Chapter 14

Normative Modeling with the Predictive Clinical Neuroscience Toolkit (PCNtoolkit)

Saige Rutherford and Andre F. Marquand

Abstract

In this chapter, we introduce normative modeling as a tool for mapping variation across large neuroimaging datasets. We provide practical guidance to illustrate how normative models can be used to map diverse patterns of individual differences found within the large datasets used to train the models. In other words, while normative modeling is a method often applied to big datasets containing thousands of subjects, it provides single subject inference and prediction. We use an open-source Python package, Predictive Clinical Neuroscience Toolkit (PCNtoolkit) and showcase several helpful tools (including an interface that does not require coding) to run a normative modeling analysis, evaluate the model fit, and visualize the results.

Key words Neuroimaging, normative modeling, PCNtoolkit, individual differences

1 Introduction

Normative modeling in neuroimaging refers to the statistical analysis of brain imaging data from a large sample of individuals in order to establish typical patterns of brain structure and function [2–7]. The overarching aim is to define a reference range for the given brain measurement (structure or function) in a certain sample and to create a reference standard against which to compare individuals, often with neurological or psychiatric conditions [8–17]. This is typically achieved by fitting flexible probabilistic regression models [See Glossary] to map centiles of variation in the population, akin to the use of growth charts in pediatric medicine (see example in Fig. 1). Owing to their ability to make predictions at the level of the individual participants, normative models can be used to detect subtle changes in brain structure or function that may indicate the early stages of a disease or to evaluate the effects of a certain treatment or intervention. These models can also be used

Robert Whelan and Hervé Lemaître (eds.), *Methods for Analyzing Large Neuroimaging Datasets*, Neuromethods, vol. 218, https://doi.org/10.1007/978-1-0716-4260-3_14,

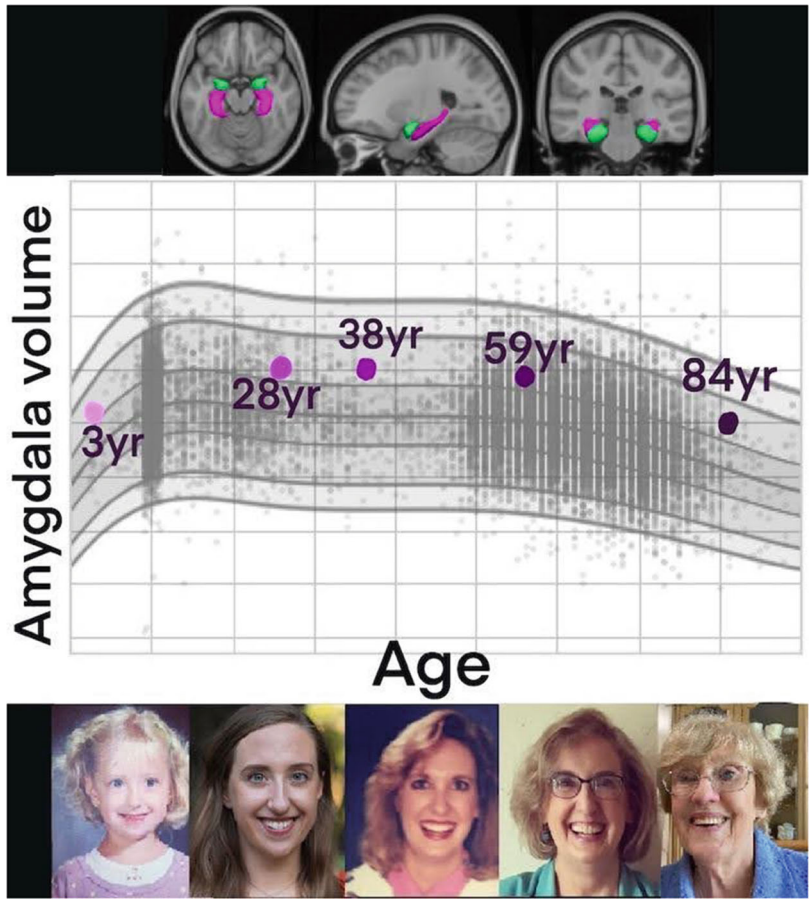


Fig. 1 Big data ($N = \sim$58,000 subjects) normative model of amygdala volume across the human lifespan (ages two to 100) [1]

in clinical settings to evaluate brain function of individuals with suspected neurological or psychiatric disorders and to monitor the progression of the disorder over time.

2 Predictive Clinical Neuroscience Toolkit (PCNToolkit)

The Predictive Clinical Neuroscience (PCN) toolkit [18] is a Python package designed for multi-purpose tasks in clinical neuroimaging, including normative modelling, trend surface modelling in addition to providing implementations of a number of fundamental machine learning algorithms [See Glossary].

Normative modelling essentially aims to predict centiles of variance in a response variable (e.g., a region of interest or other neuroimaging-derived measure) on the basis of a set of covariates (e.g., age, clinical scores, diagnosis). In this paper, we take an applied perspective and provide guidance about how to perform

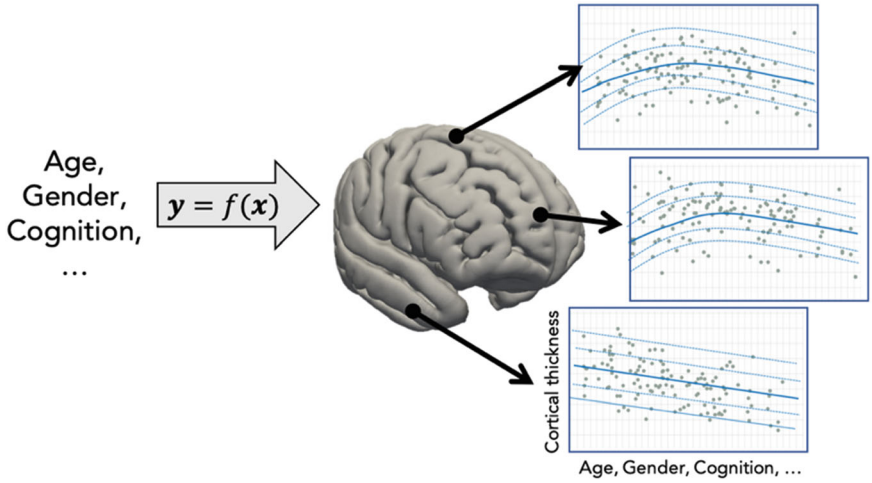


Fig. 2 Normative modeling example

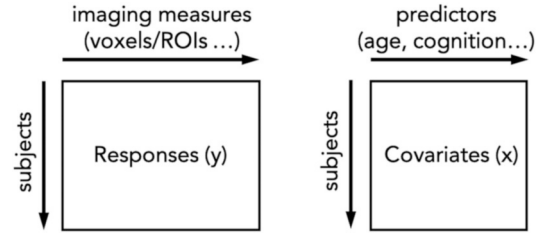


Fig. 3 Matrix representation of biological response variables and covariates

normative modelling in practice. We refer the reader to other review and protocol papers where in-depth conceptual and theoretical overviews of the approach can be found [19–21]. For example, the image below shows an example of a normative model that aims to predict vertex-wise cortical thickness data, essentially fitting a separate model for each vertex (Fig. 2).

In practice, normative modelling is done by regressing the biological response variables against a set of clinical or demographic covariates. In the instructions that follow, it is helpful to think of these as being stored in matrices as shown below (Fig. 3).

There are many options for this, but techniques that provide a distributional form for the centiles are appealing, since they help to estimate extreme centiles (where data are sparsest) more efficiently and can help to avoid centile crossings [22]. Bayesian methods are also beneficial in this regard because they also allow separation of modelling uncertainty from variation in the data. Many applications of normative modelling use Gaussian Process Regression [See Glossary], which is the default method in this toolkit. However, other

Table 1
Overview of available open-source resources for normative modeling

Resource	Description	Link
PCNportal	No code required interface for accessing pre-trained normative models	https://pcnportal.dccn.nl/
Gitter	Website for communication with PCNtoolkit developers	https://gitter.im/predictive-clinical-neuroscience/community
Read The Docs	Wiki resource page for the PCNtoolkit	https://pcntoolkit.readthedocs.io/en/latest/
GitHub	Code base for the PCNtoolkit. Contributions welcome!	https://github.com/amarquand/PCNtoolkit
Google Colab	Run python notebooks in a web browser without setup of python environment	Bayesian Linear Regression Hierarchical Bayesian regression

algorithms are available and scale better to estimating normative models on large datasets. These algorithms include Bayesian Linear Regression (BLR) and Hierarchical Bayesian Linear Regression (HBR). In the code tutorials included with this chapter, we implement normative models using BLR. Typically, each response variable (brain region) is estimated independently. In the sections that follow, we provide code tutorials for running a normative modeling analysis, with specific explanations of the modeling choices (these explanations are embedded in the relevant tutorials, which are available online via the links summarized in Table 1 and Fig. 4).

3 Code Tutorial 1: Transferring Pre-trained Big Data Normative Models

This code shows how to apply the coefficients from pre-estimated normative models to new data, such as regional cortical thickness from Freesurfer preprocessing. This can be done in two different ways: (1) using a new set of data derived from the same sites used to estimate the model and (2) on a completely different set of sites. In the latter case, we also need to estimate the site effect, which requires some calibration/adaptation data. As an illustrative example, we use a dataset derived from several OpenNeuro datasets and adapt the learned model to make predictions on these data [see Chapter 2]. This code can be run in your web browser using Google Colab here.

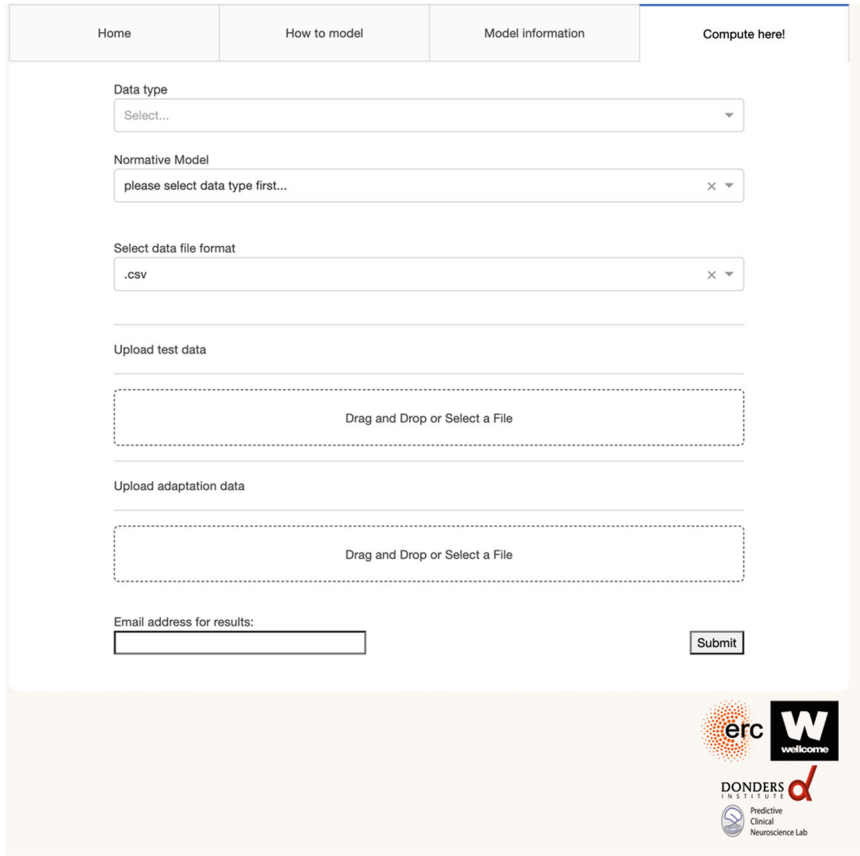


Fig. 4 Overview of the (no code required) PCNportal website for running normative modeling analysis

3.1 Using Lifespan Models to Make Predictions on New Data

3.1.1 The First Step Is to Install PCNtoolkit

```
#!/usr/bin/env python

get_ipython().system(' pip install pcntoolkit==0.28')

get_ipython().system(' git clone https://github.com/predictive-clinical-
neuroscience/braincharts.git')
```

3.1.2 Import Necessary Python Libraries

Next, the necessary libraries need to be imported. You need to be in the scripts folder when you import the libraries in the code block below, because there is a function specific to normative modeling—called *nm_utils*—that is in the scripts folder that we need to import.

```
import os

os.chdir('/content/braincharts/scripts/') # this path is setup for running on Google Colab.

Change it to match your local path if running locally



# Now we import the required libraries

import numpy as np

import pandas as pd

import pickle

from matplotlib import pyplot as plt

import seaborn as sns

from pcntoolkit.normative import estimate, predict, evaluate

from pcntoolkit.util.utils import compute_MSLL, create_design_matrix

from nm_utils import remove_bad_subjects, load_2d
```

3.1.3 Select and Unzip Model Folder

In this step, you will first unzip the models. You start by changing the directory. In this example, you will use the biggest sample as your training set (approx. $N = 58{,}000$ subjects from 82 sites). For more info on the other pre-trained models available in this repository, please refer to the accompanying paper [1].

```
# change the directory


os.chdir('/content/braincharts/models/')



#unzip the data

get_ipython().system(' unzip lifespan_57K_82sites.zip')
```

3.1.4 Set Paths and Directory Names

Next, you need to configure some basic variables, like where we want the analysis to be done (e.g., in a particular folder on your computer) and which lifespan model you want to use.

> **Note**
> We maintain a list of site IDs for each dataset, which describe the site names in the training and test data ('site_ids_tr' and 'site_ids_te'), plus also the adaptation data. The training site IDs are provided as a text file in the distribution and the test IDs are extracted automatically from the pandas dataframe (see below). If you use additional data from the sites (e.g., later waves from ABCD), it may be necessary to adjust the site names to match the names in the training set. See Rutherford et al. [1] for more details.

```
# Which model do we wish to use?
model_name = 'lifespan_57K_82sites'
site_names = 'site_ids_ct_82sites.txt'

# Where the analysis takes place
root_dir = '/content/braincharts'

# Where the data files live
data_dir = '/content/braincharts/docs'

# Where the models live
out_dir = os.path.join(root_dir, 'models', model_name)

# Load a set of site IDs from this model. This must match the training data
with open(os.path.join(root_dir,'docs', site_names)) as f:
    site_ids_tr = f.read().splitlines()
```

3.2 Loading Data

3.2.1 Test Data

First, you need to load the test data. For the purposes of this tutorial, you will make predictions for a multi-site transfer dataset, derived from OpenNeuro.

```
test_data = os.path.join(data_dir, 'OpenNeuroTransfer_ct_te.csv')


df_te = pd.read_csv(test_data)


# Extract a list of unique site ids from the test set

site_ids_te =  sorted(set(df_te['site'].to_list()))
```

3.2.2 Adaption Data

Next, you need to load the adaptation data. If the data you wish to make predictions for are not derived from the same scanning sites as those in the training set, it is necessary to learn the site effect so that it can be accounted for it in the predictions. In order to do this in an unbiased way, it is necessary to use a separate dataset, which is referred to as 'adaptation data. This must contain data for all the same sites as in the test dataset (if not, a warning is displayed) and we assume these are coded in the same way, based on a the 'site-num' column in the dataframe.

```
adaptation_data = os.path.join(data_dir, 'OpenNeuroTransfer_ct_ad.csv')


df_ad = pd.read_csv(adaptation_data)


# Extract a list of unique site ids from the test set

site_ids_ad =  sorted(set(df_ad['site'].to_list()))


if not all(elem in site_ids_ad for elem in site_ids_te):

   print('Warning: some of the testing sites are not in the adaptation data')
```

3.3 Configure the Models to Fit

3.3.1 Select Brain Phenotypes

Now, you configure which *imaging derived phenotypes* (IDPs) you would like to process. This is just a list of column names in the dataframe you have loaded above. You can either load the whole set (i.e., all phenotypes for which you have models for) or you can specify a subset.

```
# Load the list of idps for left and right hemispheres, plus subcortical regions
with open(os.path.join(data_dir, 'phenotypes_ct_lh.txt')) as f:
    idp_ids_lh = f.read().splitlines()
with open(os.path.join(data_dir, 'phenotypes_ct_rh.txt')) as f:
    idp_ids_rh = f.read().splitlines()
with open(os.path.join(data_dir, 'phenotypes_sc.txt')) as f:
    idp_ids_sc = f.read().splitlines()


# We choose here to process all idps
idp_ids = idp_ids_lh + idp_ids_rh + idp_ids_sc


# ... or alternatively, we could just specify a list
idp_ids = [ 'Left-Thalamus-Proper', 'Left-Lateral-Ventricle', 'rh_MeanThickness_thickness']
```

3.3.2 Configure Model Parameters

Now, you should configure some parameters to fit the model. First, choose which columns of the pandas dataframe contain the covariates (age and sex). The site parameters are configured automatically later on by the 'configure_design_matrix()' function when looping through the IDPs in the list. The supplied coefficients are derived from a 'warped' Bayesian linear regression model, which uses a nonlinear warping function to model non-Gaussianity ('sinarcsinh') plus a non-linear basis expansion (a cubic b-spline basis set with 5 knot points, which is the default value in the PCNtoolkit package). For further details about the likelihood warping approach, see Rutherford et al. (2022) [1] and Fraza et al. (2022) [23]. Since you are sticking with the default value, you do not need to specify any parameters for this, but you do need to specify the

limits. Below, you choose to pad the input by a few years either side of the input range and set a couple of options that control the estimation of the model.

```
# Which data columns do we wish to use as covariates?

cols_cov = ['age','sex']


# Limits for cubic B-spline basis

xmin = -5

xmax = 110


# Absolute Z threshold above which a sample is considered to be an outlier (without fitting any

model)

outlier_thresh = 7
```

3.4 Making Predictions

The next step is to make predictions. The code below will make predictions for each IDP separately. This is done by extracting a column from the dataframe (i.e., specifying the IDP as the response variable) and saving it as a numpy array. Next, configure the covariates, which is a numpy data array having the number of rows equal to the number of datapoints in the test set. The columns are specified as follows:

- The covariate columns (here age and sex, coded as 0 = female/1 = male).
- Dummy coded columns for the sites in the training set (one column per site).
- Columns for the basis expansion (seven columns for the default parameterization).

Once these are saved as numpy arrays in ascii format (as here) or (alternatively) in pickle format, these are passed as inputs to the 'predict()' method in the PCNtoolkit normative modelling framework. These are written in the same format to the location specified by 'idp_dir.' At the end of this step, you will have a set of predictions and Z-statistics for the test dataset that you can take forward to further analysis.

Note
When you need to make predictions on new data, the procedure is more involved, since we need to prepare, process and store covariates, response variables, and site ids for the adaptation data.

```
for idp_num, idp in enumerate(idp_ids):
    print('Running IDP', idp_num, idp, ':')
    idp_dir = os.path.join(out_dir, idp)
    os.chdir(idp_dir)

    # Extract and save the response variables for the test set
    y_te = df_te[idp].to_numpy()
```

```
# Save the variables
resp_file_te = os.path.join(idp_dir, 'resp_te.txt')
np.savetxt(resp_file_te, y_te)

# Configure and save the design matrix
cov_file_te = os.path.join(idp_dir, 'cov_bspline_te.txt')
X_te = create_design_matrix(df_te[cols_cov],
                            site_ids = df_te['site'],
                            all_sites = site_ids_tr,
                            basis = 'bspline',
                            xmin = xmin,
                            xmax = xmax)
np.savetxt(cov_file_te, X_te)

# Check whether all sites in the test set are represented in the training set
if all(elem in site_ids_tr for elem in site_ids_te):
    print('All sites are present in the training data')

    # Just make predictions
    yhat_te, s2_te, Z = predict(cov_file_te,
                                alg='blr',
                                respfile=resp_file_te,
```

```
                    model_path=os.path.join(idp_dir,'Models'))
else:
    print('Some sites missing from the training data. Adapting model')

    # Save the covariates for the adaptation data
    X_ad = create_design_matrix(df_ad[cols_cov],
                        site_ids = df_ad['site'],
                        all_sites = site_ids_tr,
                        basis = 'bspline',
                        xmin = xmin,
                        xmax = xmax)
    cov_file_ad = os.path.join(idp_dir, 'cov_bspline_ad.txt')
    np.savetxt(cov_file_ad, X_ad)

    # Save the responses for the adaptation data
    resp_file_ad = os.path.join(idp_dir, 'resp_ad.txt')
    y_ad = df_ad[idp].to_numpy()
    np.savetxt(resp_file_ad, y_ad)

    # Save the site ids for the adaptation data
    sitenum_file_ad = os.path.join(idp_dir, 'sitenum_ad.txt')
    site_num_ad = df_ad['sitenum'].to_numpy(dtype=int)
    np.savetxt(sitenum_file_ad, site_num_ad)
```

```
# Save the site ids for the test data
sitenum_file_te = os.path.join(idp_dir, 'sitenum_te.txt')
site_num_te = df_te['sitenum'].to_numpy(dtype=int)
np.savetxt(sitenum_file_te, site_num_te)


yhat_te, s2_te, Z = predict(cov_file_te,
                            alg = 'blr',
                            respfile = resp_file_te,
                            model_path = os.path.join(idp_dir,'Models'),
                            adaptrespfile = resp_file_ad,
                            adaptcovfile = cov_file_ad,
                            adaptvargroupfile = sitenum_file_ad,
                            testvargroupfile = sitenum_file_te)
```

3.5 Plotting

3.5.1 Configure Age Range of Plots - Dummy Data

In this step, you plot the centiles of variation estimated by the normative model. You do this by making use of a set of dummy covariates that span the whole range of the input space (for age) for a fixed value of the other covariates (e.g., sex) so that we can make predictions for these dummy data points, then plot them. We configure these dummy predictions using the same procedure as

we used for the real data. We can use the same dummy data for all the IDPs we wish to plot.

```
# Which sex do we want to plot?
sex = 1 # 1 = male 0 = female
if sex == 1:
    clr = 'blue';
else:
    clr = 'red'

# Create dummy data for visualization
print('configuring dummy data ...')
xx = np.arange(xmin, xmax, 0.5)
X0_dummy = np.zeros((len(xx), 2))
X0_dummy[:,0] = xx
X0_dummy[:,1] = sex

# Create the design matrix
X_dummy = create_design_matrix(X0_dummy, xmin=xmin, xmax=xmax, site_ids=None,
all_sites=site_ids_tr)

# Save the dummy covariates
cov_file_dummy = os.path.join(out_dir,'cov_bspline_dummy_mean.txt')
np.savetxt(cov_file_dummy, X_dummy)
```

3.5.2 Plot Real Data

Here, you will plot the normative models. First, we loop through the IDPs, plotting each one separately. The outputs of this step are a set of quantitative regression metrics for each IDP and a set of centile curves which we plot the test data against. This part of the code is relatively complex because we need to keep track of many quantities for the plotting. We also need to remember whether the data need to be warped or not. By default, in PCNtoolkit, predictions in the form of 'yhat,' 's2' are always in the warped (Gaussian) space. If we want predictions in the input (non-Gaussian) space, then we need to warp them with the inverse of the estimated warping function. This can be done using the function 'nm.blr.warp.warp_predictions(),'

Note
It is necessary to update the intercept for each of the sites. For purposes of visualization, here we do this by adjusting the median of the data to match the dummy predictions but note that all the quantitative metrics are estimated using the predictions that are adjusted properly using a learned offset (or adjusted using a hold-out adaptation set, as above).

Note For the calibration data we require at least two data points of the same sex in each site to be able to estimate the variance. Of course, in a real example, you would want many more than just two since we need to get a reliable estimate of the variance for each site.

```
sns.set(style='whitegrid')


for idp_num, idp in enumerate(idp_ids):
```

```
print('Running IDP', idp_num, idp, ':')
idp_dir = os.path.join(out_dir, idp)
os.chdir(idp_dir)

# Load the true data points
yhat_te = load_2d(os.path.join(idp_dir, 'yhat_predict.txt'))
s2_te = load_2d(os.path.join(idp_dir, 'ys2_predict.txt'))
y_te = load_2d(os.path.join(idp_dir, 'resp_te.txt'))

# Set up the covariates for the dummy data
print('Making predictions with dummy covariates (for visualisation)')
yhat, s2 = predict(cov_file_dummy,
                   alg = 'blr',
                   respfile = None,
                   model_path = os.path.join(idp_dir,'Models'),
                   outputsuffix = '_dummy')

# Load the normative model
with open(os.path.join(idp_dir,'Models', 'NM_0_0_estimate.pkl'), 'rb') as handle:
    nm = pickle.load(handle)

# Get the warp and warp parameters
W = nm.blr.warp
```

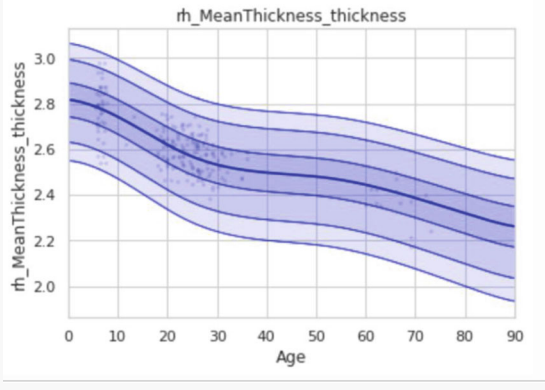
```
warp_param = nm.blr.hyp[1:nm.blr.warp.get_n_params()+1]


# First, we warp predictions for the true data and compute evaluation metrics
med_te = W.warp_predictions(np.squeeze(yhat_te), np.squeeze(s2_te), warp_param)[0]
med_te = med_te[:, np.newaxis]
print('metrics:', evaluate(y_te, med_te))


# Then, we warp dummy predictions to create the plots
med, pr_int = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2), warp_param)


# Extract the different variance components to visualise
beta, junk1, junk2 = nm.blr._parse_hyps(nm.blr.hyp, X_dummy)
s2n = 1/beta # variation (aleatoric uncertainty)
s2s = s2-s2n # modelling uncertainty (epistemic uncertainty)


# Plot the data points
y_te_rescaled_all = np.zeros_like(y_te)
for sid, site in enumerate(site_ids_te):
    # Plot the true test data points
    if all(elem in site_ids_tr for elem in site_ids_te):
        # All data in the test set are present in the training set


        # First, we select the data points belonging to this particular site
```

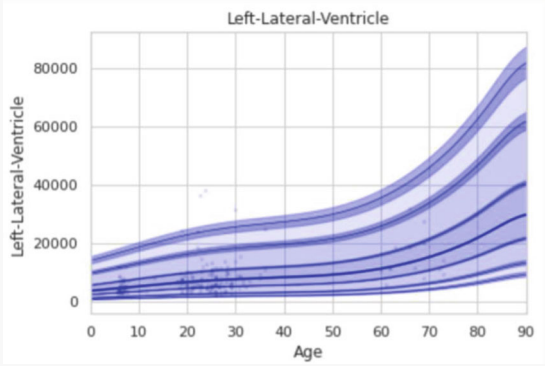
```
        idx = np.where(np.bitwise_and(X_te[:,2] == sex, X_te[:,sid+len(cols_cov)+1] !=0))[0]
        if len(idx) == 0:
            print('No data for site', sid, site, 'skipping...')
            continue

        # Then directly adjust the data
        idx_dummy = np.bitwise_and(X_dummy[:,1] > X_te[idx,1].min(), X_dummy[:,1] <
X_te[idx,1].max())
        y_te_rescaled = y_te[idx] - np.median(y_te[idx]) + np.median(med[idx_dummy])
    else:
        # We need to adjust the data based on the adaptation dataset

        # First, select the data point belonging to this particular site
        idx = np.where(np.bitwise_and(X_te[:,2] == sex, (df_te['site'] == site).to_numpy()))[0]

        # Load the adaptation data
        y_ad = load_2d(os.path.join(idp_dir, 'resp_ad.txt'))
        X_ad = load_2d(os.path.join(idp_dir, 'cov_bspline_ad.txt'))
        idx_a = np.where(np.bitwise_and(X_ad[:,2] == sex, (df_ad['site'] ==
site).to_numpy()))[0]
        if len(idx) < 2 or len(idx_a) < 2:
            print('Insufficent data for site', sid, site, 'skipping...')
            continue
```

```
        # Adjust and rescale the data
        y_te_rescaled, s2_rescaled = nm.blr.predict_and_adjust(nm.blr.hyp,
                                                X_ad[idx_a,:],
                                                np.squeeze(y_ad[idx_a]),
                                                Xs=None,
                                                ys=np.squeeze(y_te[idx]))
    # Plot the (adjusted) data points
    plt.scatter(X_te[idx,1], y_te_rescaled, s=4, color=clr, alpha = 0.1)

  # Plot the median of the dummy data
  plt.plot(xx, med, clr)

  # Fill the gaps in between the centiles
  junk, pr_int25 = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2), warp_param,
percentiles=[0.25,0.75])
  junk, pr_int95 = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2), warp_param,
percentiles=[0.05,0.95])
  junk, pr_int99 = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2), warp_param,
percentiles=[0.01,0.99])
  plt.fill_between(xx, pr_int25[:,0], pr_int25[:,1], alpha = 0.1,color=clr)
  plt.fill_between(xx, pr_int95[:,0], pr_int95[:,1], alpha = 0.1,color=clr)
  plt.fill_between(xx, pr_int99[:,0], pr_int99[:,1], alpha = 0.1,color=clr)
```

```
    # Make the width of each centile proportional to the epistemic uncertainty
    junk, pr_int25l = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2-0.5*s2s),
warp_param, percentiles=[0.25,0.75])
    junk, pr_int95l = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2-0.5*s2s),
warp_param, percentiles=[0.05,0.95])
    junk, pr_int99l = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2-0.5*s2s),
warp_param, percentiles=[0.01,0.99])
    junk, pr_int25u = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2+0.5*s2s),
warp_param, percentiles=[0.25,0.75])
    junk, pr_int95u = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2+0.5*s2s),
warp_param, percentiles=[0.05,0.95])
    junk, pr_int99u = W.warp_predictions(np.squeeze(yhat), np.squeeze(s2+0.5*s2s),
warp_param, percentiles=[0.01,0.99])
    plt.fill_between(xx, pr_int25l[:,0], pr_int25u[:,0], alpha = 0.3,color=clr)
    plt.fill_between(xx, pr_int95l[:,0], pr_int95u[:,0], alpha = 0.3,color=clr)
    plt.fill_between(xx, pr_int99l[:,0], pr_int99u[:,0], alpha = 0.3,color=clr)
    plt.fill_between(xx, pr_int25l[:,1], pr_int25u[:,1], alpha = 0.3,color=clr)
    plt.fill_between(xx, pr_int95l[:,1], pr_int95u[:,1], alpha = 0.3,color=clr)
    plt.fill_between(xx, pr_int99l[:,1], pr_int99u[:,1], alpha = 0.3,color=clr)

    # Plot actual centile lines
    plt.plot(xx, pr_int25[:,0],color=clr, linewidth=0.5)
```

```
    plt.plot(xx, pr_int25[:,1],color=clr, linewidth=0.5)
    plt.plot(xx, pr_int95[:,0],color=clr, linewidth=0.5)
    plt.plot(xx, pr_int95[:,1],color=clr, linewidth=0.5)
    plt.plot(xx, pr_int99[:,0],color=clr, linewidth=0.5)
    plt.plot(xx, pr_int99[:,1],color=clr, linewidth=0.5)

    plt.xlabel('Age')
    plt.ylabel(idp)
    plt.title(idp)
    plt.xlim((0,90))
    plt.savefig(os.path.join(idp_dir, 'centiles_' + str(sex)),  bbox_inches='tight')
    plt.show()

os.chdir(out_dir)
```





3.6 Organize the Output Files into a Single CSV File

Here, you will explore an example output folder of a single model (one ROI). It is useful to understand what each of these output files represents. Look at the variable names and comments in the code block above:

```
# folder contents

get_ipython().system(' ls rh_MeanThickness_thickness/')
```

You should check that the number of deviation scores matches the number of subjects in the test set. There should be one deviation score per subject (one line per subject), which you can verify by counting the line numbers in the Z_predict.txt file:

```
# lines count

get_ipython().system(' cat rh_MeanThickness_thickness/Z_predict.txt | wc')
```

The deviation scores are output as a text file in separate folders. You will want to summarize the deviation scores across all models estimates so you organize them into a single file and merge the deviation scores into the original data file.

```
get_ipython().system(' mkdir deviation_scores')


get_ipython().system(' for i in *; do if [[ -e ${i}/Z_predict.txt ]]; then cp ${i}/Z_predict.txt
deviation_scores/${i}_Z_predict.txt; fi; done')


z_dir = '/content/braincharts/models/' + model_name + '/deviation_scores/'
filelist = [name for name in os.listdir(z_dir)]
os.chdir(z_dir)
Z_df = pd.concat([pd.read_csv(item, names=[item[:-4]]) for item in filelist], axis=1)


df_te.reset_index(inplace=True)


Z_df['sub_id'] = df_te['sub_id']


df_te_Z = pd.merge(df_te, Z_df, on='sub_id', how='inner')


df_te_Z.to_csv('OpenNeuroTransfer_deviation_scores.csv', index=False)
```

4 Code Tutorial 2: Visualizing the Results

This tutorial walks through several examples that visualize the outputs created by the normative modeling analysis that was run in tutorial 1. Again, this code can be run in your web browser using Google Colab here.

4.1 Brain Space Visualization of Extreme Deviations

First, we count the number of extreme (positive and negative) deviations at each brain region and visualize the count for each hemisphere. You can click around in 3D space on the visualizations (Scroll in/out, move the brain around, etc.)

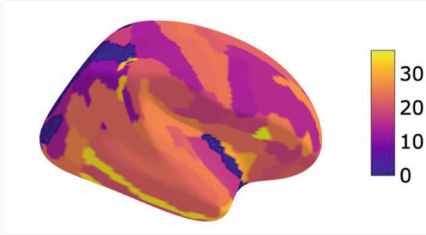
```
#!/usr/bin/env python


get_ipython().system(' git clone https://github.com/predictive-clinical-
neuroscience/PCNtoolkit-demo.git')


import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from nilearn import plotting
import nibabel as nib
from nilearn import datasets


os.chdir('/content/PCNtoolkit-demo')


Z_df = pd.read_csv('data/Z_long_format.csv')


# Change this threshold to view more or less extreme deviations.
# Discuss what you think is an appropriate threshold and adjust the below variables
accordingly.
Z_positive = Z_df.query('value > 2')
Z_negative = Z_df.query('value < -2')


positive_left_z = Z_positive.query('hemi == "left"')
positive_right_z = Z_positive.query('hemi == "right"')
positive_sc_z = Z_positive.query('hemi == "subcortical"')
negative_left_z = Z_negative.query('hemi == "left"')
negative_right_z = Z_negative.query('hemi == "right"')
negative_sc_z = Z_negative.query('hemi == "subcortical"')


positive_left_z2 =
positive_left_z['ROI_name'].value_counts().rename_axis('ROI').reset_index(name='counts')
```

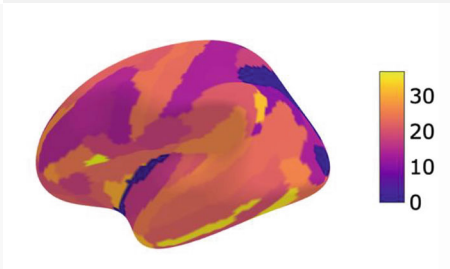
```
positive_right_z2 =
positive_right_z['ROI_name'].value_counts().rename_axis('ROI').reset_index(name='counts')
positive_sc_z2 =
positive_sc_z['ROI_name'].value_counts().rename_axis('ROI').reset_index(name='counts')
negative_left_z2 =
negative_left_z['ROI_name'].value_counts().rename_axis('ROI').reset_index(name='counts')
negative_right_z2 =
negative_right_z['ROI_name'].value_counts().rename_axis('ROI').reset_index(name='counts')
negative_sc_z2 =
negative_sc_z['ROI_name'].value_counts().rename_axis('ROI').reset_index(name='counts')


destrieux_atlas = datasets.fetch_atlas_surf_destrieux()
fsaverage = datasets.fetch_surf_fsaverage()


# The parcellation is already loaded into memory
parcellation_l = destrieux_atlas['map_left']
parcellation_r = destrieux_atlas['map_right']


nl = pd.read_csv('data/nilearn_order.csv')


atlas_r = destrieux_atlas['map_right']
atlas_l = destrieux_atlas['map_left']
```

```
nl_ROI = nl['ROI'].to_list()


nl_positive_left = pd.merge(nl, positive_left_z2, on='ROI', how='left')

nl_positive_right = pd.merge(nl, positive_right_z2, on='ROI', how='left')


nl_positive_left['counts'] = nl_positive_right['counts'].fillna(0)

nl_positive_right['counts'] = nl_positive_right['counts'].fillna(0)


nl_positive_left = nl_positive_left['counts'].to_numpy()

nl_positive_right = nl_positive_right['counts'].to_numpy()


a_list = list(range(1, 76))

parcellation_positive_l = atlas_l

for i, j in enumerate(a_list):

    parcellation_positive_l = np.where(parcellation_positive_l == j, nl_positive_left[i],

parcellation_positive_l)


a_list = list(range(1, 76))

parcellation_positive_r = atlas_r

for i, j in enumerate(a_list):

    parcellation_positive_r = np.where(parcellation_positive_r == j, nl_positive_right[i],

parcellation_positive_r)
```

```
view_pos_r = plotting.view_surf(fsaverage.infl_right, parcellation_positive_r,
threshold=None, symmetric_cmap=False, cmap='plasma', bg_map=fsaverage.sulc_right)
```

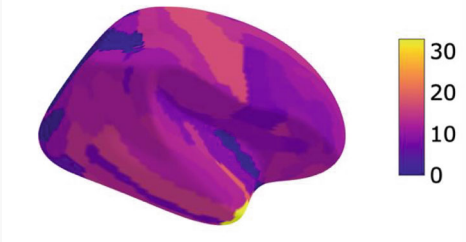



```
view_pos_l = plotting.view_surf(fsaverage.infl_left, parcellation_positive_l, threshold=None,
symmetric_cmap=False, cmap='plasma', bg_map=fsaverage.sulc_left)
```

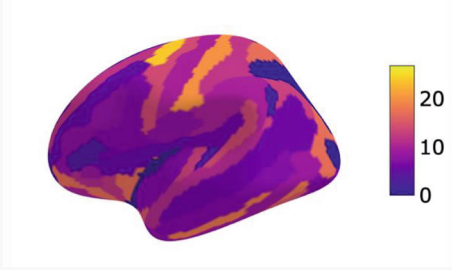



```
nl_negative_left = pd.merge(nl, negative_left_z2, on='ROI', how='left')
nl_negative_right = pd.merge(nl, negative_right_z2, on='ROI', how='left')

nl_negative_left['counts'] = nl_negative_left['counts'].fillna(0)
nl_negative_right['counts'] = nl_negative_right['counts'].fillna(0)

nl_negative_left = nl_negative_left['counts'].to_numpy()
nl_negative_right = nl_negative_right['counts'].to_numpy()
```

```
a_list = list(range(1, 76))
parcellation_negative_l = atlas_l
for i, j in enumerate(a_list):
    parcellation_negative_l = np.where(parcellation_negative_l == j, nl_negative_left[i],
parcellation_negative_l)

a_list = list(range(1, 76))
parcellation_negative_r = atlas_r
for i, j in enumerate(a_list):
    parcellation_negative_r = np.where(parcellation_negative_r == j, nl_negative_right[i],
parcellation_negative_r)

view_neg_r = plotting.view_surf(fsaverage.infl_right, parcellation_negative_r,
threshold=None, symmetric_cmap=False, cmap='plasma', bg_map=fsaverage.sulc_right)
```




```
view_neg_l = plotting.view_surf(fsaverage.infl_left, parcellation_negative_l, threshold=None,
symmetric_cmap=False, cmap='plasma', bg_map=fsaverage.sulc_left)
```

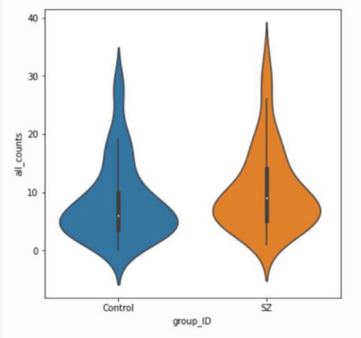


4.2 Violin Plots of the Extreme Deviations

Here, you can count the number of 'extreme' deviations that each person has (both positive and negative) and summarize the distribution of extreme deviations for healthy controls and patients with schizophrenia.

```
Z_df = pd.read_csv('data/fcon1000_te_Z.csv')

deviation_counts = Z_df.loc[:, Z_df.columns.str.contains('Z_predict')]

deviation_counts['positive_count'] = deviation_counts[deviation_counts >= 2].count(axis=1)

deviation_counts['negative_count'] = deviation_counts[deviation_counts <= -2].count(axis=1)

deviation_counts['participant_id'] = Z_df['sub_id']
deviation_counts['group_ID'] = Z_df['group']
deviation_counts['site_ID'] = Z_df['site']
```

```
deviation_counts['all_counts'] = deviation_counts['positive_count'] +
deviation_counts['negative_count']


fig, ax = plt.subplots(figsize=(6,6))
sns.violinplot(data=deviation_counts, y="all_counts", x="group_ID", inner='box', ax=ax);
plt.legend=False
```



5 Conclusion

5.1 Evaluation

There are multiple results created from the normative model analysis. First, the evaluation metrics for each model (brain region) are saved to a CSV file in Subheading 3. The evaluation metrics can be visualized in numerous formats, such as histograms/density plots, scatter plots with fitted centiles, or brain-space visualizations. Several examples of these visualizations were shown in Subheading 4 on visualization. Quality checking the normative model evaluation metrics should be done to ensure proper model estimation. If a model fits well to the data, the evaluation metrics should follow a Gaussian distribution. Beyond the summary metrics, there are individual metrics that can be helpful for interpretation because they quantify the uncertainty of each individual's predicted value (for every brain region).

5.2 Limitations

As datasets grow in size, there is a need to use automated quality metrics. This means there could unintentionally be poor quality data included in the training set. Whenever possible, users should consider manually quality checking their own data. If the dataset is

too large to check every subject, consider randomly checking a portion of the dataset and using the automated quality metrics to inform a threshold for which subjects should be manually visually inspected.

Another consideration, when training big data normative models, is that there are going to be differences in the available data modalities collected across studies and sites. The commonly available data needs to be considered when deciding which studies to include and which covariates to use in modeling. If the goal is to share the model, using uncommon covariates or brain measures will affect the utility and accessibility of the model.

It is also important to consider that normative modeling may not always be the best approach in all settings, as it is dependent on the chosen reference population (usually but not necessarily taken to be population of healthy controls), and it might not be appropriate in certain cases, for example, if there are substantial differences in the target cohort that are unrelated to the clinical condition of interest. In some situations, such as when studying rare diseases, there may not be enough data available to establish a normative model, or the population of healthy controls may not be representative of the population being studied. However, we acknowledge that we are not obliged to fit the normative model only using healthy data. It is equally valid to fit a normative model using patient and control data, just note that changing the reference cohort (training set) demographics changes the interpretation of the centiles. If you are modeling 'healthy' lifespan populations, the sample size will likely be large (on the order of thousands) because of the availability of publicly shared data from healthy controls. In contrast, if you want to model a specific clinical population, the sample size will be smaller due to availability of data. A smaller dataset that appropriately addresses the given research question is suitable. There is no 'one size fits all' approach to normative modeling.

5.3 Post-hoc Analysis

There are multiple possible downstream analyses that can be performed after a normative model has been fit, evaluated, and visualized. While there are too many diverging paths to cover them all within this chapter, we highlight recent work on post-hoc normative modeling possibilities. These include sub-typing using clustering algorithms [17] and stratification [8]. For example, such approaches applied to autism spectrum disorder (ASD) have shown particular promise for parsing the biological heterogeneity underlying this condition. For example Zabihi et al. [17] showed that a subset of individuals with autism have widespread patterns of increased cortical thickness relative to population norms, whereas others have widespread patterns of decreased cortical thickness. We refer to Rutherford et al. [24] for a detailed overview of the possibilities of downstream analyses that can be conducted using

normative models and an in-depth comparison between normative modeling outputs (deviation scores) and raw data using different data modalities (structural and functional MRI) across several tasks (multivariate prediction (regression and classification) and case-control group different testing). Owing to this flexibility, and the ability to move beyond group level inferences to individual prediction, we consider that normative modeling is a promising method for understanding variation in large datasets.

References

1. Rutherford S, Fraza C, Dinga R, Kia SM, Wolfers T, Zabihi M, Berthet P, Worker A, Verdi S, Andrews D, Han LK, Bayer JM, Dazzan P, McGuire P, Mocking RT, Schene A, Sripada C, Tso IF, Duval ER, Chang S-E, Penninx BW, Heitzeg MM, Burt SA, Hyde LW, Amaral D, Wu Nordahl C, Andreasssen OA, Westlye LT, Zahn R, Ruhe HG, Beckmann C, Marquand AF (2022) Charting brain growth and aging at high spatial precision. eLife 11:e72904. https://doi.org/10.7554/eLife.72904
2. de Boer AAA, Kia SM, Rutherford S, Zabihi M, Fraza C, Barkema P, Westlye LT, Andreassen OA, Hinne M, Beckmann CF, Marquand A (2022) Non-gaussian normative modelling with hierarchical bayesian regression. bioRxiv. https://doi.org/10.1101/2022.10.05.510988
3. Dinga R, Fraza CJ, Bayer JMM, Kia SM, Beckmann CF, Marquand AF (2021) Normative modeling of neuroimaging data using generalized additive models of location scale and shape. bioRxiv. https://doi.org/10.1101/2021.06.14.448106
4. Fraza CJ, Dinga R, Beckmann CF, Marquand AF (2021) Warped Bayesian linear regression for normative modelling of big data. NeuroImage 245:118715. https://doi.org/10.1016/j.neuroimage.2021.118715
5. Kia SM, Huijsdens H, Dinga R, Wolfers T, Mennes M, Andreassen OA, Westlye LT, Beckmann CF, Marquand AF (2020) Hierarchical Bayesian regression for multi-site normative modeling of neuroimaging data. In: Martel AL, Abolmaesumi P, Stoyanov D, Mateus D, Zuluaga MA, Zhou SK, Racoceanu D, Joskowicz L (eds) Medical image computing and computer assisted intervention—MICCAI 2020. Springer International Publishing, Cham, pp 699–709
6. Kia SM, Huijsdens H, Rutherford S, de Boer A, Dinga R, Wolfers T, Berthet P, Mennes M, Andreassen OA, Westlye LT, Beckmann CF, Marquand AF (2022) Closing the life-cycle of normative modeling using federated hierarchical Bayesian regression. PLoS One 17:e0278776. https://doi.org/10.1371/journal.pone.0278776
7. Kia SM, Marquand A (2018) Normative modeling of neuroimaging data using scalable multi-task Gaussian processes. arXiv:1806.01047. https://doi.org/10.48550/arXiv.1806.01047
8. Floris DL, Wolfers T, Zabihi M, Holz NE, Zwiers MP, Charman T, Tillmann J, Ecker C, Dell'Acqua F, Banaschewski T, Moessnang C, Baron-Cohen S, Holt R, Durston S, Loth E, Murphy DGM, Marquand A, Buitelaar JK, Beckmann CF, Ahmad J, Ambrosino S, Auyeung B, Banaschewski T, Baron-Cohen S, Baumeister S, Beckmann CF, Bölte S, Bourgeron T, Bours C, Brammer M, Brandeis D, Brogna C, de Bruijn Y, Buitelaar JK, Chakrabarti B, Charman T, Cornelissen I, Crawley D, Dell'Acqua F, Dumas G, Durston S, Ecker C, Faulkner J, Frouin V, Garcés P, Goyard D, Ham L, Hayward H, Hipp J, Holt R, Johnson MH, Jones EJH, Kundu P, Lai M-C, Liogier d'Ardhuy X, Lombardo MV, Loth E, Lythgoe DJ, Mandl R, Marquand A, Mason L, Mennes M, Meyer-Lindenberg A, Moessnang C, Mueller N, Murphy DGM, Oakley B, O'Dwyer L, Oldehinkel M, Oranje B, Pandina G, Persico AM, Ruggeri B, Ruigrok A, Sabet J, Sacco R, San José Cáceres A, Simonoff E, Spooren W, Tillmann J, Toro R, Tost H, Waldman J, Williams SCR, Wooldridge C, Zwiers MP (2021) Atypical brain asymmetry in autism—a candidate for clinically meaningful stratification. Biol Psychiatry Cogn Neurosci Neuroimaging 6:802–812. https://doi.org/10.1016/j.bpsc.2020.08.008
9. Pinaya WHL, Scarpazza C, Garcia-Dias R, Vieira S, Baecker L, da Costa PF, Redolfi A, Frisoni GB, Pievani M, Calhoun VD, Sato JR, Mechelli A (2021) Using normative modelling to detect disease progression in mild cognitive impairment and Alzheimer's disease in a cross-

sectional multi-cohort study. Sci Rep 11: 15746. https://doi.org/10.1038/s41598-021-95098-0

10. Remiszewski N, Bryant JE, Rutherford SE, Marquand AF, Nelson E, Askar I, Lahti AC, Kraguljac NV (2022) Contrasting case-control and normative reference approaches to capture clinically relevant structural brain abnormalities in patients with first-episode psychosis who are antipsychotic naive. JAMA Psychiatry 79: 1133–1138. https://doi.org/10.1001/jamapsychiatry.2022.3010
11. Verdi S, Marquand AF, Schott JM, Cole JH (2021) Beyond the average patient: how neuroimaging models can address heterogeneity in dementia. Brain J Neurol 144:2946–2953. https://doi.org/10.1093/brain/awab165
12. Wolfers T, Arenas AL, Onnink AMH, Dammers J, Hoogman M, Zwiers MP, Buitelaar JK, Franke B, Marquand AF, Beckmann CF (2017) Refinement by integration: aggregated effects of multimodal imaging markers on adult ADHD. J Psychiatry Neurosci 42: 386–394. https://doi.org/10.1503/jpn.160240
13. Wolfers T, Doan NT, Kaufmann T, Alnæs D, Moberget T, Agartz I, Buitelaar JK, Ueland T, Melle I, Franke B, Andreassen OA, Beckmann CF, Westlye LT, Marquand AF (2018) Mapping the heterogeneous phenotype of schizophrenia and bipolar disorder using normative models. JAMA Psychiatry 75:1146–1155. https://doi.org/10.1001/jamapsychiatry.2018.2467
14. Wolfers T, Beckmann CF, Hoogman M, Buitelaar JK, Franke B, Marquand AF (2020) Individual differences v. the average patient: mapping the heterogeneity in ADHD using normative models. Psychol Med 50:314–323. https://doi.org/10.1017/S0033291719000084
15. Wolfers T, Rokicki J, Alnaes D, Berthet P, Agartz I, Kia SM, Kaufmann T, Zabihi M, Moberget T, Melle I, Beckmann CF, Andreassen OA, Marquand AF, Westlye LT (2021) Replicating extensive brain structural heterogeneity in individuals with schizophrenia and bipolar disorder. Hum Brain Mapp 42:2546–2555. https://doi.org/10.1002/hbm.25386
16. Zabihi M, Oldehinkel M, Wolfers T, Frouin V, Goyard D, Loth E, Charman T, Tillmann J, Banaschewski T, Dumas G, Holt R, Baron-Cohen S, Durston S, Bölte S, Murphy D, Ecker C, Buitelaar JK, Beckmann CF, Marquand AF (2019) Dissecting the heterogeneous cortical anatomy of autism Spectrum disorder using normative models. Biol Psychiatry Cogn Neurosci Neuroimaging 4:567–578. https://doi.org/10.1016/j.bpsc.2018.11.013
17. Zabihi M, Floris DL, Kia SM, Wolfers T, Tillmann J, Arenas AL, Moessnang C, Banaschewski T, Holt R, Baron-Cohen S, Loth E, Charman T, Bourgeron T, Murphy D, Ecker C, Buitelaar JK, Beckmann CF, Marquand A, EU-AIMS LEAP Group (2020) Fractionating autism based on neuroanatomical normative modeling. Transl Psychiatry 10:384. https://doi.org/10.1038/s41398-020-01057-0
18. Marquand A, Rutherford S, Kia SM, Wolfers T, Fraza C, Dinga R, Zabihi M (2021) PCNToolkit
19. Marquand AF, Rezek I, Buitelaar J, Beckmann CF (2016) Understanding heterogeneity in clinical cohorts using normative models: beyond case-control studies. Biol Psychiatry 80:552–561. https://doi.org/10.1016/j.biopsych.2015.12.023
20. Marquand AF, Kia SM, Zabihi M, Wolfers T, Buitelaar JK, Beckmann CF (2019) Conceptualizing mental disorders as deviations from normative functioning. Mol Psychiatry 24: 1415–1424. https://doi.org/10.1038/s41380-019-0441-1
21. Rutherford S, Kia SM, Wolfers T, Fraza C, Zabihi M, Dinga R, Berthet P, Worker A, Verdi S, Ruhe HG, Beckmann CF, Marquand AF (2022) The normative modeling framework for computational psychiatry. Nat Protoc 17:1711–1734. https://doi.org/10.1038/s41596-022-00696-5
22. Borghi E, de Onis M, Garza C, Van den Broeck J, Frongillo EA, Grummer-Strawn L, Van Buuren S, Pan H, Molinari L, Martorell R, Onyango AW, Martines JC, WHO Multicentre Growth Reference Study Group (2006) Construction of the World Health Organization child growth standards: selection of methods for attained growth curves. Stat Med 25:247–265. https://doi.org/10.1002/sim.2227
23. Fraza C, Zabihi M, Beckmann CF, Marquand AF (2022) The extremes of normative modelling. bioRxiv. https://doi.org/10.1101/2022.08.23.505049
24. Rutherford S, Barkema P, Tso IF, Sripada C, Beckmann CF, Ruhe HG, Marquand AF (2022) Evidence for embracing normative modeling. Elife. https://doi.org/10.7554/eLife.85082