}
order = [
    ["MKDA Density", "ALE"],
    ["MKDA Chi-Squared", "SCALE"],
    ["KDA", None]
]

fig, axes = plt.subplots(figsize=(12, 6), nrows=3, ncols=2)
```

 CC By 4.0: © Taylor Salo et al.

```
for i_row, row_names in enumerate(order):
    for j_col, name in enumerate(row_names):
        if not name:
            axes[i_row, j_col].axis("off")
            continue

        img = meta_results[name]
        if name == "MKDA Chi-Squared":
            cmap = "RdBu_r"
        else:
            cmap = "Reds"

        display = plotting.plot_stat_map( img,
            annotate=False,
            axes=axes[i_row, j_col],
            cmap=cmap,
            cut_coords=[5, -15, 10],
            draw_cross=False,
            figure=fig,
        )
        axes[i_row, j_col].set_title(name)

        colorbar = display._cbar
        colorbar_ticks = colorbar.get_ticks()
        if colorbar_ticks[0] < 0:
            new_ticks = [colorbar_ticks[0], 0, colorbar_ticks[-1]]
        else:
            new_ticks = [colorbar_ticks[0], colorbar_ticks[-1]]
        colorbar.set_ticks(new_ticks, update_ticks=True)

glue("figure_cbma_uncorr", fig, display=False)
```

A number of other coordinate-based meta-analysis algorithms exist, which are not yet implemented in NiMARE. We describe these algorithms briefly in Future Directions.

IMAGE-BASED META-ANALYSIS

Image-based meta-analysis (IBMA) methods perform a meta-analysis directly on brain images (either whole-brain or partial) rather than on extracted peaks. On paper, IBMA is superior to CBMA in virtually all respects, as the availability of analysis-level parameter and variance estimates at all analyzed voxels allows researchers to use the full complement of standard meta-analysis techniques, instead of having to resort to Kernel-based or other methods that require additional spatial assumptions. In principle, given a set of maps that contains no missing values (i.e., where there are *k* valid pairs of parameter and variance estimates at each voxel), one can simply conduct a voxel-wise version of any standard meta-analysis or meta-regression method commonly used in other biomedical or social science fields.

In practice, the utility of IBMA methods has historically been quite limited, as unthresholded statistical maps have been unavailable for the vast majority of neuroimaging studies. However, the introduction and rapid adoption of NeuroVault,[33] a database for unthresholded statistical images, has made image-based meta-analysis increasingly viable. Although coverage of the literature remains limited, and IBMAs of maps drawn from the NeuroVault database are likely to omit at least some (and in some cases most) relevant studies due to limited metadata, we believe the time is ripe for researchers to start including both CBMAs and IBMAs in published meta-analyses, with the aspirational goal of eventually transitioning exclusively to the latter. To this end, NiMARE supports a range of different IBMA methods, including a number of estimators of the gold standard mixed-effects meta-regression model, as well as several alternative estimators suitable for use when some of the traditional inputs are unavailable.

NiMARE's IBMA Estimators are light wrappers around classes from PyMARE, a library for standard (i.e., non-neuroimaging) meta-analyses developed by the same team as NiMARE.

In the optimal situation, meta-analysts have access to both contrast (i.e., parameter estimate) maps and their associated standard error maps for a number of studies. With these data, researchers can fit the traditional random-effects

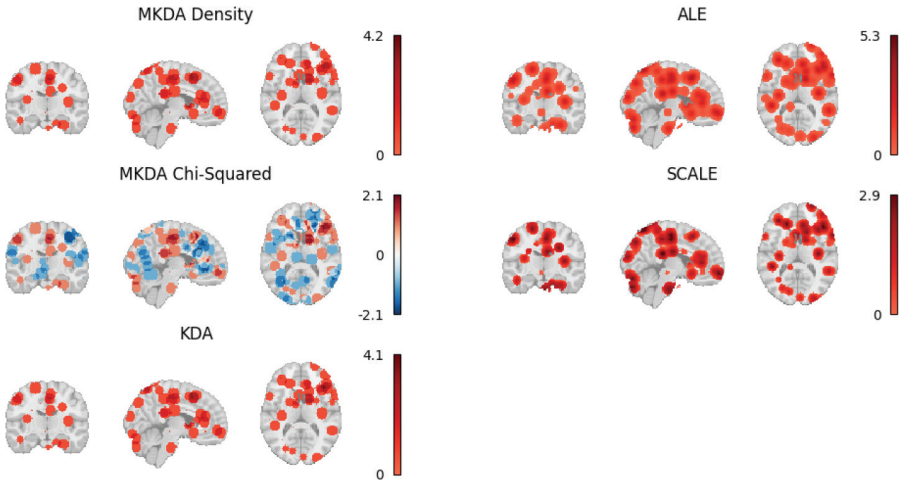

Fig. 5. Thresholded results from MKDA density, KDA, ALE, and SCALE meta-analyses.

 CC By 4.0: © Taylor Salo et al.

meta-regression model using one of several methods that vary in the way they estimate the between-study variance (τ^2). Currently, supported estimators include the **DerSimonian-Laird** method,[45] the **Hedges** method,[46] and **maximum-likelihood** (ML) and **restricted maximum-likelihood** (REML) approaches. NiMARE can also perform fixed-effects meta-regression via weighted least-squares, although there are few IBMA scenarios where a fixed-effects analysis would be indicated. It is worth noting that the non-likelihood-based estimators (i.e., DerSimonian-Laird and Hedges) have a closed-form solution and are implemented in an extremely efficient way in NiMARE (i.e., computation is performed on all voxels in parallel). However, these estimators also produce more biased estimates under typical conditions (e.g., when sample sizes are very small), implying a tradeoff from the user's perspective.

Alternatively, when users only have access to contrast maps and associated sample sizes, they can use the supported sample size-based likelihood estimator, which assumes that within-study variance is constant across studies, and uses maximum-likelihood or restricted maximum-likelihood to estimate between-study variance, as described in Sangnawakij et al..[47] When users have access only to contrast maps, they can use the permuted **OLS** estimator, which uses ordinary least squares and employs a max-type permutation scheme for family-wise error correction [48,49] that has been validated on neuroimaging data[50] and relies on the nilearn library.

Finally, when users only have access to z-score maps, they can use either the **Fisher's**[51] or the **Stouffer's**[52] estimators. When sample size information is available, users may incorporate that information into the Stouffer's method, via the method described in.[53]

Given the paucity of image-based meta-analytic Datasets, we have included the tools to build a dataset from a NeuroVault collection of 21 pain studies, originally described in Maumet and Nichols.[54]

```
from nimare import dataset, extract, utils

dset_dir = extract.download_nidm_pain(data_dir=data_
path, overwrite=False) dset_file = os.path.join(utils.get_
resource_path(), "nidm_pain_dset.json") img_dset = dataset.
Dataset(dset_file)

# Point the Dataset toward the images we've
downloaded
img_dset.update_path(dset_dir)
```

```
INFO:nimare.extract.utils:Dataset found in ./../data/nimare-paper/
data/nidm_21pain
```

```
INFO:nimare.utils:Shared path detected: '/Users/taylor/
Documents/nbc/nimarepaper/data/nimare-paper/data/
nidm_21pain/'
```

Transforming images

Researchers may share their statistical maps in many forms, some of which are direct transformations of one another. For example, researchers may share test statistic maps with z-statistics or t-statistics, and, as long as we know the degrees of freedom associated with the t-test, we can convert between the two easily. To that end, NiMARE includes a class, **ImageTransformer**, which will calculate target image types from available ones, as long as the available images are compatible with said transformation.

Here, we use ImageTransformer to calculate z-statistic and variance maps for all studies with compatible images. This allows us to apply more image-based meta-analysis algorithms to the Dataset.

```
import warnings

from nimare import transforms

# The images used in this example have NaNs
in any voxels outside the brain.
# Generally, we recommend having zeros in
masked-out areas,
# but the data are what they are in this
case.
# Nilearn will raise warnings when users
resample images with NaNs.
# This will not cause any problems for this
example, so we will simply filter
# those warnings out.
warnings.filterwarnings(action="ignore",
category=RuntimeWarning, module="nilearn")

img_transformer = transforms.ImageTransformer(target=["z",
"varcope"], overwrite=False)
img_dset = img_transformer.transform(img_dset)
```

```
INFO:nimare.utils:Shared path detected: '/Users/taylor/
Documents/nbc/nimarepaper/data/nimare-paper/data/
nidm_21pain/'
```

Now that we have filled in as many gaps in the Dataset as possible, we can start running meta-analyses. We will start with a DerSimonian-Laird meta-analysis **(DerSimonianLaird)**.

```
from nimare import meta

dsl_meta = meta.ibma.DerSimonianLaird(resample=True)
dsl_results = dsl_meta.fit(img_dset)

# Retain the z-statistic map for later use
dsl_img = dsl_results.get_map("z", return_type="image")
```

Now we will apply other available IBMA Estimators to the same Dataset and save their results to files for comparison.

 CC By 4.0: © Taylor Salo et al.

```
# Stouffer's
stouffers_meta = meta.ibma.Stouffers(use_sample_size=False,
resample=True)
stouffers_results = stouffers_meta.fit(img_dset)
stouffers_img = stouffers_results.get_map("z",
return_type="image")
del stouffers_meta, stouffers_results

# Stouffer's with weighting based on sample
size
wstouffers_meta = meta.ibma.Stouffers(use_sample_size=True,
resample=True) wstouffers_results = wstouffers_meta.fit(img_dset)
wstouffers_img = wstouffers_results.get_map("z",
return_type="image")
del wstouffers_meta, wstouffers_results

# Fisher's
fishers_meta = meta.ibma.Fishers(resample=True)
fishers_results = fishers_meta.fit(img_dset)
fishers_img = fishers_results.get_map("z", return_type="image")
del fishers_meta, fishers_results

# Permuted Ordinary Least Squares
ols_meta = meta.ibma.PermutedOLS(resample=True)
ols_results = ols_meta.fit(img_dset)
ols_img = ols_results.get_map("z", return_type="image")
del ols_meta, ols_results

# Weighted Least Squares
wls_meta = meta.ibma.WeightedLeastSquares(resample=True)
wls_results = wls_meta.fit(img_dset)
wls_img = wls_results.get_map("z", return_type="image")
del wls_meta, wls_results

# Hedges'
hedges_meta = meta.ibma.Hedges(resample=True)
hedges_results = hedges_meta.fit(img_dset)
hedges_img = hedges_results.get_map("z", return_type="image")
del hedges_meta, hedges_results

# Use atlas for likelihood-based estimators
from nilearn import datasets, image, input_data

atlas = datasets.
fetch_atlas_harvard_oxford("cort-maxprob-thr25-2mm")

# nilearn's NiftiLabelsMasker cannot handle
NaNs at the moment,
# and some of the NIDM-Results packs' beta
images have NaNs at the edge of the brain.
# So, we will create a reduced version of
the atlas for this analysis.
nan_mask = image.math_img("~np.any(np.isnan(img), axis=3)",
img=img_dset.images["beta"].tolist())
atlas = image.resample_to_img(atlas["maps ], nan_mask)
nanmasked_atlas = image.math_img("mask * atlas", mask=nan_
mask, atlas=atlas)
masker = input_data.NiftiLabelsMasker(nanmasked_atlas)
del atlas, nan_mask, nanmasked_atlas

# Variance-Based Likelihood
vbl_meta = meta.ibma.VarianceBasedLikelihood(method="reml",
mask=masker, resample=True)
vbl_results = vbl_meta.fit(img_dset)
vbl_img = vbl_results.get_map("z", return_type="image")
del vbl_meta, vbl_results

# Sample Size-Based Likelihood
ssbl_meta = meta.ibma.SampleSizeBasedLikelihood(method="reml",
mask=masker, resample=True)
ssbl_results = ssbl_meta.fit(img_dset)
ssbl_img = ssbl_results.get_map("z", return_type="image")
del ssbl_meta, ssbl_results, masker
```

Comparing algorithms

Here, we load the z-statistic map from each of the IBMA Estimators we have used throughout this chapter and plot them all side by side.

```
meta_results = {
        "DerSimonian-Laird": dsl_img,
        "Stouffer's": stouffers_img,
        "Weighted Stouffer's": wstouffers_img,
        "Fisher's": fishers_img,
        "Ordinary Least Squares": ols_img,
        "Weighted Least Squares": wls_img,
        "Hedges'": hedges_img,
        "Variance-Based Likelihood": vbl_img,
        "Sample Size-Based Likelihood": ssbl_img,
}
order = [
        ["Fisher's", "Stouffer's", "Weighted Stouffer's"],
        ["DerSimonian-Laird", "Hedges'", "Weighted Least
        Squares"],
        ["Ordinary Least Squares", "Variance-Based Likelihood",
        "Sample Size-Based Likelihood"],
]

fig, axes = plt.subplots(figsize=(18, 6), nrows=3, ncols=3)

for i_row, row_names in enumerate(order):
        for j_col, name in enumerate(row_names):
        file_ = meta_results[name]
        display = plotting.plot_stat_map(
                file_,
                annotate=False,
                axes=axes[i_row, j_col],
                cmap="RdBu_r",
                cut_coords=[5, -15, 10],
                draw_cross=False,
                figure=fig,
        )
        axes[i_row, j_col].set_title(name)

        colorbar = display._cbar
        colorbar_ticks = colorbar.get_ticks()
        if colorbar_ticks[0] < 0:
                new_ticks = [colorbar_ticks[0], 0, colorbar_ticks[-1]]
        else:
                new_ticks = [colorbar_ticks[0], colorbar_ticks[-1]]
                colorbar.set_ticks(new_ticks, update_ticks=True)

glue("figure_uncorr_ibma", fig, display=False)
```

 CC By 4.0: © Taylor Salo et al.

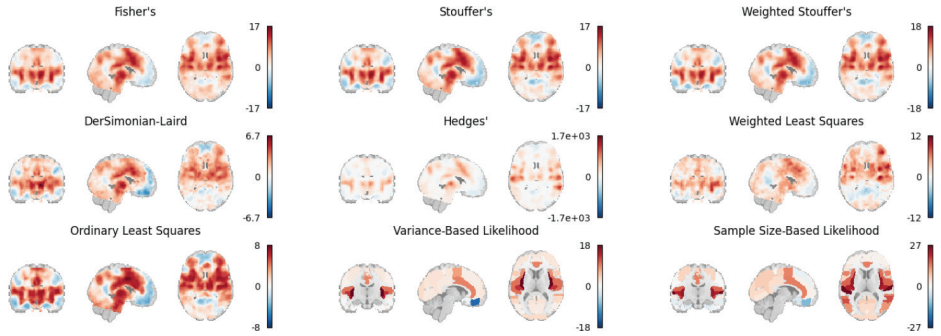

Fig. 6. An array of plots of the statistical maps produced by the image-based meta-analysis methods. The likelihood-based meta-analyses are run on atlases instead of voxelwise.

MULTIPLE COMPARISONS CORRECTION

In NiMARE, multiple comparisons correction is separated from each CBMA and IBMA Estimator so that any number of relevant correction methods can be applied after the Estimator has been fit to the Dataset. Some correction options, such as the montecarlo option for FWE correction, are designed to work specifically with a given Estimator (and are indeed implemented within the Estimator class, and only called by the Corrector).

Correctors are divided into two subclasses: FWECorrectors, which correct based on family-wise error rate, and FDRCorrectors, which correct based on FDR.

All Correctors are initialized with a number of parameters, including the correction method that will be used. After that, you can use the transform method on a MetaResult object produced by a CBMA or IBMA Estimator to apply the correction method. This will return an updated MetaResult object, with both the statistical maps from the original MetaResult, as well as new, corrected maps.

Here we will apply both FWE and FDR correction to results from a MKDADensity meta-analysis, performed back in multilevel Kernel density analysis.

In the following example, we use 5000 iterations for Monte Carlo FWE correction. Normally, one would use at least 10,000 iterations, but we reduced this for the sake of speed.

```
from nimare import meta, correct

mkdad_meta = meta.cbma.mkda.MKDADensity.load(os.path.
join(data_path, "MKDADensity.pkl.gz"))

mc_corrector = correct.FWECorrector(method="montecarlo",
n_iters=5000, n_cores=4)
mc_results = mc_corrector.transform(mkdad_meta.results)
mc_results.save_maps(output_dir=out_dir,
prefix="MKDADensity_FWE")

fdr_corrector = correct.FDRCorrector(method="indep")
fdr_results = fdr_corrector.transform(mkdad_meta.results)
```

```
INFO:nimare.correct:Using correction method implemented
in Estimator: nimare.meta.cbma.mkda.MKDADensity.
correct_fwe_montecarlo.

100%      5000/5000 [10 29<00 00, 7.18it/s]

INFO:nimare.meta.cbma.base:Using null distribution for voxel-
level FWE correction.
```

Statistical maps saved by NiMARE MetaResults automatically follow a naming convention based loosely on the Brain Imaging Data Standard (BIDS). Let's take a look at the files created by the FWECorrector.

```
from glob import glob

fwe_maps = sorted(glob(os.path.join(out_dir, "MKDADensity_
FWE*.nii.gz")))
fwe_maps = [os.path.basename(fwe_map) for fwe_map in
fwe_maps]
print("\n".join(fwe_maps))
```

```
MKDADensity_FWE_logp_desc-mass_level-cluster_corr-FWE_
method-montecarlo.nii.gz
MKDADensity_FWE_logp_desc-size_level-cluster_corr-FWE_
method-montecarlo.nii.gz
MKDADensity_FWE_logp_level-voxel_corr-FWE_method-
montecarlo.nii.gz
MKDADensity_FWE_p.nii.gz
MKDADensity_FWE_stat.nii.gz
MKDADensity_FWE_z.nii.gz
MKDADensity_FWE_z_desc-mass_level-cluster_corr-FWE_method-
montecarlo.nii.gz
MKDADensity_FWE_z_desc-size_level-cluster_corr-FWE_method-
montecarlo.nii.gz
MKDADensity_FWE_z_level-voxel_corr-FWE_method-montecarlo.nii.gz
```

If you ignore the prefix, which was specified in the call to MetaResult.save_maps, the maps all have a common naming convention. The maps from the original meta-analysis (before multiple comparisons correction)

 CC By 4.0: © Taylor Salo et al.

are simply named according to the values contained in the map (e.g., z, stat, p).

Maps generated by the correction method, however, use a series of key-value pairs to indicate how they were generated. The corr key indicates whether FWE or FDR correction was applied. The method key reflects the correction method employed, which was defined by the method parameter used to create the Corrector. The level key simply indicates if the map was corrected at the voxel or cluster level. Finally, the desc key reflects any necessary description that goes beyond what is already covered by the other entities.

DERIVATIVE ANALYSES

Meta-analytic databases and algorithms may be employed for derivative analyses, including subtraction analysis, meta-analytic coactivation modeling (MACM), meta-analytic clustering, coactivation-based parcellation (CBP), meta-analytic independent component analysis (meta-ICA), semantic model development, and meta-analytic functional decoding. In this part, we describe the derivative analyses implemented in NiMARE and include examples of use cases.

META-ANALYTIC SUBTRACTION ANALYSIS

Subtraction analysis refers to the voxel-wise comparison of two meta-analytic samples. In image-based meta-analysis, comparisons between groups of maps can generally be accomplished within the standard meta-regression framework (i.e., by adding a covariate that codes for group membership). However, coordinate-based subtraction analysis requires special extensions for CBMA algorithms.

Subtraction analysis to compare the results of two ALE meta-analyses was originally implemented by[17] and later extended by Eickhoff et al.[37]. In this approach, two groups of experiments (A and B) are compared using a group assignment randomization procedure in which voxel-wise null distributions are generated by randomly reassigning experiments between the two groups and calculating ALE-difference scores for each permutation. Real ALE-difference scores (i.e., the ALE values for one group minus the ALE values for the other) are compared against these null distributions to determine voxel-wise significance. In the original implementation of the algorithm, this procedure is performed separately for a group A > B contrast and a group B > A contrast, where each contrast is limited to voxels that were significant in the first group's original meta-analysis.

In NiMARE, we use an adapted version of the subtraction analysis method in **ALESubtraction**. The NiMARE implementation analyzes all voxels, rather than only those that show a significant effect of A alone or B alone as in the original implementation.

Running a subtraction analysis with the standard number of iterations (10,000) may require more than 4 GB of RAM, which is NeuroLibre's limit. We will instead use only 1000 iterations so that the analysis will run successfully on NeuroLibre's server. For publication-quality subtraction analyses, we recommend using the standard 10,000 iterations.

```
from nimare import meta

kern = meta.kernel.ALEKernel()
sub_meta = meta.cbma.ale.ALESubtraction(kernel_
transformer=kern, n_iters=1000)
sub_results = sub_meta.fit(sleuth_dset1, sleuth_dset2)
```

Alternatively, MKDA Chi-squared analysis is inherently a subtraction analysis method, in that it compares foci from two groups of studies. Generally, one of these groups is a sample of interest, while the other is a meta-analytic database (minus the studies in the sample). With this setup, meta-analysts can infer whether there is greater convergence of foci in a voxel as compared to the baseline across the field (as estimated with the meta-analytic database), much like SCALE. However, if the database is replaced with a second sample of interest, the analysis ends up comparing convergence between the two groups.

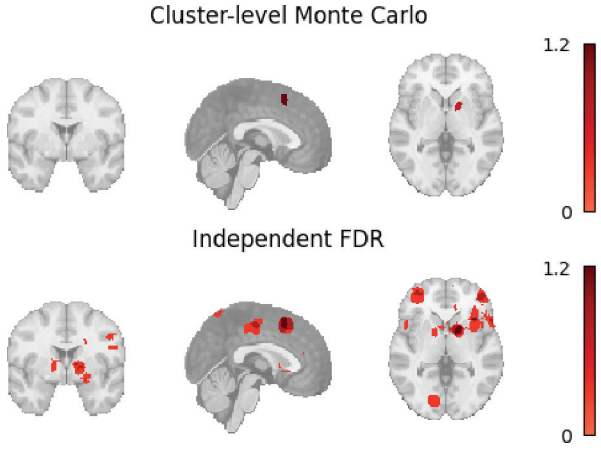

Fig. 7. An array of plots of the corrected statistical maps produced by the different multiple comparisons correction methods.

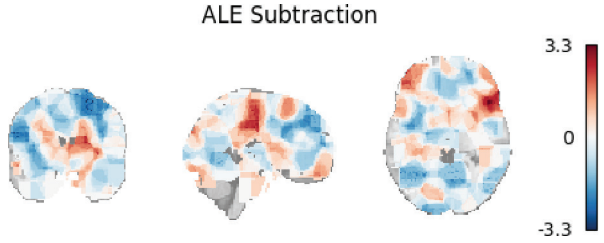

Fig. 8. Unthresholded z-statistic map for the subtraction analysis of the two example Sleuth-based Datasets.

 CC By 4.0: © Taylor Salo et al.

META-ANALYTIC COACTIVATION MODELING

Meta-analytic coactivation modeling (MACM),[55–57] also known as meta-analytic connectivity modeling, uses meta-analytic data to measure co-occurrence of activations between brain regions providing evidence of functional connectivity of brain regions across tasks. In coordinate-based MACM, whole-brain studies within the database are selected based on whether or not they report at least one peak in a region of interest specified for the analysis. These studies are then subjected to a meta-analysis, often comparing the selected studies to those remaining in the database. In this way, the significance of each voxel in the analysis corresponds to whether there is greater convergence of foci at the voxel among studies, which also report foci in the region of interest than those which do not.

MACM results have historically been accorded a similar interpretation to task-related functional connectivity (e.g.[58,59]), although this approach is quite removed from functional connectivity analyses of task fMRI data (e.g., beta-series correlations, psychophysiological interactions, or even seed-to-voxel functional connectivity analyses on task data). Nevertheless, MACM analyses do show high correspondence with resting-state functional connectivity.[60] MACM has been used to characterize the task-based functional coactivation of the cerebellum,[61] lateral prefrontal cortex,[62] fusiform gyrus,[63] and several other brain regions.

Within NiMARE, MACMs can be performed by selecting studies in a Dataset based on the presence of activation within a target mask or coordinate-centered sphere. While some algorithms, such as SCALE, may have been designed with MACMs in mind, in practice MACMs may be performed with any valid Estimator.

In this section, we will perform two MACMs – one with a target mask and one with a coordinate-centered sphere. For the former, we use **get_studies_by_mask()**. For the latter, we use **get_studies_by_coordinate()**.

```
# Create Dataset only containing studies
with peaks within the amygdala mask
amygdala_mask = os.path.join(data_path, "amygdala_roi.nii.gz")
amygdala_ids = neurosynth_dset.
get_studies_by_mask(amygdala_mask)
dset_amygdala = neurosynth_dset.slice(amygdala_ids)

# Create Dataset only containing studies
with peaks within the sphere ROI
sphere_ids = neurosynth_dset.get_studies_by_coordinate([[24,
-2, -20]], r=6)
dset_sphere = neurosynth_dset.slice(sphere_ids)
```

The amygdala dataset includes more than 1300 studies. Running a meta-analysis on such a large dataset may require more than 4 GB of RAM, which is NeuroLibre's limit. Therefore, we will further reduce the dataset to its first 500 studies, in order to run the meta-analysis successfully on NeuroLibre's server. For publication-quality analyses, we would recommend using the entire dataset.

```
print(dset_amygdala)
dset_amygdala = dset_amygdala.slice(dset_amygdala.ids[:500])
print(dset_amygdala)
```

```
Dataset(1369 experiments, space='mni152_2mm')
Dataset(500 experiments, space='mni152_2mm')
```

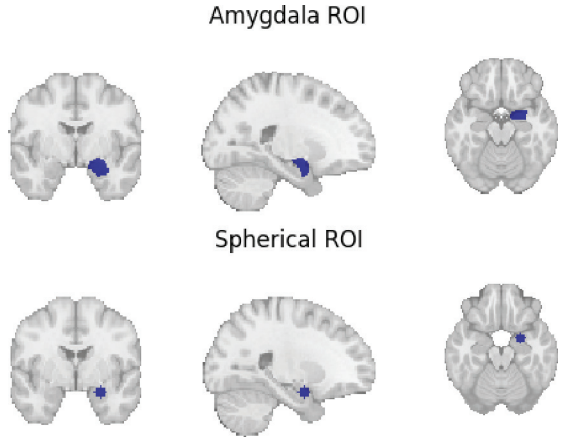

Fig. 9. Region of interest masks for (1) a target mask-based MACM and (2) a coordinate-based MACM.

Once the Dataset has been reduced to studies with coordinates within the mask or sphere requested, any of the supported CBMA Estimators can be run.

```
from nimare import meta

meta_amyg = meta.cbma.ale.ALE(kernel__sample_size=20)
results_amyg = meta_amyg.fit(dset_amygdala)

meta_sphere = meta.cbma.ale.ALE(kernel__sample_size=20)
results_sphere = meta_sphere.fit(dset_sphere)
```

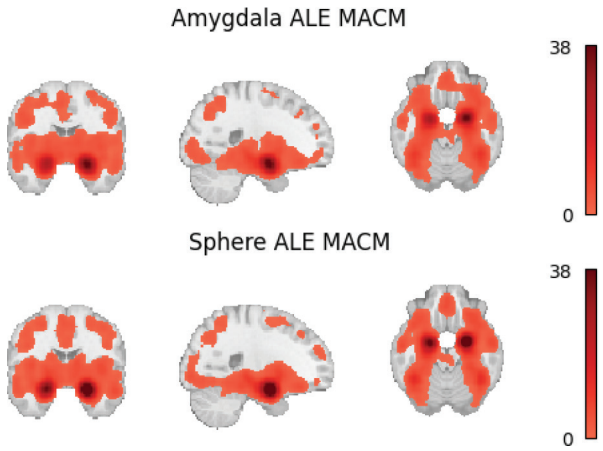

Fig. 10. Unthresholded z-statistic maps for (1) the target mask-based MACM and (2) the coordinate-based MACM.

 CC By 4.0: © Taylor Salo et al.

AUTOMATED ANNOTATION

As mentioned in the discussion of BrainMap (BrainMap), manually annotating studies in a meta-analytic database can be a time-consuming and labor-intensive process. To facilitate more efficient (albeit lower-quality) annotation, NiMARE supports a number of automated annotation approaches. These include N-gram term extraction, Cognitive Atlas term extraction and hierarchical expansion, LDA, and GCLDA.

NiMARE users may download abstracts from PubMed as long as study identifiers in the Dataset correspond to PubMed IDs (as in Neurosynth and NeuroQuery). Abstracts are much more easily accessible than full article text, so most annotation methods in NiMARE rely on them.

Below, we use the function **download_abstracts()** to download abstracts for the Neurosynth Dataset. This will attempt to extract metadata about each study in the Dataset from PubMed, and then add the abstract available on Pubmed to the Dataset's texts attribute, under a new column names "abstract".

download_abstracts() only works when there is internet access. Since this book will often be built on nodes without internet access, we will share the code used to download abstracts but will actually load and use a pre-generated version of the Dataset.

```
# First, load a Dataset without abstracts
neurosynth_dset_first_500 = dataset.Dataset.load(
    os.path.join(data_path, "neurosynth_dataset_first500.pkl.gz")
)
# Now, download the abstracts using your
email address
neurosynth_dset_first_500 = extract.download_abstracts(
    neurosynth_dset_first_500,
    email="example@email.com",
)

# Finally, save the Dataset with abstracts
to a pkl.gz file
neurosynth_dset_first_500.save(
    os.path.join(data_path, "neurosynth_dataset_first500_with_
    abstracts.pkl.gz"),
)
```

```
neurosynth_dset_first_500 = dataset.Dataset.load(
        os.path.join(data_path, "neurosynth_dataset_first500_
        with_abstracts.pkl.gz"),
)
```

N-gram term extraction

N-gram term extraction refers to the vectorization of text into contiguous sets of words that can be counted as individual tokens. The upper limit on the number of words in these tokens is set by the user.

NiMARE has the function **generate_counts()** to extract n-grams from text. This method produces either term counts or term frequency-inverse document frequency (tf-idf) values for each of the studies in a Dataset.

```
from nimare import annotate

counts_df = annotate.text.generate_counts(
    neurosynth_dset_first_500.texts,
    text_column="abstract",
    tfidf=False,
    min_df=10,
    max_df=0.95,
)
```

This term count DataFrame will be used later, to train a GCLDA model.

Cognitive Atlas term extraction and hierarchical expansion

Cognitive Atlas term extraction leverages the structured nature of the Cognitive Atlas in order to extract counts for individual terms and their synonyms in the ontology, as well as to apply hierarchical expansion to these counts based on the relationships specified between terms. This method produces both basic term counts and expanded term counts based on the weights applied to different relationship types present in the ontology.

First, we must use **download_cognitive_atlas()** to download the current version of the Cognitive Atlas ontology. This includes both information about individual terms in the ontology and asserted relationships between those terms.

NiMARE will automatically attempt to extrapolate likely alternate forms of each term in the ontology, in order to make extraction easier. For an example, see Fig. 11.

```
cogatlas = extract.download_cognitive_atlas(data_dir=data_
path, overwrite=False)
id_df = pd.read_csv(cogatlas["ids"])
rel_df = pd.read_csv(cogatlas["relationships"])

cogat_counts_df, rep_text_df = annotate.cogat.extract_cogat(
neurosynth_dset_first_500.texts, id_df, text_column="abstract"
)
```

	id	name	alias
803	trm_4f244ad7dcde7	dot motion task	random-dot motion task
1563	trm_4f244ad7dcde7	dot motion task	dot motion task
1589	trm_4f244ad7dcde7	dot motion task	dot-motion task
1595	trm_4f244ad7dcde7	dot motion task	moving-dot task
2039	trm_4f244ad7dcde7	dot motion task	rdm task

Fig. 11. An example of alternate forms characterized by the Cognitive Atlas and extrapolated by NiMARE. Certain alternate forms (i.e., synonyms) are specified within the Cognitive Atlas, while others are inferred automatically by NiMARE according to certain rules (e.g., removing parentheses).

 CC By 4.0: © Taylor Salo et al.

```
INFO:nimare.extract.utils:Dataset found in ./../data/nimare-paper/data/cognitive_atlas
```

```
# Define a weighting scheme.
# In this scheme, observed terms will also count toward any hypernyms (isKindOf),
# holonyms (isPartOf), and parent categories (inCategory) as well.
weights = {"isKindOf": 1, "isPartOf": 1, "inCategory": 1}
expanded_df = annotate.cogat.expand_counts(cogat_counts_df, rel_df, weights)

# Sort by total count and reduce for better visualization
series = expanded_df.sum(axis=0)
series = series.sort_values(ascending=False)
series = series[series > 0]
columns = series.index.tolist()
```

```
# Raw counts
fig, axes = plt.subplots(figsize=(16, 16), nrows=2, sharex=True)
pos = axes[0].imshow(
    cogat_counts_df[columns].values,
    aspect="auto",
    vmin=0,
    vmax=10,
)
fig.colorbar(pos, ax=axes[0])
axes[0].set_title("Counts Before Expansion", fontsize=20)
axes[0].yaxis.set_visible(False)
axes[0].xaxis.set_visible(False)
axes[0].set_ylabel("Study", fontsize=16)
axes[0].set_xlabel("Cognitive Atlas Term", fontsize=16)

# Expanded counts
pos = axes[1].imshow(
    expanded_df[columns].values,
    aspect="auto",
    vmin=0,
    vmax=10,
)
fig.colorbar(pos, ax=axes[1])
axes[1].set_title("Counts After Expansion", fontsize=20)
axes[1].yaxis.set_visible(False)
axes[1].xaxis.set_visible(False)
axes[1].set_ylabel("Study", fontsize=16)
axes[1].set_xlabel("Cognitive Atlas Term", fontsize=16)

fig.tight_layout()
glue("figure_cogat_expansion", fig, display=False)
```

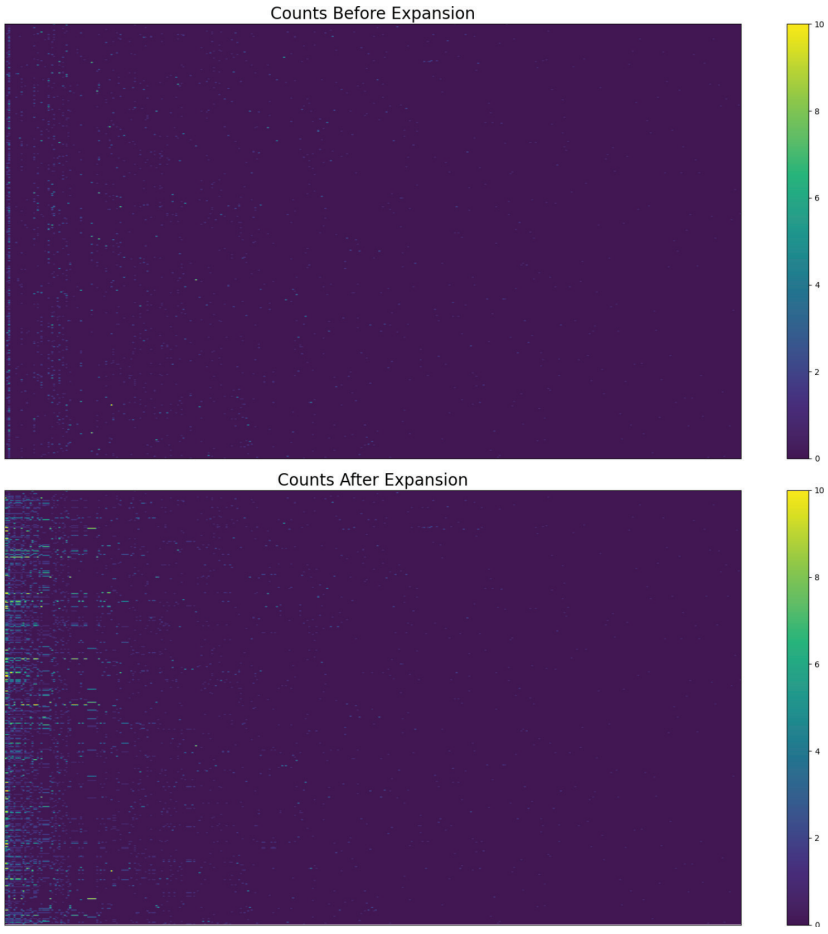

Fig. 12. The effect of hierarchical expansion on Cognitive Atlas term counts from abstracts in Neurosynth's first 500 papers. There are too many terms and studies to show individual labels.

 CC By 4.0: © Taylor Salo et al.

Latent Dirichlet allocation

LDA[64] was originally combined with meta-analytic neuroimaging data in.[23] LDA is a generative topic model which, for a text corpus, builds probability distributions across documents and words. In LDA, each document is considered a mixture of topics. This works under the assumption that each document was constructed by first randomly selecting a topic based on the document's probability distribution across topics, and then randomly selecting a word from that topic based on the topic's probability distribution across words. While this is not a useful generative model for producing documents, LDA is able to discern cohesive topics of related words. Poldrack et al.[23] were able to apply LDA to full texts from neuroimaging articles in order to develop cognitive neuroscience-related topics and to run topic-wise meta- analyses. This method produces two sets of probability distributions: (1) the probability of a word given topic and (2) the probability of a topic given article.

NiMARE's **LDAModel** is a light wrapper around scikit-learn's LDA implementation.

Here, we train an LDA model (**LDAModel**) on the first 500 studies of the Neurosynth Dataset, with 50 topics in the model.

```
from nimare import annotate

lda_model = annotate.lda.LDAModel(n_topics=50, max_
iter=1000, text_column="abstract")

# Fit the model
lda_model.fit(neurosynth_dset_first_500)
```

The most important products of training the LDAModel object is its distributions_ attribute. LDAModel.distributions_ is a dictionary containing arrays and DataFrames created from training the model. We are particularly interested in the p_topic_g_word_df distribution, which is a pandas DataFrame in which each row corresponds to a topic and each column corresponds to a term (n-gram) extracted from the Dataset's texts. The cells contain weights indicating the probability distribution across terms for each topic.

Additionally, the LDAModel updates the Dataset's **annotations** attribute, by adding columns corresponding to each of the topics in the model. Each study in the Dataset thus receives a weight for each topic, which can be used to select studies for topic-based meta-analyses or functional decoding.

Let's take a look at the results of the model training. First, we will reorganize the DataFrame a bit to show the top 10 terms for each of the first 10 topics.

```
lda_df = lda_model.distributions_["p_topic_g_word_df"].T
column_names = {c: f"Topic {c}" for c in lda_df.columns}
lda_df = lda_df.rename(columns=column_names)
temp_df = lda_df.copy()
lda_df = pd.DataFrame(columns=lda_df.columns, index=np.
arange(10)) lda_df.index.name = "Term"
for col in lda_df.columns:
        top_ten_terms = temp_df.sort_values(by=col,
        ascending=False).index.tolist()[:10]
        lda_df.loc[:, col] = top_ten_terms
lda_df = lda_df[lda_df.columns[:10]]
glue("table_lda", lda_df)
```

Term	Topic LDA50__1	Topic LDA50__2	Topic LDA50__3	Topic LDA50__4	Topic LDA50__5	Topic LDA50__6
0	responses stimuli	reward	motor	attentional	networks	genesis
1	items	switch	reaction	selection	imagination	thirst
2	severe	trials	sensory	interference	nuclei	cerebral activations
3	regional	ofc	4a 4p	response selection	involved hand	saline
4	19	rewards	modalities	cingulate	caudate nuclei	flow change
5	plays important	taste	modality	multiple	suggest networks	infusion
6	suggestion	feedback	somatosensory	color	premotor parietal	51
7	resolution interference	win	4p	resources	simulation	regional cerebral
8	string	varying	3b	subserve	experimental conditions	concentration
9	retrieval memories	punishment	tasks	allocation	partly	frontal gyri

Fig. 13. The top 10 terms for each of the first 10 topics in the trained LDA model.

 CC By 4.0: © Taylor Salo et al.

Generalized correspondence latent Dirichlet allocation

GCLDA is a recently-developed algorithm that trains topics on both article abstracts and coordinates.[30] GCLDA assumes that topics within the fMRI literature can also be localized to brain regions, in this case modeled as three-dimensional Gaussian distributions. These spatial distributions can also be restricted to pairs of Gaussians that are symmetric across brain hemispheres. This method produces two sets of probability distributions: the probability of a word given topic (GCLDAModel.p_word_g_topic_), and the probability of a voxel given topic (GCLDAModel.p_voxel_g_topic_).

Here we train a GCLDA model (**GCLDAModel**) on the first 500 studies of the Neurosynth Dataset. The model will include 50 topics, in which the spatial distribution for each topic will be defined as having two Gaussian distributions that are symmetrically localized across the longitudinal fissure.

GCLDAModel generally takes a very long time to train.

Below, we show how one would train a GCLDA model. However, we will load a pretrained model instead of actually training the model.

```
gclda_model = annotate.gclda.GCLDAModel(
    counts_df,
    neurosynth_dset_first_500.coordinates,
    n_regions=2,
    n_topics=50,
    symmetric=True,
    mask=neurosynth_dset_first_500.masker.mask_img,
)
gclda_model.fit(n_iters=2500, loglikely_freq=500)
```

```
gclda_model = annotate.gclda.GCLDAModel.load(os.path.
join(data_path, "gclda_model.pkl.gz"))
```

The GCLDAModel retains the relevant probability distributions in the form of numpy arrays, rather than pandas DataFrames. However, for the topic-term weights (p_word_g_topic_), the data are more interpretable as a DataFrame, so we will create one. We will also reorganize the raw DataFrame to show the top 10 terms for each of the first 10 topics.

```
gclda_arr = gclda_model.p_word_g_topic_
gclda_vocab = gclda_model.vocabulary
topic_names = [f"Topic {str(i).zfill(3)}" for i in range(gclda_arr.
shape[1])] gclda_df = pd.DataFrame(index=gclda_vocab,
columns=topic_names, data=gclda_arr)
temp_df = gclda_df.copy()
```

Term	Topic 000	Topic 001	Topic 002	Topic 003	Topic 004	Topic 005	Top 00
0	amygdala	visual	motion	learning	visual	modality	ag
1	faces	cortex	temporal	task	network	spatial	adul
2	neutral	functional	mt	performance	greater	sensory	matt
3	emotional	cortical	visual	sequence	occipital	gyrus	childre
4	functional	magnetic resonance	sulcus	fronto	frontal	modalities	usin
5	stimuli	visual cortex	occipital	practice	evidence	known	volum
6	emotion	magnetic	dorsal	learned	human	movements	m
7	response	stimulation	human	awareness	demonstrated	bilaterally	youn
8	functional magnetic	resonance	sensitive	early	direction	angular	yea
9	responses	spatial	perception	parietal	pattern	goal	wome

. . . .

Fig. 14. The top 10 terms for each of the first 10 topics in the trained GCLDA model.

 CC By 4.0: © Taylor Salo et al.

```
gclda_df = pd.DataFrame(columns=gclda_df.columns, index=np.
arange(10))
gclda_df.index.name = "Term"
for col in temp_df.columns:
        top_ten_terms = temp_df.sort_values(by=col,
        ascending=False).index.tolist()[:10]
        gclda_df.loc[:, col] = top_ten_terms

gclda_df = gclda_df[gclda_df.columns[:10]]
glue("table_gclda", gclda_df)
```

We also want to see how the topic-voxel weights render on the brain, so we will simply unmask the p_voxel_g_topic_ array with the Dataset's masker.

```
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(12, 10))

topic_img_4d =
neurosynth_dset_first_500.masker.
inverse_transform(gclda_model.p_voxel_g_topic_.T)
# Plot first ten topics
topic_counter = 0
for i_row in range(5):
    for j_col in range(2):
        topic_img = image.index_img(topic_img_4d,
        index=topic_counter)
        display = plotting.plot_stat_map(
            topic_img,
            annotate=False,
            cmap="Reds",
            draw_cross=False,
            figure=fig,
            axes=axes[i_row, j_col],
        )
        axes[i_row, j_col].set_title(f"Topic {str(topic_counter).zfill(3)}")
        topic_counter += 1

        colorbar = display._cbar
        colorbar_ticks = colorbar.get_ticks()
        if colorbar_ticks[0] < 0:
            new_ticks = [colorbar_ticks[0], 0, colorbar_ticks[-1]]
        else:
            new_ticks = [colorbar_ticks[0], colorbar_ticks[-1]]
        colorbar.set_ticks(new_ticks, update_ticks=True)
glue("figure_gclda_topics", fig, display=False)
```

META-ANALYTIC FUNCTIONAL DECODING

Functional decoding performed with meta-analytic data, refers to methods which attempt to predict mental states from neuroimaging data using a large-scale meta-analytic database.[65] Such analyses may also be referred to as "informal reverse inference",[66] "functional characterization analysis",[67–69] "open-ended decoding",[30] or simply "functional decoding".[70–72] While the terminology is far from standardized, we will refer to this method as **meta-analytic functional decoding** in order to

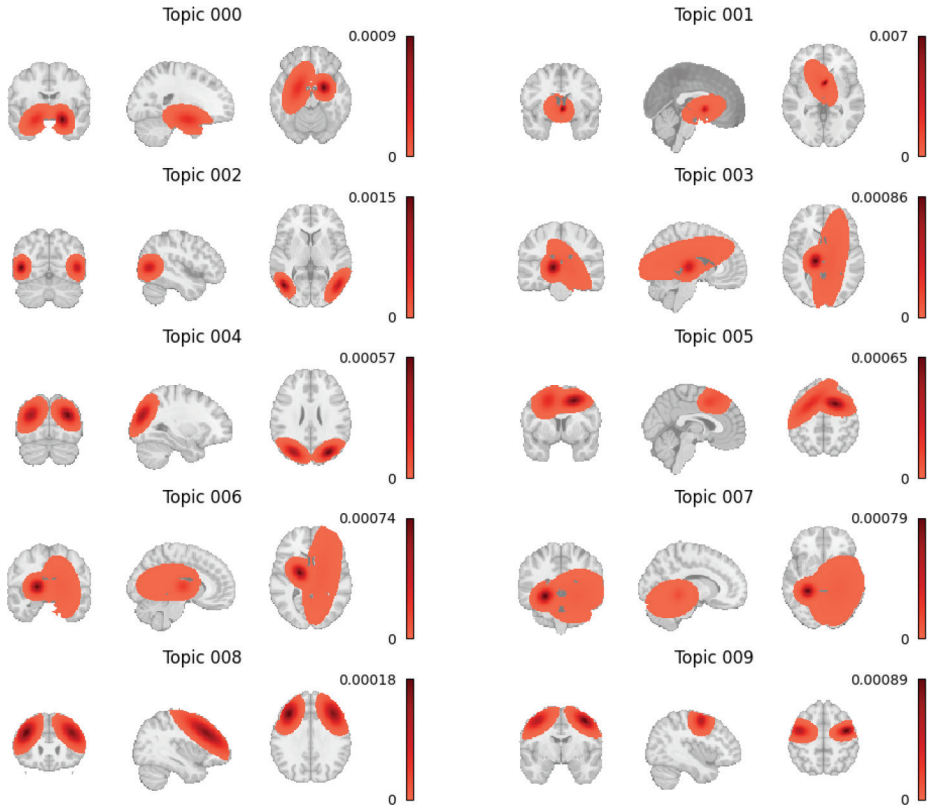

Fig. 15. Topic weight maps for the first 10 topics in the GCLDA model.

 CC By 4.0: © Taylor Salo et al.

distinguish it from alternative methods like multivariate decoding and model-based decoding.[66] Meta-analytic functional decoding is often used in conjunction with MACM, meta-analytic clustering, meta- analytic parcellation, and meta-ICA, in order to characterize resulting brain regions, clusters, or components. Meta-analytic functional decoding models have also been extended for the purpose of **meta-analytic functional encoding**, wherein text is used to generate statistical images.[30,73,74]

Four common approaches are correlation-based decoding, dot-product decoding, weight-sum decoding, and Chi-square decoding. We will first discuss continuous decoding methods (i.e., correlation and dot-product), followed by discrete decoding methods (weight-sum and Chi-square).

Decoding continuous inputs

When decoding unthresholded statistical maps (such as Fig. 16), the most common approaches are to simply correlate the input map with maps from the database, or to compute the dot product between the two maps. In Neurosynth, meta-analyses are performed for each label (i.e., term or topic) in the database and then the input image is correlated with the resulting unthresholded statistical map from each meta-analysis. Performing statistical inference on the resulting correlations is not straightforward, however, as voxels display strong spatial correlations, and the true degrees of freedom are consequently unknown (and likely far smaller than the nominal number of voxels). In order to interpret the results of this decoding approach, users typically select some arbitrary number of top correlation coefficients ahead of time, and use the associated labels to describe the input map. However, such results should be interpreted with great caution.

This approach can also be applied to an image-based database like NeuroVault, either by correlating input data with meta-analyzed statistical maps, or by deriving distributions of correlation coefficients by grouping statistical maps in the database according to label. Using these distributions, it is possible to statistically compare labels in order to assess label significance. NiMARE includes methods for both correlation-based decoding and correlation distribution-based decoding, although the correlation-based decoding is better established and should be preferred over the correlation distribution-based decoding. As such, we will only show the **CorrelationDecoder** here.

CorrelationDecoder currently runs *very* slowly. We strongly recommend running it on a subset of labels within the Dataset. It is also quite memory-intensive.

In this example, we have only run the decoder using features appearing in >10% and <90% of the first 500 studies in the Dataset. Additionally, we have pregenerated the results and will simply show the code that *would* generate those results, as the decoder requires too much memory for NeuroLibre's servers.

```
from nimare import decode, meta

corr_decoder = decode.continuous.CorrelationDecoder(
            frequency_threshold=0.001,
            meta_estimator=meta.MKDADensity(kernel_
            transformer=kern, memory_limit=None),
            target_image="z",
            features=target_features,
            memory_limit="500mb",
)
corr_decoder.fit(neurosynth_dset_first500)
corr_df = corr_decoder.transform(continuous_map)
```

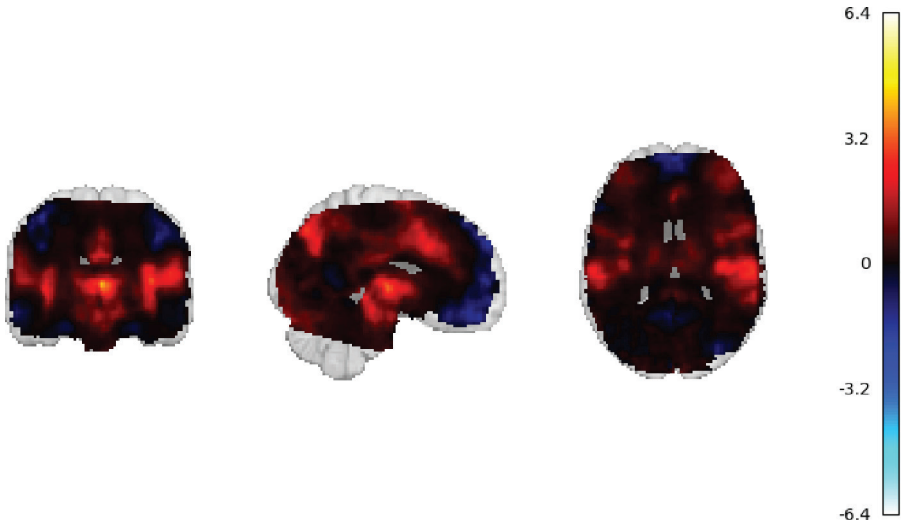

Fig. 16. The unthresholded statistical map that will be used for continuous decoding.

 CC By 4.0: © Taylor Salo et al.

feature	r
terms_abstract_tfidf__cingulate	0.448665
terms_abstract_tfidf__anterior cingulate	0.415396
terms_abstract_tfidf__anterior	0.391049
terms_abstract_tfidf__auditory	0.343746
terms_abstract_tfidf__conditions	0.338773
terms_abstract_tfidf__cortices	0.331308
terms_abstract_tfidf__information	0.330473
terms_abstract_tfidf__role	0.326022
terms_abstract_tfidf__level	0.322515
terms_abstract_tfidf__healthy	0.319750

Fig. 17. The top 10 terms, sorted by absolute correlation coefficient, from the correlation decoding method.

```
import pandas as pd

corr_df = pd.read_table(
    os.path.join(data_path, "correlation_decoder_results.tsv"),
    index_col="feature",
)
```

Decoding discrete inputs

Decoding regions of interest (ROIs) requires a different approach than decoding unthresholded statistical maps. One simple approach, used by GCLDA and implemented in the function **gclda_decode_roi()**, simply sums the P(topic|voxel) distribution across all voxels in the ROI in order to produce a value associated with each topic for the ROI. These weight sum values are arbitrarily scaled and cannot be compared across ROIs. We will not show this method because of its simplicity and the fact that it can only currently be applied to a GCLDA model.

Before we dig into the other decoding methods are available, let's take a look at the ROI we want to decode.

One method which relies on correlations, much like the continuous correlation decoder, is the ROI association decoding method **(ROIAssociationDecoder)**, originally implemented in the Neurosynth Python library. In this method, each study with coordinates in the dataset is convolved with a Kernel transformer to produce a modeled activation map. The resulting modeled activation maps are then masked with a region of interest (i.e., the target of the decoding), and the values are averaged within the ROI. These averaged modeled activation values are then correlated with the term weights for all labels in the dataset. This decoding method produces a single correlation coefficient for each of the dataset's labels.

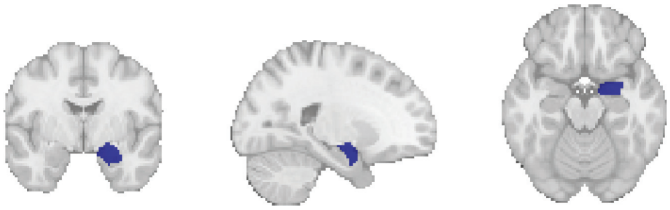

Fig. 18. The amygdala region of interest mask that will be used for discrete decoding.

feature	r
terms_abstract_tfidf__amygdala	0.627186
terms_abstract_tfidf__fear	0.334112
terms_abstract_tfidf__reinforcement	0.329516
terms_abstract_tfidf__neutral faces	0.320801
terms_abstract_tfidf__appraisal	0.317214
terms_abstract_tfidf__conditioning	0.286165
terms_abstract_tfidf__age sex	0.273753
terms_abstract_tfidf__neutral	0.272666
terms_abstract_tfidf__amygdala response	0.267777
terms_abstract_tfidf__olfactory	0.265687

Fig. 19. The top 10 terms, sorted by absolute correlation coefficient, from the ROI association decoding method.

Because the ROIAssociationDecoder generates modeled activation maps for all of the experiments in the Dataset, we will only fit this decoder to the first 500 studies.

```
from nimare import decode

assoc_decoder = decode.discrete.ROIAssociationDecoder(
    amygdala_roi, kernel_transformer=kern, u=0.05,
    correction="fdr_bh",
)
assoc_decoder.fit(neurosynth_dset_first500)
assoc_df = assoc_decoder.transform()
```

```
INFO:nimare.base:Retaining 2941/(3228 features.
```

A more theoretically driven approach to ROI decoding is to use Chi-square-based methods. The two methods that use Chi-squared tests are the BrainMap decoding method and an adaptation of Neurosynth's meta-analysis method.

In both Chi-square-based methods, studies are first selected from a coordinate-based database according to some criterion. For example, if decoding a region of interest, users might select studies reporting at least

 CC By 4.0: © Taylor Salo et al.

one coordinate within 5 mm of the ROI. Metadata (such as ontological labels) for this subset of studies are then compared with those of the remaining, unselected portion of the database in a confusion matrix. For each label in the ontology, studies are divided into four groups: selected and label-positive (SS+L+), selected and label-negative (SS+L −), unselected and label-positive (SS-L+), and unselected and label-negative (SS-L−). Each method then compares these groups in order to evaluate both consistency and specificity of the relationship between the selection criteria and each label, which are evaluated in terms of both statistical significance and effect size.

BrainMap method

The BrainMap discrete decoding method, implemented in **BrainMapDecoder**, compares the distributions of studies with each label within the sample against those in a larger database while accounting for the number of foci from each study. Broadly speaking, this method assumes that the selection criterion is associated with one peak per study, which means that it is likely only appropriate for selection criteria based around foci, such as ROIs. One common analysis, meta-analytic clustering, involves dividing studies within a database into meta-analytic groupings based on the spatial similarity of their modeled activation maps (i.e., study-wise pseudostatistical maps produced by convolving coordinates with a Kernel). The resulting sets of studies are often functionally decoded in order to build a functional profile associated with each meta-analytic grouping. While these groupings are defined as subsets of the database, they are not selected based on the location of an individual peak, and so weighting based on the number of foci would be inappropriate.

This decoding method produces four outputs for each label. First, the distribution of studies in the sample with the label are compared with the distributions of other labels within the sample. This consistency analysis produces both a measure of statistical significance (i.e., a *P* value) and a measure of effect size (i.e., the likelihood of being selected given the presence of the label). Next, the studies in the sample are compared with the studies in the rest of the database. This specificity analysis produces a *P* value and an effect size measure of the posterior probability of having the label given selection into the sample. A detailed algorithm description is presented in Appendix I: BrainMap Discrete Decoding.

```
brainmap_decoder = decode.discrete.BrainMapDecoder(
    frequency_threshold=0.001,
    u=0.05,
    correction="fdr_bh",
)
brainmap_decoder.fit(neurosynth_dset)
brainmap_df = brainmap_decoder.transform(amygdala_ids)
```

Neurosynth method

The implementation of the MKDA Chi-squared meta-analysis method used by Neurosynth is quite similar to BrainMap's method for decoding, if applied to annotations instead of modeled activation values. This method, implemented in **NeurosynthDecoder**, compares the distributions of studies with each label within the sample against those in a larger database, but, unlike the BrainMap method, does not take foci into account. For this reason, the Neurosynth method would likely be more appropriate for selection criteria not based on ROIs (e.g., for characterizing meta-analytic groupings

Term	pForward	zForward	likelihoodForward	pRe
terms_abstract_tfidf__magnetic	1.000000e+00	0.000000	1.627342	1.000000
terms_abstract_tfidf__magnetic resonance	1.000000e+00	0.000000	1.569942	1.000000
terms_abstract_tfidf__resonance	1.000000e+00	0.000000	1.560855	1.000000
terms_abstract_tfidf__amygdala	6.176744e-07	4.985607	6.407830	2.15210
terms_abstract_tfidf__functional magnetic	1.000000e+00	0.000000	1.664782	1.000000
terms_abstract_tfidf__using	1.000000e+00	0.000000	1.205627	1.000000
terms_abstract_tfidf__response	1.000000e+00	0.000000	1.902148	1.000000
terms_abstract_tfidf__stimuli	1.000000e+00	0.000000	2.087218	1.000000
terms_abstract_tfidf__human	1.000000e+00	0.000000	2.128483	1.000000
terms_abstract_tfidf__neutral	5.556739e-03	2.772852	5.502956	2.753866

Fig. 20. The top 10 terms, sorted by reverse-inference posterior probability, from the BrainMap Chi-squared decoding method.

 CC By 4.0: © Taylor Salo et al.

Term	pForward	zForward	probForward	pReverse
terms_abstract_tfidf__neutral faces	6.163179e-12	6.875826	0.024702	4.685748e-31
terms_abstract_tfidf__conditioning	1.026669e-16	8.301660	0.022543	1.221816e-32
terms_abstract_tfidf__olfactory	1.482800e-02	2.436553	0.021075	2.058612e-12
terms_abstract_tfidf__conditioned	7.813282e-05	3.950056	0.017439	2.697558e-13
terms_abstract_tfidf__reinforcement	7.813282e-05	3.950056	0.016397	2.058612e-12
terms_abstract_tfidf__unpleasant	7.813282e-05	3.950056	0.015924	4.616361e-12
terms_abstract_tfidf__amygdala	1.953150e-123	23.628732	0.006013	2.152100e-18
terms_abstract_tfidf__amygdala response	1.482800e-02	2.436553	0.014081	1.526484e-07
terms_abstract_tfidf__differentiated	1.482800e-02	2.436553	0.014081	1.526484e-07
terms_abstract_tfidf__pleasant	1.482800e-02	2.436553	0.013307	5.206710e-07

Fig. 21. The top 10 terms, sorted by reverse-inference posterior probability, from the Neurosynth Chi-squared decoding method.

from a meta-analytic clustering analysis). However, the Neurosynth method requires user-provided information that BrainMap does not. Namely, in order to estimate probabilities for the consistency and specificity analyses with Bayes' Theorem, the Neurosynth method requires a prior probability of a given label. Typically, a value of 0.5 is used (i.e., the estimated probability that an individual is undergoing a given mental process described by a label, barring any evidence from neuroimaging data, is predicted to be 50%). This is, admittedly, a poor prediction, which means that probabilities estimated based on this prior are not likely to be accurate, though they may still serve as useful estimates of effect size for the analysis.

Like the BrainMap method, this method produces four outputs for each label. For the consistency analysis, this method produces both a *P* value and a conditional probability of selection given the presence of the label and the prior probability of having the label. For the specificity analysis, the Neurosynth method produces both a *P* value and a posterior probability of presence of the label given selection and the prior probability of having the label. A detailed algorithm description is presented in Appendix II: Neurosynth Discrete Decoding.

```
neurosynth_decoder = decode.discrete.NeurosynthDecoder(
    frequency_threshold=0.001,
    u=0.05,
    correction="fdr_bh",
)
neurosynth_decoder.fit(neurosynth_dset)
neurosynth_df = neurosynth_decoder.transform(amygdala_ids)
```

In both methods, the database acts as an estimate of the underlying distribution of labels in the real world, such that the probability of having a peak in an ROI given the presence of the label might be interpreted as the probability of a brain activating a specific brain region given that the individual is experiencing a given mental state. This is a very poor interpretation, given that any database of neuroimaging results will be skewed more toward the interests of the field than the distribution of mental states or processes experienced by humans, which is why decoding must be interpreted with extreme caution. It is important not to place too much emphasis on the results of functional decoding analyses, although they are very useful in that they can provide a quantitative estimate behind the kinds of interpretations generally included in discussion sections that are normally only backed by informal literature searches or prior knowledge.

The meta-analytic functional decoding methods in NiMARE provide a very rudimentary approach for open-ended decoding (i.e., decoding across a very large range of mental states) that can be used with resources like NeuroVault. However, standard classification methods have also been applied to datasets from NeuroVault (e.g.[75]), although these methods do not fall under NiMARE's scope.

FUTURE DIRECTIONS

NiMARE's mission statement encompasses a range of tools that have not yet been implemented in the package. In the future, we plan to incorporate a number of

 CC By 4.0: © Taylor Salo et al.

additional methods. Here, we briefly describe several of these tools.

Integration with external databases

A resource that may ultimately be integrated with Neurosynth is Brainspell. Brainspell is a port of the Neurosynth database in which users may manually annotate the automatically extracted study information. The goal of Brainspell is to crowdsource annotation through both expert and nonexpert annotators, which would address the primary weaknesses of BrainMap (i.e., slow growth) and Neurosynth (i.e., noise in data extraction and annotation). Annotations in Brainspell may use labels from the Cognitive Paradigm Ontology (CogPO),[14] an ontology adapted from the BrainMap Taxonomy, or from the Cognitive Atlas,[76] a collaboratively generated ontology built by contributions from experts across the field of cognitive science. Users may also correct the coordinates extracted by Neurosynth, which may suffer from extraction errors, and may add important metadata like the number of subjects associated with each comparison in each study.

Brainspell has suffered from low growth, which is why its annotations have not been integrated back into Neurosynth, but a new frontend tool for Brainspell, geared toward meta-analysts, has been developed called metaCurious. MetaCurious facilitates neuroimaging meta-analyses by allowing users to iteratively perform literature searches and to annotate rejected articles with reasons for exclusion. In addition to these features, metaCurious users can annotate studies with the same labels and metadata as Brainspell, but with the features geared toward meta-analysts site usage is expected to exceed that of Brainspell proper.

While NiMARE does not natively include tools for interacting with Brainspell or metaCurious, there are plans to support NiMARE-format exports in both services.

Seed-based D-Mapping

Seed-based d-mapping (SDM),[77] previously known as signed differential mapping, is a relatively recently developed approach designed to incorporate both peak-specific effect size estimates and unthresholded images, when available. In SDM, foci are convolved with an anisotropic Kernel which, unlike the Gaussian and spherical kernels employed in ALE and MKDA, respectively, accounts for tissue type to provide more empirically realistic spatial models of the clusters from the original studies. The SDM algorithm is not yet supported in NiMARE, given the difficulty in implementing an algorithm without access to code.

Model-based CBMA

Model-based algorithms, a recent alternative to Kernel-based approaches, model foci from studies as the products of stochastic models sampling some underlying distribution. Some of these methods include the Bayesian hierarchical independent cluster process model (BHICP),[78] the Bayesian spatially adaptive binary regression model (SBR),[79] the hierarchical Poisson/Gamma random field model (HPGRF/BHPGM),[80] the spatial Bayesian latent factor regression model (SBLFRM),[81] and the random effects log Gaussian Cox process model (RFX-LGCP).[82]

Although these methods are much more computationally intensive than Kernel-based algorithms, they provide information that Kernel-based methods cannot, such as spatial confidence intervals, effect size estimate confidence intervals, and the facilitation of reverse inference. A more thorough description of the relative strengths of model-based algorithms is presented in,[34] but these benefits, at the cost of computational efficiency, have led the authors to recommend Kernel-based methods for exploratory analysis and model-based methods for confirmatory analysis.

NiMARE does not currently implement any model-based CBMA algorithms, although there are plans to include at least one in the future.

Additional automated annotation methods

Several papers have used article text to automatically annotate meta-analytic databases with a range of methods. Alhazmi et al.[83] used a combination of correspondence analysis and clustering to identify subdomains in the cognitive neuroscience literature from Neurosynth text. Monti et al.[31] generated word and document embeddings in vector space from Neurosynth abstracts using deep Boltzmann machines, which allowed them to cluster words based on semantic similarity or to describe Neurosynth articles in terms of these word clusters. Nunes[74] used article abstracts from Neurosynth to represent documents as dense vectors as well. These document vectors were then used in conjunction with corresponding coordinates to cluster words into categories, essentially annotating Neurosynth articles according to a new "ontology" based on both abstract text and coordinates.

Meta-analytic databases may also be used in conjunction with existing ontologies in order to redefine mental states or to refine the ontology. For example, Yeo et al.[84] used the Author-Topic model to identify connections between paradigm classes (i.e., tasks) and behavioral domains (i.e., mental states) from the BrainMap Taxonomy using the BrainMap database. Other examples include using meta-analytic clustering, combined with functional decoding, to identify groups of terms/labels that co-occur in neuroimaging data, in order to determine if the divisions currently employed in existing ontologies accurately reflect how mental states are separated in the mind (e.g.[85–87]).

 CC By 4.0: © Taylor Salo et al.

Surface-based meta-analysis

Currently, NiMARE only supports volumetric meta-analysis. However, we eventually plan to support surface-based meta-analyses, which may require new coordinate-based meta-analysis algorithms, as the current methods do not generalize to surfaces.

SUMMARY

The advent of open, large-scale databases of neuroimaging results, whether full, unthresholded statistical maps, or simple coordinates, has allowed for the development of a wide variety of methods for performing fMRI meta-analyses and related analyses. These methods are often (but not always) released as tools for the community to use, written in a range of languages and with highly variable interfaces. As a consequence, it is difficult for meta-analysts to keep abreast of the current literature and to employ whatever method is most appropriate to address a given question. NiMARE provides a centralized repository for these tools, which will make it easier for researchers to keep track of new methods and also provides said tools with extensive documentation and a standardized programmatic interface, which will allow researchers to use whatever tool is most appropriate for their research, without unnecessarily steep learning curves.

Given that NiMARE is open source and collaboratively developed on GitHub, methodologists may contribute their own meta-analytic algorithms directly, or interested third parties may implement these algorithms using papers or external tools as a basis for understanding the methods.

Acknowledgments

We would like to thank Yifan Yu and Jérôme Dockès, who provided feedback on the article.

We would also like to thank the NeuroLibre team, including Loic Tetrel, Agah Karakuzu, Elizabeth DuPre, Samir Das, Nikola Stikov, and Pierre Bellec, for their help and support in reviewing and publishing the preprint form of this article.

We would also like to thank Fa Meilong for peer reviewing this article for Aperture.

Funding

Primary funding for this project was provided by NSF 1631325 and NIH R01-DA041353. Contributions from co-authors were provided with support from NIH R01MH096906 (Neurosynth), NIH P41-EB019936 (ReproNim), NIH R01-MH083320 (CANDIShare), NIH RF1-MH120021 (NIDM), as well as the Canada First Research Excellence Fund, awarded to McGill University for the Healthy Brains for Healthy Lives initiative and the Brain Canada Foundation with support from Health Canada.

Code and data availability

All code used for this article is available at https://github.com/NBCLab/nimare-paper.

All data used for this article is available at https://drive.google.com/uc?id=1e5KqMjYbQZYBxc6z760VhdruOoywqkbw.

REFERENCES

1. Salo T, Yarkoni T, Nichols TE, et al. Neurostuff/nimare: 0.0.12rc1. February 2022. https://doi.org/10.5281/zenodo.6091632.
2. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: machine learning in Python. J Mach Learn Res. 2011;12:2825–30.
3. Buitinck L, Louppe G, Blondel M, et al. API design for machine learning software: experiences from the scikit-learn project. In ECML PKDD workshop: languages for data mining and machine learning. 2013. p. 108–22.
4. Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat Methods. 2020;17(3):261–72.
5. van der Walt S, Colbert SC, Varoquaux G. The NumPy array: a structure for efficient numerical computation. 2011.
6. Harris CR, Millman KJ, van der Walt SJ, et al. Array programming with NumPy. Nature. 2020;585(7825):357–62.
7. McKinney W. Data structures for statistical computing in python. 2010.
8. Brett M, Markiewicz CJ, Hanke M, et al. Nipy/nibabel: 3.1.1. June 2020. https://doi.org/10.5281/zenodo.3924343.
9. Abraham A, Pedregosa F, Eickenberg M, et al. Machine learning for neuroimaging with scikit-learn. Front Neuroinform. 2014;8:14.
10. Seabold S, Perktold J. Statsmodels: econometric and statistical modeling with python. 2010.
11. da Costa-Luis C, Larroque SK, Altendorf K, et al. tqdm: A fast, extensible progress bar for Python and CLI. September 2020. https://doi.org/10.5281/zenodo.4026750.
12. Poldrack RA, Yarkoni T. From brain maps to cognitive ontologies: informatics and the search for mental structure. Annu Rev Psychol. 2016;67:587–612.
13. Poldrack A. Mapping mental function to brain structure: how can cognitive neuroimaging succeed? Perspect Psychol Sci. 2010;5(6):753–61.
14. Turner JA, Laird AR. The cognitive paradigm ontology: design and application. Neuroinformatics. 2012;10(1):57–66.
15. Fox PT, Laird AR, Fox SP, et al. BrainMap taxonomy of experimental design: description and evaluation. Hum Brain Mapp. 2005;25(1):185–98.
16. Fox PT, Lancaster JL. Opinion: mapping context and content: the BrainMap model. Nat Rev Neurosci. 2002;3(4):319–21.
17. Laird AR, Lancaster JL, Fox PT. BrainMap: the social evolution of a human brain mapping database. Neuroinformatics. 2005;3(1):65–78.
18. Yanes JA, Riedel MC, Ray KL, et al. Neuroimaging meta-analysis of cannabis use studies reveals convergent functional alterations in brain regions supporting cognitive control and reward processing. J Psychopharmacol. 2018;32(3):283–95. https://doi.org/10.1177/0269881117744995.
19. Yarkoni T, Poldrack RA, Nichols TE, Van Essen DC, Wager TD. Large-scale automated synthesis of human functional neuroimaging data. Nat Methods. 2011;8(8):665–70.
20. Chang LJ, Yarkoni T, Khaw MW, Sanfey AG. Decoding the role of the insula in human cognition: functional parcellation and large-scale reverse inference. Cereb Cortex. 2013;23(3):739–49.
21. de la Vega A, Chang LJ, Banich MT, Wager TD, Yarkoni T. Large-scale meta-analysis of human medial frontal cortex reveals tripartite functional organization. J Neurosci. 2016;36(24):6553–62.
22. de la Vega A, Yarkoni T, Wager TD, Banich MT. Large-scale meta-analysis suggests low regional modularity in lateral frontal cortex. Cereb Cortex. 2018;28(10):3414–28.
23. Poldrack RA, Mumford JA, Schonberg T, Kalar D, Barman B, Yarkoni T. Discovering relations between mind, brain, and mental disorders using topic mapping. PLoS Comput Biol. 2012;8(10):e1002707.

 CC By 4.0: © Taylor Salo et al.

24. Josipovic Z. Neural correlates of nondual awareness in meditation. *Ann N Y Acad Sci*. 2014;1307:9–18.
25. Zeidman P, Mullally SL, Schwarzkopf DS, Maguire EA. Exploring the parahippocampal cortex response to high and low spatial frequency spaces. Neuroreport. 2012;23(8):503–7.
26. Wager TD, Atlas LY, Lindquist MA, Roy M, Woo C-W, Kross E. An fMRI-based neurologic signature of physical pain. N. Engl. J. Med. 2013;368(15):1388–97.
27. Chen Y, Fowler CH, Papa VB, et al. Adolescents' behavioral and neural responses to e-cigarette advertising. Addict Biol. 2018;23(2):761–71.
28. Pantelis PC, Byrge L, Tyszka JM, Adolphs R, Kennedy DP. A specific hypoactivation of right temporo-parietal junction/posterior superior temporal sulcus in response to socially awkward situations in autism. Soc Cogn Affect Neurosci. 2015;10(10):1348–56.
29. Tambini A, Rimmele U, Phelps EA, Davachi L. Emotional brain states carry over and enhance future memory formation. Nat. Neurosci. 2017;20(2):271–8.
30. Rubin TN, Koyejo O, Gorgolewski KJ, Jones MN, Poldrack RA, Yarkoni T. Decoding brain activity using a large-scale probabilistic functional-anatomical atlas of human cognition. PLoS Comput. Biol. 2017;13(10):e1005649.
31. Monti R, Lorenz R, Leech R, Anagnostopoulos C, Montana G. Text-mining the neurosynth corpus using deep boltzmann machines. 2016.
32. Dockès J, Poldrack RA, Primet R, et al. NeuroQuery, comprehensive meta-analysis of human brain mapping. Elife. March 2020.
33. Gorgolewski KJ, Varoquaux G, Rivera G, et al. NeuroVault.org: a web-based repository for collecting and sharing unthresholded statistical maps of the human brain. Front Neuroinform. 2015;9:8.
34. Samartsidis P, Montagna S, Johnson TD, Nichols TE. The coordinate-based meta-analysis of neuroimaging data. 2017.
35. Müller VI, Cieslik EC, Laird AR, et al. Ten simple rules for neuroimaging meta-analysis. Neurosci Biobehav Rev. 2018;84:151–61.
36. Laird AR, Fox PM, Price CJ, et al. ALE meta-analysis: controlling the false discovery rate and performing statistical contrasts. Hum Brain Mapp. 2005;25(1):155–64.
37. Eickhoff SB, Bzdok D, Laird AR, Kurth F, Fox PT. Activation likelihood estimation meta-analysis revisited. NeuroImage. 2012;59(3):2349–61https://doi.org/10.1016/j.neuroimage.2011.09.017.
38. Wager TD, Lindquist M, Kaplan L. Meta-analysis of functional neuroimaging data: current and future directions. Soc. Cogn. Affect. Neurosci. 2007;2(2):150–58.
39. Wager TD, Phan KL, Liberzon I, Taylor SF. Valence, gender, and lateralization of functional brain anatomy in emotion: a meta-analysis of findings from neuroimaging. Neuroimage. 2003;19(3):513–31.
40. Wager TD, Jonides J, Reading S. Neuroimaging studies of shifting attention: a meta-analysis. Neuroimage. 2004;22(4):1679–93.
41. Eickhoff SB, Bzdok D, Laird AR, Kurth F, Fox PT. Activation likelihood estimation meta-analysis revisited. Neuroimage. 2012;59(3):2349–61.
42. Turkeltaub PE, Eickhoff SB, Laird AR, Fox M, Wiener M, Fox P. Minimizing within-experiment and within-group effects in activation likelihood estimation meta-analyses. Hum Brain Mapp. 2012;33(1):1–13.
43. Turkeltaub PE, Eden GF, Jones KM, Zeffiro TA. Meta-analysis of the functional neuroanatomy of single-word reading: method and validation. Neuroimage. 2002;16(3 Pt 1):765–80.
44. Langner R, Rottschy C, Laird AR, Fox PT, Eickhoff SB. Meta-analytic connectivity modeling revisited: controlling for activation base rates. Neuroimage. 2014;99:559–70.
45. DerSimonian R, Laird N. Meta-analysis in clinical trials. 1986.
46. Hedges LV, Hedges LV, Olkin I. Statistical methods for meta-analysis. Academic Press; 1985.
47. Sangnawakij P, Böhning D, Niwitpong S-A, Adams S, Stanton M, Holling H. Meta-analysis without study- specific variance information: heterogeneity case. Stat Methods Med Res. 2019;28(1):196–210.
48. Freedman D, Lane D. A nonstochastic interpretation of reported significance levels. 1983.
49. Anderson MJ, Robinson J. Permutation tests for linear models. 2001.
50. Winkler AM, RidgwayGR, Webster MA, Smith SM, Nichols TE. Permutation inference for the general linear model. Neuroimage. 2014;92:381–97.
51. Fisher RA. Statistical methods for research workers. Vol.102. Oliver and Boyd; 1925.
52. Riley JW, Stouffer SA, Suchman EA, Devinney LC, Star SA, Williams RM. The American soldier: adjustment during army life. 1949.
53. Zaykin DV. Optimally weighted z-test is a powerful method for combining probabilities in meta-analysis. J Evol Biol. 2011;24(8):1836–41.
54. Maumet C, Nichols TE. Minimal data needed for valid & accurate image-based fMRI meta-analysis. Preprint, 2016.
55. Laird AR, Eickhoff SB, Li K, Robin DA, Glahn DC, Fox PT. Investigating the functional heterogeneity of the default mode network using coordinate-based meta-analytic modeling. J Neurosci. 2009;29(46):14496–505.
56. Robinson JL, Laird AR, Glahn DC, Lovallo WR, Fox PT. Metaanalytic connectivity modeling: delineating the functional connectivity of the human amygdala. Hum Brain Mapp. 2010;31(2):173–84.
57. Eickhoff SB, Jbabdi S, Caspers S, et al. Anatomical and functional connectivity of cytoarchitectonic areas within the human parietal operculum. J Neurosci. 2010;30(18):6409–21.
58. Hok P, Opavský R, Hluštík P, Tüdös Z. 29. Meta-analytic and resting-state functional connectivity of the claustrum. 2015.
59. Kellermann TS, Caspers S, Fox PT, et al Task- and resting-state functional connectivity of brain regions related to affection and susceptible to concurrent cognitive demand. *Neuroimage*. 2013;72:69–82.
60. Reid AT, Hoffstaedter F, Gong G, et al. A seed-based cross-modal comparison of brain connectivity measures. Brain Struct Funct. 2017;222(3):1131–51.
61. Riedel MC, Ray KL, Dick AS, et al. Meta-analytic connectivity and behavioral parcellation of the human cerebellum. Neuroimage. 2015;117:327–42.
62. Reid AT, Bzdok D, Langner R, et al. Multimodal connectivity mapping of the human left anterior and posterior lateral prefrontal cortex. Brain Struct Funct. 2016;221(5):2589–605.
63. Caspers J, Zilles K, Amunts K, Laird AR, Fox PT, Eickhoff SB. Functional characterization and differential coactivation patterns of two cytoarchitectonic visual areas on the human posterior fusiform gyrus. Hum Brain Mapp. 2014;35(6):2754–67.
64. Blei DM, Ng AY, Jordan MI. Latent dirichlet allocation. J Mach Learn Res. 2003;3:993–1022.
65. Smith SM, Fox PT, Miller KL, et al. Correspondence of the brain's functional architecture during activation and rest. 2009.
66. Poldrack RA. Inferring mental states from neuroimaging data: from reverse inference to large-scale decoding. Neuron. 2011;72(5):692–7.
67. Bzdok D, Laird AR, Zilles K, Fox PT, Eickhoff SB. An investigation of the structural, connectional, and functional subspecialization in the human amygdala. Hum Brain Mapp. 2013;34(12):3247–66.
68. Cieslik EC, Zilles K, Caspers S, et al. Is there "one" DLPFC in cognitive action control? Evidence for heterogeneity from Co-Activation-Based parcellation. 2013.
69. Rottschy C, Caspers S, Roski C, et al. Differentiated parietal connectivity of frontal regions for "what" and "where" memory. 2013.
70. Amft M, Bzdok D, Laird AR, Fox PT, Schilbach L, Eickhoff SB. Definition and characterization of an extended social-affective default network. Brain Struct Funct. 2015;220(2):1031–49.
71. Bzdok D, Langner R, Schilbach L, et al. Characterization of the temporo-parietal junction by combining data-driven parcellation, complementary connectivity analyses, and functional decoding. Neuroimage. 2013;81:381–92.
72. Nickl-Jockschat T, Rottschy C, Thommes J, et al. Neural networks related to dysfunctional face processing in autism spectrum disorder. Brain Struct. Funct. 2015;220(4):2355–71.
73. Dockès J, Wassermann D, Poldrack R, Suchanek F, Thirion B, Varoquaux G. Text to brain: predicting the spatial distribution of neuroimaging observations from text reports. 2018.
74. Nunes A. Word2brain. *bioRxiv*, 2018. https://www.biorxiv.org/content/early/2018/04/11/299024.
75. Varoquaux G, Schwartz Y, Poldrack RA, et al. Atlases of cognition with large-scale human brain mapping. PLoS Comput Biol. 2018;14(11):e1006565.
76. Poldrack RA, Kittur A, Kalar D, et al. The Cognitive Atlas: toward a knowledge foundation for cognitive neuroscience. 2011.
77. Radua J, Mataix-Cols D, Phillips ML,. A new meta-analytic method for neuroimaging studies that combines reported peak coordinates and statistical parametric maps. Eur Psychiatry. 2012;27(8):605–11.
78. Kang J, Johnson TD, Nichols TE, Wager TD. Meta analysis of functional neuroimaging data via Bayesian spatial point processes. J Am Stat Assoc. 2011;106(493):124–34.
79. Yue YR, Lindquist MA, Loh JM. Meta-analysis of functional neuroimaging data using Bayesian nonparametric binary regression. 2012.
80. Kang J, Nichols TE, Wager TD, Johnson TD. A Bayesian hierarchical spatial point process model for multi- type neuroimaging meta-analysis. Ann Appl Stat. 2014;8(3):1800–24.
81. Montagna S, Wager T, Barrett LF, Johnson TD, Nichols TE. Spatial bayesian latent factor regression modeling of coordinate-based meta-analysis data. *Biometrics*. 2018;74(1):342–53.
82. Samartsidis P, Eickhoff CR, Eickhoff SB, et al.. Bayesian log-gaussian cox process regression: with applications to meta-analysis of neuroimaging working memory studies. J R Stat Soc Ser C Appl Stat. 2019;68(1):217–34.

 CC By 4.0: © Taylor Salo et al.

83. Alhazmi FH, Beaton D, Abdi H. Semantically defined subdomains of functional neuroimaging literature and their corresponding brain regions. Hum Brain Mapp. 2018;39(7):2764–76.
84. Yeo BTT, Krienen FM, Eickhoff SB, et al. Functional specialization and flexibility in human association cortex. Cereb Cortex. 2016;26(1):465
85. Laird AR, Riedel MC, Sutherland MT, et al. Neural architecture underlying classification of face perception paradigms. Neuroimage. 2015;119:70–80.
86. Riedel MC, Yanes JA, RayKL, et al. Dissociable meta- analytic brain networks contribute to coordinated emotional processing. Hum Brain Mapp. 2018;39(6):2514–31.
87. Bottenhorn KL, Flannery JS, Boeving ER, et al. Cooperating yet distinct brain networks engaged during naturalistic paradigms: a meta-analysis of functional MRI results. Netw Neurosci. 2019;3(1):27–48.

Appendix I: BrainMap discrete decoding

The BrainMap discrete decoding method compares the distributions of studies with each label within the sample against those in a larger database while accounting for the number of foci from each study. Broadly speaking, this method assumes that the selection criterion is associated with one peak per study, which means that it is likely only appropriate for selection criteria based around foci, such as ROIs. One common analysis, meta-analytic clustering, involves dividing studies within a database into meta-analytic groupings based on the spatial similarity of their modeled activation maps (i.e., study-wise pseudo-statistical maps produced by convolving coordinates with a Kernel). The resulting sets of studies are often functionally decoded in order to build a functional profile associated with each meta-analytic grouping. While these groupings are defined as subsets of the database, they are not selected based on the location of an individual peak, and so weighting based on the number of foci would be inappropriate.

This decoding method produces four outputs for each label. First, the distribution of studies in the sample with the label are compared with the distributions of other labels within the sample. This consistency analysis produces both a measure of statistical significance (i.e., a *P* value) and a measure of effect size (i.e., the likelihood of being selected given the presence of the label). Next, the studies in the sample are compared to the studies in the rest of the database. This specificity analysis produces a *P* value and an effect size measure of the posterior probability of having the label given selection into the sample. A detailed algorithm description is presented below.

The BrainMap method for discrete functional decoding performs both forward and reverse inference using an annotated coordinate-based database and a target sample of studies within that database. Unlike the Neurosynth approach, the BrainMap approach incorporates information about the number of foci associated with each study in the database.

1. Select studies in the database according to some criterion (e.g., having at least one peak in an ROI).
2. For each label, studies in the database can now be divided into four groups.
   - Label-positive and selected : S_{s+l+}
   - Label-negative and selected : S_{s+l-}
   - Label-positive and unselected : S_{s-l+}
   - Label-negative and unselected : S_{s-l-}
3. Additionally, the number of foci associated with each of these groups is extracted.
   - Number of foci from studies with label, F_{l+}
   - Number of foci from studies without label, F_{l-}
   - Total number of foci in the database, $F_{db} = F_{l+} + F_{l-}$
4. Compute the number of times any label is used in the database, L_{db} (e.g., if every experiment in the database uses two labels, then this number is $2S_{db}$, where S_{db} is the total number of experiments in the database).
5. Compute the probability of being selected, $P(s^+)$.
   - $P(s^+) = S_{s+}/F_{db}$, *where* $S_{s+} = S_{s+l+} + S_{s+l-}$
6. For each label, compute the probability of having the label, $P(l^+)$.
   - $P(l^+) = S_{l+}/L_{db}$, *where* $S_{l+} = S_{s+l+} + S_{s-l+}$
7. For each label, compute the probability of being selected given presence of the label, $P(s^+|l^+)$.
   - Can be re-interpreted as the probability of activating the ROI given a mental state.
   - $P(s^+|l^+) = S_{s+l+}/F_{l+}$
8. Convert $P(s^+|l^+)$ into the forward inference likelihood, *L*.
   - $L = P(s^+|l^+)/P(s^+)$
9. Compute the probability of the label given selection, $P(l^+|s^+)$.
   - Can be reinterpreted as probability of a mental state given activation of the ROI.
   - $$P(l^+|s^+) = \frac{P(s^+ \mid l^+)P(l^+)}{P(s^+)}$$
   - This is the reverse inference posterior probability.

 CC By 4.0: © Taylor Salo et al.

10. Perform a binomial test to determine if the rate at which studies are selected from the set of studies with the label is significantly different from the base probability of studies being selected across the whole database.
    - The number of successes is $K = S_{s+l+}$, the number of trials is $n = F_{l+}$, and the hypothesized probability of success is $p = P(s^+)$
    - If $S_{s+l+} < 5$, override the P value from this test with 1, essentially ignoring this label in the analysis.
    - Convert P value to unsigned z value.
11. Perform a two-way Chi-square test to determine if presence of the label and selection are independent.
    - If $S_{s+l+} < 5$, override the P value from this test with 1, essentially ignoring this label in the analysis.
    - Convert P value to unsigned z value.

Appendix II: Neurosynth discrete decoding

The implementation of the MKDA Chi-squared meta-analysis method used by Neurosynth is quite similar to BrainMap's method for decoding, if applied to annotations instead of modeled activation values. This method compares the distributions of studies with each label within the sample against those in a larger database, but, unlike the BrainMap method, does not take foci into account. For this reason, the Neurosynth method would likely be more appropriate for selection criteria not based on ROIs (e.g., for characterizing meta-analytic groupings from a meta-analytic clustering analysis). However, the Neurosynth method requires user-provided information that BrainMap does not. Namely, in order to estimate probabilities for the consistency and specificity analyses with Bayes' Theorem, the Neurosynth method requires a prior probability of a given label. Typically, a value of 0.5 is used (i.e., the estimated probability that an individual is undergoing a given mental process described by a label, barring any evidence from neuroimaging data, is predicted to be 50%). This is, admittedly, a poor prediction, which means that probabilities estimated based on this prior are not likely to be accurate, although they may still serve as useful estimates of effect size for the analysis.

Like the BrainMap method, this method produces four outputs for each label. For the consistency analysis, this method produces both a P value and a conditional probability of selection given the presence of the label and the prior probability of having the label. For the specificity analysis, the Neurosynth method produces both a P value and a posterior probability of presence of the label given selection and the prior probability of having the label. A detailed algorithm description is presented below.

The Neurosynth method for discrete functional decoding performs both forward and reverse inference using an annotated coordinate-based database and a target sample of studies within that database. Unlike the BrainMap approach, the Neurosynth approach uses an *a priori* value as the prior probability of any given experiment including a given label.

1. Select studies in the database according to some criterion (e.g., having at least one peak in an ROI).
2. For each label, studies in the database can now be divided into four groups:
    - Label-positive and selected : S_{s+l+}
    - Label-negative and selected : S_{s+l-}
    - Label-positive and unselected : S_{s-l+}
    - Label-negative and unselected : S_{s-l-}
3. Set a prior probability p of a given mental state occurring in the real world.
    - Neurosynth uses 0.5 as the default.
4. Compute $P(s^+)$:
    - Probability of being selected, $P(s^+) = S_{s+}/(S_{s+} + S_{s-})$, where $S_{s+} = S_{s+l+} + S_{s+l-}$ and $S_{s-} = S_{s-l+} + S_{s-l-}$
5. For each label, compute $P(l^+)$:
    - $P(l^+) = S_{l+}/(S_{l+} + S_{l-})$, where $S_{l+} = S_{s+l+} + S_{s-l+}$ and $S_{l-} = S_{s+l-} + S_{s-l-}$
6. Compute $P(s^+|l^+)$:
    - $P(s^+|l^+) = S_{s+l+}/S_{l+}$
7. Compute $P(s^+|l^-)$:
    - $P(s^+|l^-) = S_{s+l-}/S_{l-}$
    - Only used to determine sign of reverse inference z value.
8. Compute $P(s^+|l^+, p)$, where is the prior probability of a label:
    - This is the forward inference posterior probability. Probability of selection given label and given prior probability of label, p.
    - $P(s^+|l^+, p) = p\ P(s^+|l^+) + (1-p)\ P(s^+|l^-)$

 CC By 4.0: © Taylor Salo et al.

9. Compute $P(l^+|s^+, p)$:
   - This is the reverse inference posterior probability. Probability of label given selection and given the prior probability of label.
   - $P(l^+|s^+, p) = p\ P(s^+|l^+)/P(s^+|l^+, p)$
10. Perform a one-way Chi-squared test to determine if the rate at which studies are selected for a given label is significantly different from the average rate at which studies are selected across labels.
    - Convert P value to signed z value using whether the number of studies selected for the label is greater than or less than the mean number of studies selected across labels to determine the sign.
11. Perform a two-way Chi-square test to determine if presence of the label and selection are independent.
    - Convert P value to signed z value using $P(s^+|l^-)$ to determine sign.

By Taylor Salo, Tal Yarkoni, Thomas E. Nichols, Jean-Baptiste Poline, Murat Bilgel, Katherine L. Bottenhorn, Simon B. Eickhoff, Dorota Jarecka, James D. Kent, Adam Kimbler, Dylan M. Nielson, Kendra M. Oudyk, Julio A. Peraza, Alexandre Pérez, Puck C. Reeders, Julio A. Yanes, and Angela R. Laird

© Copyright 2021.