Predictive Clinical Neuroscience Toolkit

Release 0.20

Saige Rutherford, Andre F. Marquand

Dec 16, 2023

GETTING STARTED

CHAPTER

ONE

INSTALLATION

1.1 Basic installation (on a local machine)

1. Install anaconda3
2. Create enviornment

```
conda create --name <env_name>
```

3. Activate environment

```
source activate <env_name>
```

4. Install required conda packages

```
conda install pip pandas scipy
```

5. Install PCNtoolkit (plus dependencies)

```
pip install pcntoolkit
```

1.2 Alternative installation (on a shared resource)

1. Make sure conda is available on the system. Otherwise install it first from https://www.anaconda.com/

```
conda --version
```

2. Create a conda environment in a shared location

```
conda create -y python==3.7.7 numpy mkl blas --prefix=/shared/conda/<env_name>
```

3. Activate the conda environment

```
conda activate /shared/conda/<env_name>
```

4. Install other dependencies

```
conda install -y pandas scipy
```

5. Install pip dependencies

```
pip --no-cache-dir install nibabel sklearn torch glob3
```

6. Clone the repo

```
git clone https://github.com/amarquand/PCNtoolkit.git
```

7. Install in the conda environment

```
cd PCNtoolkit/
python3 setup.py install
```

8. Test

```
python -c "import pcntoolkit as pk;print(pk.__file__)"
```

1.3 Quickstart usage

For normative modelling, functionality is handled by the `normative.py` script, which can be run from the command line, e.g.

```
python normative.py -c /path/to/training/covariates -t /path/to/test/covariates -r /path/
↪to/test/response/variables /path/to/my/training/response/variables
```

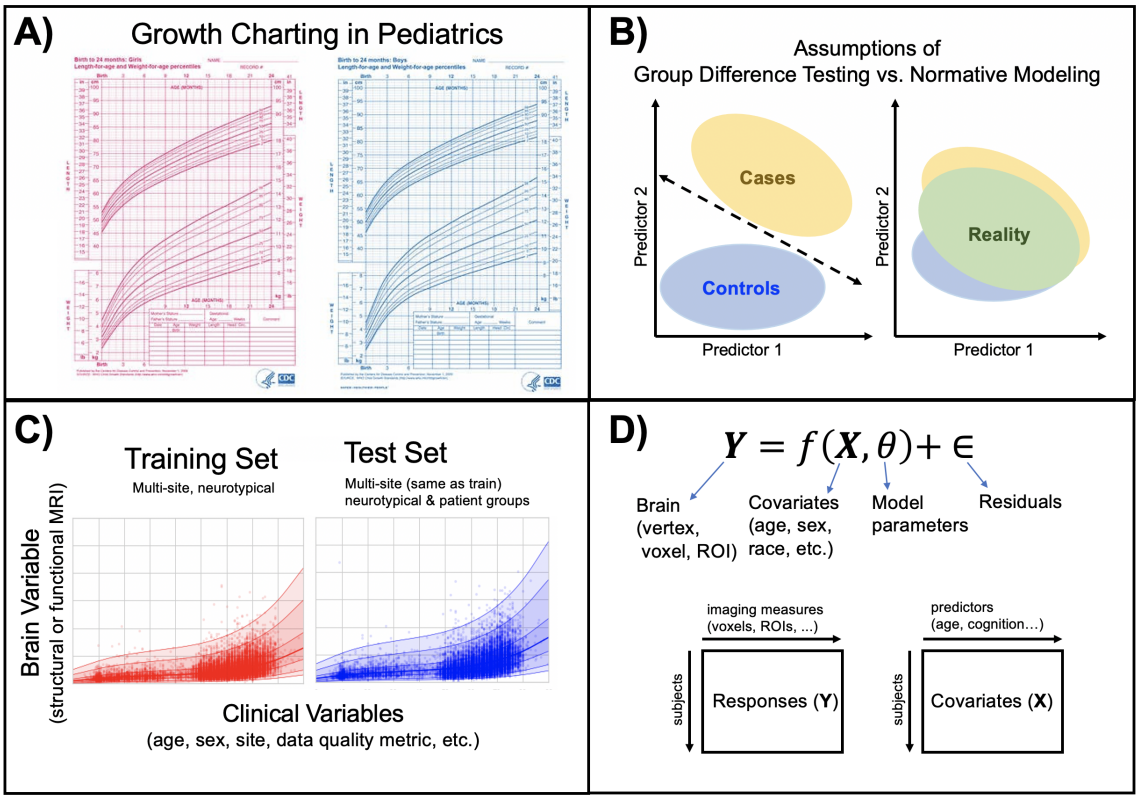
CHAPTER

TWO

PCNTOOLKIT BACKGROUND

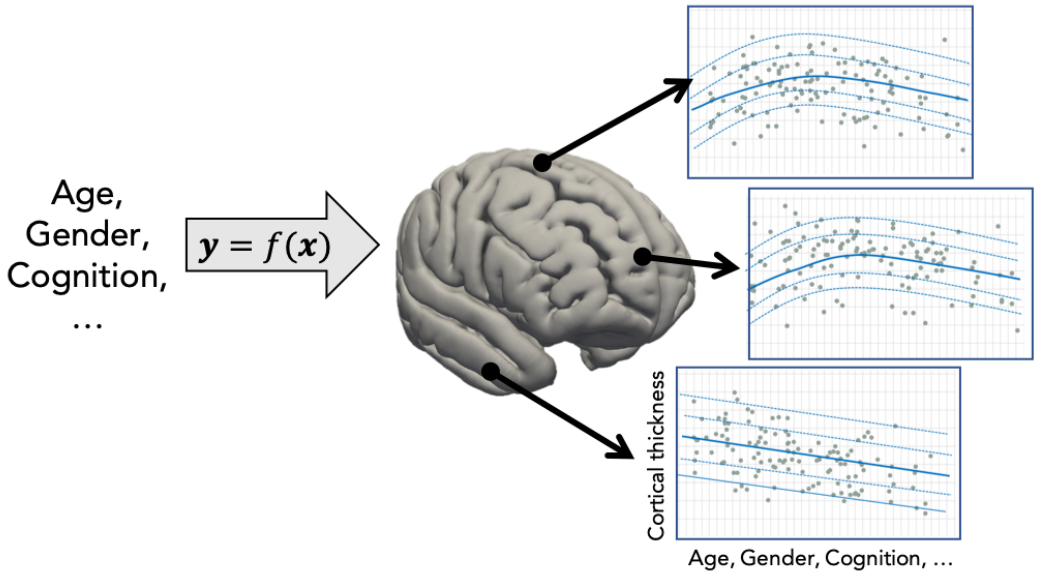
2.1 What is the PCNtoolkit?

Predictive Clinical Neuroscience (PCN) toolkit (formerly nispat) 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.

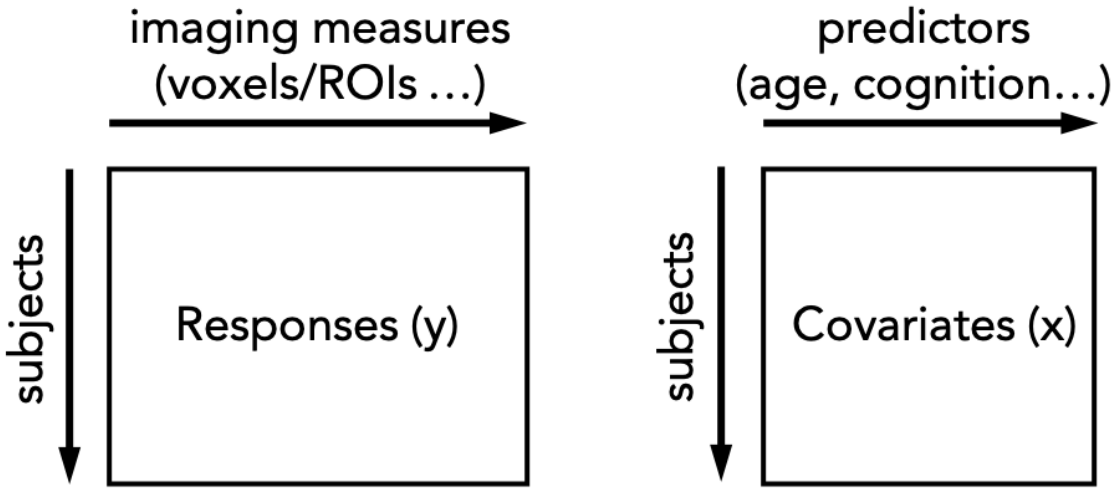


2.1.1 Intro to normative modelling

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) A conceptual overview of the approach can be found in this publication. 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.



In practice, this 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:



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 more efficiently. 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, which is the default method in this toolkit. Typically (but not always), each response variable is estimated independently.

2.1.2 Data formats

Generally the covariates are specified in text format, roughly following the FSL convention in that the text file should contain one entry (i.e. subject) per line, with columns space or tab separated and no headers. For example:

```
head cov.txt
52 55 94 4.6
49 43 59 4.6
56 80 63 5.6
39 48 42 4.3
```

For the response variables, the following data formats are supported:

- NIfTI (e.g. .nii.gz or .img/.hdr)
- CIFTI (e.g. .dtseries.nii)
- Pickle/pandas (e.g. .pkl)
- ASCII text (e.g. .txt, .csv, .tsv)

For nifti/cifti formats, data should be in timeseries format with subjects along the time dimension and these images will be masked and reshaped into vectors. If no mask is specified, one will be created automatically from the image data.

2.1.3 Basic usage (command line)

The simplest method to estimate a normative model is using the `normative.py` script which can be run from the command line or imported as a python module. For example, the following command will estimate a normative model on the basis of the matrix of covariates and responses specified in cov.txt and resp.txt respectively. These are simply tab or space separated ASCII text files that contain the variables of interest, with one subject per row.

```
python normative.py -c cov.txt -k 5 -a blr resp.txt
```

The argument `-a blr` tells the script to use Bayesian Linear regression rather than the default Gaussian process regression model and `-k 5` tells the script to run internal 5-fold cross-validation across all subjects in the covariates and responses files. Alternatively, the model can be evaluated on a separate dataset by specifying test covariates (and optionally also test responses). The following estimation algorithms are supported

Table 1: Estimation algorithms

key value	Description	Reference
hbr	Hierarchical Bayesian Regression	Kia et al 2020
blr	Bayesian Linear Regression	Huertas et al 2017
np	Neural Processes	Kia et al 2018
rfa	Random Feature Approximation	Rahimi and Recht 2007

Note that keyword arguments can also be specified from the command line to offer additional flexibility. For example, the following command will fit a normative model to the same data, but without standardizing the data first and additionally writing out model coefficients (this is not done by default because they can use a lot of disk space).

```
python normative.py -c cov.txt -k 5 -a blr resp.txt standardize=False savemodel=True
```

A full set of keyword arguments is provided in the table below. At a minimum, a set of responses and covariates must be provided and either the corresponding number of cross-validation folds or a set of test covariates.

Table 2: Keywords and command line arguments

Keyword	Command line shortcut	Description
covfunc	-c filename	Covariate file
cvfolds	-k num_folds	Number of cross-validation folds
testcov	-t filename	Test covariates
testresp	-r filename	Test responses
maskfile	-m filename	mask to apply to the response variables (nifti/cifti only)
alg	-a algorithm	Estimation algorithm: 'gpr' (default), 'blr', 'np', 'hbr' or 'rfa'. See table above.
function	-f function	function to call (estimate, predict, transfer, extend). See below
standardize	-s (skip)	Standardize the covariates and response variables using the training data
configparam	-x config	Pass the value of config to the estimation algorithm (deprecated)
outputsuffix		Suffix to apply to the output variables
saveoutput		Write output (default = True)
savemodel		Save the model coefficients and meta-data (default = False)
warp		Warping function to apply to the responses (blr only)

2.1.4 Basic usage (scripted)

The same can be done by importing the estimate function from `normative.py`. For example, the following code snippet will: (i) mask the nifti data specified in resp_train.nii.gz using the mask specified (which must have the same voxel size as the response variables) (ii) fit a linear normative model to each voxel, (iii) apply this to make predictions using the test covariates and (iv) compute deviation scores and error metrics by comparing against the true test response variables.

```
from pcntoolkit.normative import estimate

# estimate a normative model
estimate("cov_train.txt", "resp_train.nii.gz", maskfile="mask.nii.gz", \
        testresp="resp_test.nii.gz", testcov="cov_test.txt", alg="blr")
```

The estimate function does all these operations in a single step. In some cases it may be desirable to separate these steps. For example, if a normative model has been estimated on a large dataset, it may be desirable to save the model before applying it to a new dataset (e.g. from a a different site). For example, the following code snippet will first fit a model, then apply it to a set of dummy covariates so that the normative model can be plotted

```
from pcntoolkit.normative import estimate, predict

# fit a normative model, using training covariates and responses
# then apply to test dataset. Saved with file suffix '_estimate'
estimate(cov_file_tr, resp_file_tr, testresp=resp_file_te, \
        testcov=cov_file_te, alg='blr', optimizer = 'powell', \
        savemodel=True, standardize = False)

# make predictions on a set of dummy covariates (with no responses)
```

(continues on next page)

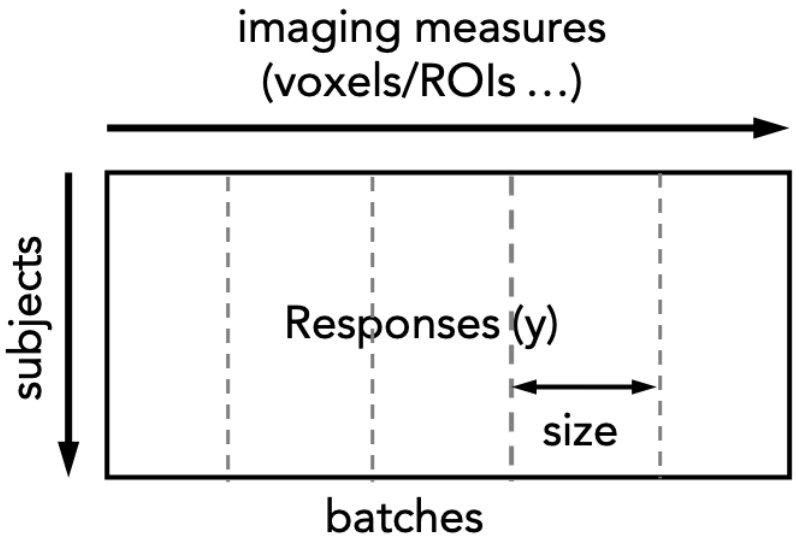
(continued from previous page)

```
# Saved with file suffix '_predict'
yhat, s2 = predict(cov_file_dummy)
```

For further information, see the developer documentation. The same can be achieved from the command line, using te `-f` argument, for example, by specifying `-f predict`.

2.1.5 Paralellising estimation to speed things up

Normative model estimation is typically quite computationally expensive, especially for large datasets. This is exacerbated by high-resolution data (e.g. voxelwise data). For such cases normative model estimation can be paralellised across multiple compute nodes which can be achieved using the `normative_parallel.py` script. This involves splitting the response matrix into a set of batches, each of a specified size, i.e.:



Each of these are then submitted to a cluster and reassembled once the cluster jobs have been completed. The following code snippet illustrates this procedure:

```
from pcntoolkit.normative_parallel import execute_nm, collect_nm, delete_nm

# General config parameters
normative_path = '/<path-to-my>/pcntoolkit/normative.py'
python_path='/<path-to-my>/bin/python'
working_dir = '/<where-results-will-be_stored>/'
log_dir = '/<where-logs-will-be_stored>/'

# cluster paramateters
job_name = 'nm_demo'   # name for the cluster job
batch_size = 10        # number of models (e.g. voxels) per batch
memory = '4gb'         # memory required
duration = '01:00:00'  # walltime
cluster = 'torque'

# fit the model. Specifying binary=True means results will be stored in .pkl format
```

(continues on next page)

(continued from previous page)

```
execute_nm(working_dir, python_path, normative_path, job_name, cov_file.txt, \
        resp_file.pkl, batch_size, memory, duration, cluster_spec=cluster, \
        cv_folds=2, log_path=log_dir, binary=True)

# wait until jobs complete ...

# reassemble results
collect_nm(working_dir, job_name, collect=True, binary=True)

# remove temporary files
delete_nm(working_dir, binary=True)
```

At the present time, only ASCII and pickle format are supported using normative parallel. Note also that it may be necessary to customise the script to support your local cluster architecture. This can be done using fairly obvious modifications to the `execute_nm()` function.

CHAPTER

THREE

MODULE INDEX

class bayesreg.**BLR**(***kwargs*)

Bases: `object`

Bayesian linear regression

Estimation and prediction of Bayesian linear regression models

Basic usage:

```
B = BLR()
hyp = B.estimate(hyp0, X, y)
ys,s2 = B.predict(hyp, X, y, Xs)
```

where the variables are

Parameters

- **hyp** – vector of hyperparmaters.
- **X** – N x D data array
- **y** – 1D Array of targets (length N)
- **Xs** – Nte x D array of test cases
- **hyp0** – starting estimates for hyperparameter optimisation

Returns

- ys - predictive mean
- s2 - predictive variance

The hyperparameters are:

```
hyp = ( log(beta), log(alpha) )  # hyp is a list or numpy array
```

The implementation and notation mostly follows Bishop (2006). The hyperparameter beta is the noise precision and alpha is the precision over lengthscale parameters. This can be either a scalar variable (a common lengthscale for all input variables), or a vector of length D (a different lengthscale for each input variable, derived using an automatic relevance determination formulation). These are estimated using conjugate gradient optimisation of the marginal likelihood.

Reference: Bishop (2006) Pattern Recognition and Machine Learning, Springer

Written by A. Marquand

dloglik(*hyp*, *X*, *y*, *Xv=None*)

Function to compute derivatives

estimate(*hyp0*, *X*, *y*, ***kwargs*)

Function to estimate the model

Parameters

- **hyp** – hyperparameter vector
- **X** – covariates
- **y** – responses
- **optimizer** – optimisation algorithm ('cg','powell','nelder-mead','l0bfgs-b')

loglik(*hyp*, *X*, *y*, *Xv=None*)

Function to compute compute log (marginal) likelihood

penalized_loglik(*hyp*, *X*, *y*, *Xv=None*, *l=0.1*, *norm='L1'*)

Function to compute the penalized log (marginal) likelihood

Parameters

- **hyp** – hyperparameter vector
- **X** – covariates
- **y** – responses
- **Xv** – covariates for heteroskedastic noise
- **l** – regularisation penalty
- **norm** – type of regulariser (L1 or L2)

post(*hyp*, *X*, *y*, *Xv=None*)

Generic function to compute posterior distribution.

This function will save the posterior mean and precision matrix as self.m and self.A and will also update internal parameters (e.g. N, D and the prior covariance (Sigma_a) and precision (Lambda_a).

Parameters

- **hyp** – hyperparameter vector
- **X** – covariates
- **y** – responses
- **Xv** – covariates for heteroskedastic noise

predict(*hyp*, *X*, *y*, *Xs*, *var_groups_test=None*, *var_covariates_test=None*, ***kwargs*)

Function to make predictions from the model

Parameters

- **hyp** – hyperparameter vector
- **X** – covariates for training data
- **y** – responses for training data
- **Xs** – covariates for test data
- **var_covariates_test** – test covariates for heteroskedastic noise

This always returns Gaussian predictions, i.e.

Returns

- ys - predictive mean
- s2 - predictive variance

predict_and_adjust(*hyp*, *X*, *y*, *Xs=None*, *ys=None*, *var_groups_test=None*, *var_groups_adapt=None*, ***kwargs*)

Function to transfer the model to a new site. This is done by first making predictions on the adaptation data given by X, adjusting by the residuals with respect to y.

Parameters

- **hyp** – hyperparameter vector
- **X** – covariates for adaptation (i.e. calibration) data
- **y** – responses for adaptation data
- **Xs** – covariate data (for which predictions should be adjusted)
- **ys** – true response variables (to be adjusted)
- **var_groups_test** – variance groups (e.g. sites) for test data
- **var_groups_adapt** – variance groups for adaptation data

There are two possible ways of using this function, depending on whether ys or Xs is specified

If ys is specified, this is applied directly to the data, which is assumed to be in the input space (i.e. not warped). In this case the adjusted true data points are returned in the same space

Alternatively, Xs is specified, then the predictions are made and adjusted. In this case the predictive variance are returned in the warped (i.e. Gaussian) space.

This function needs to know which sites are associated with which data points, which provided by var_groups_xxx, which is a list or array of scalar ids .

class gp.**CovBase**(*x=None*)

Bases: `object`

Base class for covariance functions.

All covariance functions must define the following methods:

```
CovFunction.get_n_params()
CovFunction.cov()
CovFunction.xcov()
CovFunction.dcov()
```

abstract cov(*theta*, *x*, *z=None*)

Return the full covariance (or cross-covariance if z is given)

abstract dcov(*theta*, *x*, *i*)

Return the derivative of the covariance function with respect to the i-th hyperparameter

get_n_params()

Report the number of parameters required

class gp.**CovLin**(*x=None*)

Bases: *CovBase*

Linear covariance function (no hyperparameters)

cov(*theta*, *x*, *z=None*)

Return the full covariance (or cross-covariance if z is given)

dcov(*theta*, *x*, *i*)

Return the derivative of the covariance function with respect to the i-th hyperparameter

get_n_params()

Report the number of parameters required

class gp.**CovSqExp**(*x=None*)

Bases: *CovBase*

Ordinary squared exponential covariance function. The hyperparameters are:

```
theta = ( log(ell), log(sf) )
```

where ell is a lengthscale parameter and sf2 is the signal variance

cov(*theta*, *x*, *z=None*)

Return the full covariance (or cross-covariance if z is given)

dcov(*theta*, *x*, *i*)

Return the derivative of the covariance function with respect to the i-th hyperparameter

get_n_params()

Report the number of parameters required

class gp.**CovSqExpARD**(*x=None*)

Bases: *CovBase*

Squared exponential covariance function with ARD The hyperparameters are:

```
theta = (log(ell_1, ..., log_ell_D), log(sf))
```

where ell_i are lengthscale parameters and sf2 is the signal variance

cov(*theta*, *x*, *z=None*)

Return the full covariance (or cross-covariance if z is given)

dcov(*theta*, *x*, *i*)

Return the derivative of the covariance function with respect to the i-th hyperparameter

get_n_params()

Report the number of parameters required

class gp.**CovSum**(*x=None*, *covfuncnames=None*)

Bases: *CovBase*

Sum of covariance functions. These are passed in as a cell array and intialised automatically. For example:

```
C = CovSum(x,(CovLin, CovSqExpARD))
C = CovSum.cov(x, )
```

The hyperparameters are:

```
theta = ( log(ell_1, ..., log_ell_D), log(sf2) )
```

where ell_i are lengthscale parameters and sf2 is the signal variance

cov(*theta*, *x*, *z=None*)

Return the full covariance (or cross-covariance if z is given)

dcov(*theta*, *x*, *i*)

Return the derivative of the covariance function with respect to the i-th hyperparameter

get_n_params()

Report the number of parameters required

class gp.**GPR**(*hyp=None*, *covfunc=None*, *X=None*, *y=None*, *n_iter=100*, *tol=0.001*, *verbose=False*, *warp=None*)

Bases: `object`

Gaussian process regression

Estimation and prediction of Gaussian process regression models

Basic usage:

```
G = GPR()
hyp = B.estimate(hyp0, cov, X, y)
ys, ys2 = B.predict(hyp, cov, X, y, Xs)
```

where the variables are

Parameters

- **hyp** – vector of hyperparmaters
- **cov** – covariance function
- **X** – N x D data array
- **y** – 1D Array of targets (length N)
- **Xs** – Nte x D array of test cases
- **hyp0** – starting estimates for hyperparameter optimisation

Returns

- ys - predictive mean
- ys2 - predictive variance

The hyperparameters are:

```
hyp = ( log(sn), (cov function params) )  # hyp is a list or array
```

The implementation and notation follows Rasmussen and Williams (2006). As in the gpml toolbox, these parameters are estimated using conjugate gradient optimisation of the marginal likelihood. Note that there is no explicit mean function, thus the gpr routines are limited to modelling zero-mean processes.

Reference: C. Rasmussen and C. Williams (2006) Gaussian Processes for Machine Learning

Written by A. Marquand

dloglik(*hyp*, *covfunc*, *X*, *y*)

Function to compute derivatives

estimate(*hyp0*, *covfunc*, *X*, *y*, *optimizer='cg'*)

Function to estimate the model

loglik(*hyp*, *covfunc*, *X*, *y*)

Function to compute compute log (marginal) likelihood

post(*hyp*, *covfunc*, *X*, *y*)

Generic function to compute posterior distribution.

predict(*hyp*, *X*, *y*, *Xs*)

Function to make predictions from the model

normative_parallel.**bashwrap_nm**(*processing_dir*, *python_path*, *normative_path*, *job_name*, *covfile_path*, *respfile_path*, *func='estimate'*, ***kwargs*)

This function wraps normative modelling into a bash script to run it on a torque cluster system.

Basic usage:

```
bashwrap_nm(processing_dir, python_path, normative_path, job_name, covfile_path,␣
↪respfile_path)
```

Parameters

- **processing_dir** – Full path to the processing dir
- **python_path** – Full path to the python distribution
- **normative_path** – Full path to the normative.py
- **job_name** – Name for the bash script that is the output of this function
- **covfile_path** – Full path to a .txt file that contains all covariates (subjects x covariates) for the responsefile
- **respfile_path** – Full path to a .txt that contains all features (subjects x features)
- **cv_folds** – Number of cross validations
- **testcovfile_path** – Full path to a .txt file that contains all covariates (subjects x covariates) for the testresponse file
- **testrespfile_path** – Full path to a .txt file that contains all test features
- **alg** – which algorithm to use
- **configparam** – configuration parameters for this algorithm

Outputs

A bash.sh file containing the commands for normative modelling saved to the processing directory (written to disk).

written by (primarily) T Wolfers, (adapted) S Rutherford.

normative_parallel.**check_job_status**(*jobs*)

A utility function to count the tasks with different status.

Parameters

jobs – List of job ids.

Returns

returns the number of taks athat are queued, running, completed etc

normative_parallel.**check_jobs**(*jobs*, *delay=60*)

A utility function for chacking the status of submitted jobs.

Parameters

- **jobs** – list of job ids.
- **delay** – the delay (in sec) between two consequative checks, defaults to 60.

normative_parallel.**collect_nm**(*processing_dir*, *job_name*, *func='estimate'*, *collect=False*, *binary=False*, *batch_size=None*, *outputsuffix='_estimate'*)

Function to checks and collects all batches.

Basic usage:

```
collect_nm(processing_dir, job_name)
```

Parameters

- **processing_dir** – Full path to the processing directory
- **collect** – If True data is checked for failed batches and collected; if False data is just checked
- **binary** – Results in pkl format

Outputs

Text or pkl files containing all results accross all batches the combined output (written to disk).

Returns 0

if batches fail

Returns 1

if bathches complete successfully

written by (primarily) T Wolfers, (adapted) SM Kia, (adapted) S Rutherford.

normative_parallel.**delete_nm**(*processing_dir*, *binary=False*)

This function deletes all processing for normative modelling and just keeps the combined output.

Basic usage:

```
collect_nm(processing_dir)
```

Parameters

- **processing_dir** – Full path to the processing directory.
- **binary** – Results in pkl format.

written by (primarily) T Wolfers, (adapted) SM Kia, (adapted) S Rutherford.

normative_parallel.**execute_nm**(*processing_dir*, *python_path*, *job_name*, *covfile_path*, *respfile_path*, *batch_size*, *memory*, *duration*, *normative_path=None*, *func='estimate'*, *interactive=False*, ***kwargs*)

Execute parallel normative models This function is a mother function that executes all parallel normative modelling routines. Different specifications are possible using the sub- functions.

Basic usage:

```
execute_nm(processing_dir, python_path, job_name, covfile_path, respfile_path,␣
↪batch_size, memory, duration)
```

Parameters

- **processing_dir** – Full path to the processing dir
- **python_path** – Full path to the python distribution
- **normative_path** – Full path to the normative.py. If None (default) then it will automatically retrieves the path from the installed packeage.
- **job_name** – Name for the bash script that is the output of this function
- **covfile_path** – Full path to a .txt file that contains all covariats (subjects x covariates) for the responsefile
- **respfile_path** – Full path to a .txt that contains all features (subjects x features)
- **batch_size** – Number of features in each batch
- **memory** – Memory requirements written as string for example 4gb or 500mb
- **duation** – The approximate duration of the job, a string with HH:MM:SS for example 01:01:01
- **cv_folds** – Number of cross validations
- **testcovfile_path** – Full path to a .txt file that contains all covariates (subjects x covariates) for the test response file
- **testrespfile_path** – Full path to a .txt file that contains all test features
- **log_path** – Path for saving log files
- **binary** – If True uses binary format for response file otherwise it is text
- **interactive** – If False (default) the user should manually rerun the failed jobs or collect the results. If 'auto' the job status are checked until all jobs are completed then the failed jobs are rerun and the results are automaticallu collectted. Using 'query' is similar to 'auto' unless it asks for user verification thius is immune to endless loop in the case of bugs in the code.

written by (primarily) T Wolfers, (adapted) SM Kia The documentation is adapated by S Rutherford.

normative_parallel.**qsub_nm**(*job_path*, *log_path*, *memory*, *duration*)

This function submits a job.sh scipt to the torque custer using the qsub command.

Basic usage:

```
qsub_nm(job_path, log_path, memory, duration)
```

Parameters

- **job_path** – Full path to the job.sh file.
- **memory** – Memory requirements written as string for example 4gb or 500mb.
- **duation** – The approximate duration of the job, a string with HH:MM:SS for example 01:01:01.

Outputs

Submission of the job to the (torque) cluster.

written by (primarily) T Wolfers, (adapted) SM Kia, (adapted) S Rutherford.

normative_parallel.**rerun_nm**(*processing_dir*, *log_path*, *memory*, *duration*, *binary=False*, *interactive=False*)

This function reruns all failed batched in processing_dir after collect_nm has identified the failed batches. Basic usage:

```
rerun_nm(processing_dir, log_path, memory, duration)
```

Parameters

- **processing_dir** – Full path to the processing directory
- **memory** – Memory requirements written as string for example 4gb or 500mb.
- **duration** – The approximate duration of the job, a string with HH:MM:SS for example 01:01:01.

written by (primarily) T Wolfers, (adapted) SM Kia, (adapted) S Rutherford.

normative_parallel.**retrieve_jobs**()

A utility function to retrieve task status from the outputs of qstat.

Returns
a dictionary of jobs.

normative_parallel.**sbatch_nm**(*job_path*, *log_path*)

This function submits a job.sh scipt to the torque custer using the qsub command.

Basic usage:

```
sbatch_nm(job_path, log_path)
```

Parameters

- **job_path** – Full path to the job.sh file
- **log_path** – The logs are currently stored in the working dir

Outputs
Submission of the job to the (torque) cluster.

written by (primarily) T Wolfers, (adapted) S Rutherford.

normative_parallel.**sbatchrerun_nm**(*processing_dir*, *memory*, *duration*, *new_memory=False*, *new_duration=False*, *binary=False*, ***kwargs*)

This function reruns all failed batched in processing_dir after collect_nm has identified he failed batches.

Basic usage:

```
rerun_nm(processing_dir, memory, duration)
```

Parameters

- **processing_dir** – Full path to the processing directory.
- **memory** – Memory requirements written as string, for example 4gb or 500mb.
- **duration** – The approximate duration of the job, a string with HH:MM:SS for example 01:01:01.
- **new_memory** – If you want to change the memory you have to indicate it here.
- **new_duration** – If you want to change the duration you have to indicate it here.

Outputs

Re-runs failed batches.

written by (primarily) T Wolfers, (adapted) S Rutherford.

normative_parallel.**sbatchwrap_nm**(*processing_dir*, *python_path*, *normative_path*, *job_name*, *covfile_path*, *respfile_path*, *memory*, *duration*, *func='estimate'*, ***kwargs*)

This function wraps normative modelling into a bash script to run it on a torque cluster system.

Basic usage:

```
sbatchwrap_nm(processing_dir, python_path, normative_path, job_name, covfile_path,␣
↪respfile_path, memory, duration)
```

Parameters

- **processing_dir** – Full path to the processing dir
- **python_path** – Full path to the python distribution
- **normative_path** – Full path to the normative.py
- **job_name** – Name for the bash script that is the output of this function
- **covfile_path** – Full path to a .txt file that contains all covariates (subjects x covariates) for the responsefile
- **respfile_path** – Full path to a .txt that contains all features (subjects x features)
- **cv_folds** – Number of cross validations
- **testcovfile_path** – Full path to a .txt file that contains all covariates (subjects x covariates) for the testresponse file
- **testrespfile_path** – Full path to a .txt file that contains all test features
- **alg** – which algorithm to use
- **configparam** – configuration parameters for this algorithm

Outputs

A bash.sh file containing the commands for normative modelling saved to the processing directory (written to disk).

written by (primarily) T Wolfers, (adapted) S Rutherford

normative_parallel.**split_nm**(*processing_dir*, *respfile_path*, *batch_size*, *binary*, ***kwargs*)

This function prepares the input files for normative_parallel.

Basic usage:

```
split_nm(processing_dir, respfile_path, batch_size, binary, testrespfile_path)
```

Parameters

- **processing_dir** – Full path to the processing dir
- **respfile_path** – Full path to the responsefile.txt (subjects x features)
- **batch_size** – Number of features in each batch
- **testrespfile_path** – Full path to the test responsefile.txt (subjects x features)

- **binary** – If True binary file

Outputs

The creation of a folder struture for batch-wise processing.

witten by (primarily) T Wolfers (adapted) SM Kia, (adapted) S Rutherford.

trendsurf.**create_basis**(*X*, *basis*, *mask*)

Create a basis set

This will create a basis set for the trend surface model. This is currently fit using a polynomial model of a specified degree. The models are estimated on the basis of data stored on disk in ascii or neuroimaging data formats (currently nifti only). Ascii data should be in tab or space delimited format with the number of voxels in rows and the number of subjects in columns. Neuroimaging data will be reshaped into the appropriate format

Parameters

- **X** – covariates
- **basis** – model order for the interpolating polynomial
- **mask** – mask used to apply to the data

Returns

- Phi - basis set

trendsurf.**estimate**(*filename*, *maskfile*, *basis*, *ard=False*, *outputall=False*, *saveoutput=True*, ***kwargs*)

Estimate a trend surface model

This will estimate a trend surface model, independently for each subject. This is currently fit using a polynomial model of a specified degree. The models are estimated on the basis of data stored on disk in ascii or neuroimaging data formats (currently nifti only). Ascii data should be in tab or space delimited format with the number of voxels in rows and the number of subjects in columns. Neuroimaging data will be reshaped into the appropriate format

Basic usage:

```
estimate(filename, maskfile, basis)
```

where the variables are defined below. Note that either the cfolds parameter or (testcov, testresp) should be specified, but not both.

Parameters

- **filename** – 4-d nifti file containing the images to be estimated
- **maskfile** – nifti mask used to apply to the data
- **basis** – model order for the interpolating polynomial

All outputs are written to disk in the same format as the input. These are:

Outputs

- yhat - predictive mean
- ys2 - predictive variance
- trendcoeff - coefficients from the trend surface model
- negloglik - Negative log marginal likelihood
- hyp - hyperparameters
- explainedvar - explained variance

- rmse - standardised mean squared error

trendsurf.**get_args**(**args*)

Parse command line arguments

This will parse the command line arguments for the trend surface model. The arguments are:

Parameters

- **filename** – 4-d nifti file containing the images to be estimated
- **maskfile** – nifti mask used to apply to the data
- **basis** – model order for the interpolating polynomial
- **covfile** – file containing covariates
- **ard** – use ARD
- **outputall** – output all measures

Returns

- filename - 4-d nifti file containing the images to be estimated
- maskfile - nifti mask used to apply to the data
- basis - model order for the interpolating polynomial
- covfile - file containing covariates
- ard - use ARD
- outputall - output all measures

trendsurf.**load_data**(*datafile*, *maskfile=None*)

Load data from disk

This will load data from disk, either in nifti or ascii format. If the data are in ascii format, they should be in tab or space delimited format with the number of voxels in rows and the number of subjects in columns. Neuroimaging data will be reshaped into the appropriate format

Parameters

- **datafile** – 4-d nifti file containing the images to be estimated
- **maskfile** – nifti mask used to apply to the data

Returns

- dat - data in vectorised form
- world - voxel coordinates
- mask - mask used to apply to the data

trendsurf.**main**(**args*)

trendsurf.**write_nii**(*data*, *filename*, *examplenii*, *mask*)

Write data to nifti file

This will write data to a nifti file, using the header information from an example nifti file.

Parameters

- **data** – data to be written
- **filename** – name of file to be written

- **examplenii** – example nifti file
- **mask** – mask used to apply to the data

Returns

- Phi - basis set

class rfa.**GPRRFA**(*hyp=None*, *X=None*, *y=None*, *n_feat=None*, *n_iter=100*, *tol=0.001*, *verbose=False*)

Bases: `object`

Random Feature Approximation for Gaussian Process Regression

Estimation and prediction of Bayesian linear regression models

Basic usage:

```
R = GPRRFA()
hyp = R.estimate(hyp0, X, y)
ys,s2 = R.predict(hyp, X, y, Xs)
```

where the variables are

Parameters

- **hyp** – vector of hyperparmaters.
- **X** – N x D data array
- **y** – 1D Array of targets (length N)
- **Xs** – Nte x D array of test cases
- **hyp0** – starting estimates for hyperparameter optimisation

Returns

- ys - predictive mean
- s2 - predictive variance

The hyperparameters are:

```
hyp = [ log(sn), log(ell), log(sf) ]  # hyp is a numpy array
```

where sn^2 is the noise variance, ell are lengthscale parameters and sf^2 is the signal variance. This provides an approximation to the covariance function:

```
k(x,z) = x'*z + sn2*exp(0.5*(x-z)'*Lambda*(x-z))
```

where Lambda = diag((ell_1^2, … ell_D^2))

Written by A. Marquand

dloglik(*hyp*, *X*, *y*)

Function to compute derivatives

estimate(*hyp0*, *X*, *y*, *optimizer='lbfgs'*)

Function to estimate the model

get_n_params(*X*)

loglik(*hyp*, *X*, *y*)

Function to compute compute log (marginal) likelihood

post(*hyp*, *X*, *y*)

Generic function to compute posterior distribution.

This function will save the posterior mean and precision matrix as self.m and self.A and will also update internal parameters (e.g. N, D and the prior covariance (Sigma) and precision (iSigma).

predict(*hyp*, *X*, *y*, *Xs*)

Function to make predictions from the model

fileio.**alphanum_key**(*s*)

Turn a string into a list of numbers

Basic usage:

```
alphanum_key(s)
```

Parameters

s – string to convert

fileio.**create_mask**(*data_array*, *mask*, *verbose=False*)

Create a mask from a data array or a nifti file

Basic usage:

```
create_mask(data_array, mask, verbose)
```

Parameters

- **data_array** – numpy array containing the data to write out
- **mask** – nifti image containing a mask for the image
- **verbose** – verbose output

fileio.**file_extension**(*filename*)

Determine the file extension of a file (e.g. .nii.gz)

Basic usage:

```
file_extension(filename)
```

Parameters

filename – name of the file to check

fileio.**file_stem**(*filename*)

Determine the file stem of a file (e.g. /path/to/file.nii.gz -> file)

Basic usage:

```
file_stem(filename)
```

Parameters

filename – name of the file to check

fileio.**file_type**(*filename*)

Determine the file type of a file

Basic usage:

```
file_type(filename)
```

Parameters
filename – name of the file to check

fileio.**load**(*filename*, *mask=None*, *text=False*, *vol=True*)

Load a numpy array from a file

Basic usage:

```
load(filename, mask, text, vol)
```

Parameters

- **filename** – name of the file to load
- **mask** – nifti image containing a mask for the image
- **text** – whether to write out a text file
- **vol** – whether to load the image as a volume

fileio.**load_ascii**(*filename*)

Load an ascii file into a numpy array

Basic usage:

```
load_ascii(filename)
```

Parameters
filename – name of the file to load

fileio.**load_cifti**(*filename*, *vol=False*, *mask=None*, *rmtmp=True*)

Load a cifti file into a numpy array

Basic usage:

```
load_cifti(filename, vol, mask, rmtmp)
```

Parameters

- **filename** – name of the file to load
- **vol** – whether to load the image as a volume
- **mask** – nifti image containing a mask for the image
- **rmtmp** – whether to remove temporary files

fileio.**load_nifti**(*datafile*, *mask=None*, *vol=False*, *verbose=False*)

Load a nifti file into a numpy array

Basic usage:

```
load_nifti(datafile, mask, vol, verbose)
```

Parameters

- **datafile** – name of the file to load
- **mask** – nifti image containing a mask for the image
- **vol** – whether to load the image as a volume
- **verbose** – verbose output

fileio.**load_pd**(*filename*)

Load a csv file into a pandas dataframe

Basic usage:

```
load_pd(filename)
```

Parameters

filename – name of the file to load

fileio.**predictive_interval**(*s2_forward*, *cov_forward*, *multiplicator*)

Calculates a predictive interval for the forward model

fileio.**save**(*data*, *filename*, *example=None*, *mask=None*, *text=False*, *dtype=None*)

Save a numpy array to a file

Basic usage:

```
save(data, filename, example, mask, text, dtype)
```

Parameters

- **data** – numpy array containing the data to write out
- **filename** – where to store it
- **example** – example file to copy the geometry from
- **mask** – nifti image containing a mask for the image
- **text** – whether to write out a text file
- **dtype** – data type for the output image (if different from the image)

fileio.**save_ascii**(*data*, *filename*)

Save a numpy array to an ascii file

Basic usage:

```
save_ascii(data, filename)
```

Parameters

- **data** – numpy array containing the data to write out
- **filename** – where to store it

fileio.**save_cifti**(*data*, *filename*, *example*, *mask=None*, *vol=True*, *volatlas=None*)

Save a cifti file from a numpy array

Basic usage:

```
save_cifti(data, filename, example, mask, vol, volatlas)
```

Parameters

- **data** – numpy array containing the data to write out
- **filename** – where to store it
- **example** – example file to copy the geometry from
- **mask** – nifti image containing a mask for the image
- **vol** – whether to load the image as a volume
- **volatlas** – atlas to use for the volume

fileio.**save_nifti**(*data*, *filename*, *examplenii*, *mask*, *dtype=None*)

Write output to nifti

Basic usage:

```
save_nifti(data, filename mask, dtype)
```

Parameters

- **data** – numpy array containing the data to write out
- **filename** – where to store it
- **examplenii** – nifti to copy the geometry and data type from
- **dtype** – data type for the output image (if different from the image)

Mask
nifti image containing a mask for the image

fileio.**save_pd**(*data*, *filename*)

Save a pandas dataframe to a csv file

Basic usage:

```
save_pd(data, filename)
```

Parameters

- **data** – pandas dataframe containing the data to write out
- **filename** – where to store it

fileio.**sort_nicely**(*l*)

Sort a list of strings in a natural way

Basic usage:

```
sort_nicely(l)
```

Parameters
l – list of strings to sort

fileio.**tryint**(*s*)

Try to convert a string to an integer

Basic usage:

```
tryint(s)
```

Parameters
s – string to convert

fileio.**vol2vec**(*dat*, *mask*, *verbose=False*)

Vectorise a 3d image

Basic usage:

```
vol2vec(dat, mask, verbose)
```

Parameters

- **dat** – numpy array containing the data to write out
- **mask** – nifti image containing a mask for the image
- **verbose** – verbose output

CHAPTER

FOUR

GAUSSIAN PROCESS REGRESSION

Created by

Mariam Zabihi @m_zabihi

Saige Rutherford @being_saige

Thomas Wolfers @ThomasWolfers

4.1 Background Story

Morten and Ingrid are concerned about the health of their father, Nordan. He recently turned 65 years. A few months ago he could not find his way home. Together, they visit a neurologist/psychiatrist to conduct a number of cognitive tests. However, those tests were inconclusive. While Nordan has a relatively low IQ it could not explain his trouble returning home.

Recently, the family heard about a new screening technique called normative modeling with which one can place individuals in reference to a population norm on for instance measures such as brain volume. Nordan would like to undertake this procedure to better know what is going on and to potentially find targets for treatment. Therefore, the family booked an appointment with you, the normative modeling specialist. To find out what is going on you compare Nordan's hyppocampus to the norm and to a group of persons with Dementia disorders, who have a similar IQ, age as well as the same sex as Nordan.

Do your best to get as far as you can. However, you do not need to feel bad if you cannot complete everything during the tutorial.

4.2 Task 0: Load data and install the pcntoolkit

```
#install normative modeling
! pip install pcntoolkit==0.26
```

Option 1: Connect your Google Drive account, and load data from Google Drive. Having Google Drive connected will allow you to save any files created back to your Drive folder. This step will require you to download the csv files from Github to your computer, and then make a folder in your Google Drive account and upload the csv files to this folder.

```
from google.colab import drive
drive.mount('/content/drive')
```

(continues on next page)

(continued from previous page)

```
#change dir to data on your google drive
import os
os.chdir('drive/My Drive/name-of-folder-where-you-uploaded-csv-files-from-Github/')
 ↪#Change this path to match the path to your data in Google Drive

# code by T. Wolfers
```

Option 2: Import the files directly from Github, and skip adding them to Google Drive.

```
!wget -nc https://raw.githubusercontent.com/predictive-clinical-neuroscience/PCNtoolkit-
 ↪demo/master/tutorials/CPC_2020/data/camcan_demographics.csv
!wget -nc https://raw.githubusercontent.com/predictive-clinical-neuroscience/PCNtoolkit-
 ↪demo/master/tutorials/CPC_2020/data/camcan_demographics_nordan.csv
!wget -nc https://raw.githubusercontent.com/predictive-clinical-neuroscience/PCNtoolkit-
 ↪demo/master/tutorials/CPC_2020/data/camcan_features.csv
!wget -nc https://raw.githubusercontent.com/predictive-clinical-neuroscience/PCNtoolkit-
 ↪demo/master/tutorials/CPC_2020/data/camcan_features_nordan.csv

# code by S. Rutherford
```

```
--2022-02-17 15:03:58--  https://raw.githubusercontent.com/predictive-clinical-
 ↪neuroscience/PCNtoolkit-demo/master/tutorials/CPC_2020/data/camcan_demographics.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.
 ↪199.108.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443.
 ↪.. connected.
HTTP request sent, awaiting response... 200 OK
Length: 17484 (17K) [text/plain]
Saving to: ‘camcan_demographics.csv’

camcan_demographics 100%[===================>]  17.07K  --.-KB/s    in 0.001s

2022-02-17 15:03:58 (12.9 MB/s) - ‘camcan_demographics.csv’ saved [17484/17484]

--2022-02-17 15:03:58--  https://raw.githubusercontent.com/predictive-clinical-
 ↪neuroscience/PCNtoolkit-demo/master/tutorials/CPC_2020/data/camcan_demographics_nordan.
 ↪csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.
 ↪199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443.
 ↪.. connected.
HTTP request sent, awaiting response... 200 OK
Length: 332 [text/plain]
Saving to: ‘camcan_demographics_nordan.csv’

camcan_demographics 100%[===================>]     332  --.-KB/s    in 0s

2022-02-17 15:03:58 (15.5 MB/s) - ‘camcan_demographics_nordan.csv’ saved [332/332]

--2022-02-17 15:03:58--  https://raw.githubusercontent.com/predictive-clinical-
 ↪neuroscience/PCNtoolkit-demo/master/tutorials/CPC_2020/data/camcan_features.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.
```

(continues on next page)

(continued from previous page)

```
↪199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443.
↪.. connected.
HTTP request sent, awaiting response... 200 OK
Length: 188944 (185K) [text/plain]
Saving to: ‘camcan_features.csv’

camcan_features.csv 100%[===================>] 184.52K  --.-KB/s    in 0.05s

2022-02-17 15:03:58 (3.88 MB/s) - ‘camcan_features.csv’ saved [188944/188944]

--2022-02-17 15:03:58--  https://raw.githubusercontent.com/predictive-clinical-
↪neuroscience/PCNtoolkit-demo/master/tutorials/CPC_2020/data/camcan_features_nordan.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.
↪199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443.
↪.. connected.
HTTP request sent, awaiting response... 200 OK
Length: 1695 (1.7K) [text/plain]
Saving to: ‘camcan_features_nordan.csv’

camcan_features_nor 100%[===================>]   1.66K  --.-KB/s    in 0s

2022-02-17 15:03:59 (25.3 MB/s) - ‘camcan_features_nordan.csv’ saved [1695/1695]
```

4.3 TASK 1: Format input data

You have four files. The features and demographics file for the normsample and two files of the same name for Nordan your test sample. As one of your coworkers has done the preporcessing and quality control there are more subjects in the demographics file than in the features file of the norm sample. Please select the overlap of participants between those two files.

Question for your understanding:

1) Why do we have to select the overlap between participants in terms of featrues and demographics?

```
import pandas as pd

# read in the files.
norm_demographics = pd.read_csv('camcan_demographics.csv',
                                sep= ",",
                                index_col = 0)
norm_features = pd.read_csv('camcan_features.csv',
                            sep=",",
                            index_col = 0)

# check columns through print [there are other better options]
print(norm_demographics)
print(norm_features)

# find overlap in terms of participants between norm_sample_features and
```

(continues on next page)

(continued from previous page)

```
# norm_sample_demographics

norm_demographics_features = pd.concat([norm_demographics, norm_features],
                                       axis = 1,
                                       join = 'inner') # inner checks overlap
                                                       # outer combines
print(norm_demographics_features)

# code by T. Wolfers
```

```
             age sex_name  sex  IQ_random
paricipants
CC110033      24     MALE    1         73
CC110037      18     MALE    1        103
CC110045      24   FEMALE    0        124
CC110056      22   FEMALE    0        124
CC110062      20     MALE    1        126
...          ...      ...  ...        ...
CC722542      79     MALE    1        116
CC722651      79   FEMALE    0        128
CC722891      84   FEMALE    0        129
CC723197      80   FEMALE    0         96
CC723395      86   FEMALE    0        145

[707 rows x 4 columns]
             left_Hippocampal_tail  ...  right_Whole_hippocampus
participants                        ...
CC110033                482.768229  ...              3531.764896
CC110037                595.269259  ...              3835.426137
CC110045                655.847194  ...              3681.494304
CC110056                561.345626  ...              3461.373764
CC110062                756.521166  ...              4782.407821
...                            ...  ...                      ...
CC722542                467.896808  ...              3284.108783
CC722651                406.326167  ...              3210.272905
CC722891                393.430481  ...              2423.675065
CC723197                475.929914  ...              3043.146264
CC723395                444.301617  ...              2988.001288

[651 rows x 26 columns]
          age sex_name  sex  ...  right_fimbria  right_HATA  right_Whole_hippocampus
CC110033   24     MALE    1  ...      87.127463   73.589184              3531.764896
CC110037   18     MALE    1  ...      99.657823   60.920924              3835.426137
CC110045   24   FEMALE    0  ...      69.436808   59.323542              3681.494304
CC110056   22   FEMALE    0  ...      60.505521   51.726283              3461.373764
CC110062   20     MALE    1  ...      92.215816   85.484454              4782.407821
...       ...      ...  ...  ...            ...         ...                      ...
CC722542   79     MALE    1  ...      46.144212   43.966509              3284.108783
CC722651   79   FEMALE    0  ...      68.730322   59.699644              3210.272905
CC722891   84   FEMALE    0  ...      27.913196   38.629828              2423.675065
CC723197   80   FEMALE    0  ...      51.893458   65.474967              3043.146264
CC723395   86   FEMALE    0  ...      68.335159   62.081225              2988.001288
```

(continues on next page)

(continued from previous page)

```
[650 rows x 30 columns]
```

4.4 TASK 2: Prepare the covariate_normsample and testresponse_normsample file.

As mentioned in the introductory presentation those files need a specific format and the entries need to be seperated by spaces. Use whatever method you know to prepare those files based on the data provided in TASK 1. Save those files in .txt format in your drive. Also get rid of the column names and participant IDs.

Given that we only have limited time in this practical we have to make a selection for the features based on your prior knowledge. With the information in mind that Nordan does not remember his way home, which subfield of the hyppocampus is probably a good target for the investigations? Select a maximum of four hyppocampal regions as features.

NOTE: Normative modeling is a screening tool we just make this selection due to time constraints, in reality we build these models on millions of putative biomarkers that are not restricted to brain imaging.

Qestions for your understanding:

2) What is the requirement for the features in terms of variable properties (e.g. dicotomous or continous)? 3) What is the requirement for the covariates in terms of these properties? 4) What are the requirements for both together? 5) How does this depent on the algorithm used?

```
# perpare covariate_normsample for sex and age
covariate_normsample = norm_demographics_features[['sex',
                                                   'age']]

covariate_normsample.to_csv('covariate_normsample.txt',
                            sep = ' ',
                            header = False,
                            index = False)

# perpare features_normsample for relevant hyppocampal subfields
features_normsample = norm_demographics_features[['left_CA1',
                                                  'left_CA3',
                                                  'right_CA1',
                                                  'right_CA3']]

features_normsample.to_csv('features_normsample.txt',
                           sep = ' ',
                           header = False,
                           index = False)

# code by T. Wolfers
```

4.5 TASK 3: Estimate normative model

Once you have prepared and saved all the necessary files. Look at the pcntoolkit for running normative modeling. Select an appropritate method set up the toolkit and run your analyses using 2-fold cross validation in the normsample. Change the output suffix from estimate to '_2fold'.

HINT: You primarily need the estimate function.

SUGGESTION: While this process is running you can go to the next TASK 4, you will have no doubt when it is correctly running.

Question for your understaning:

6) What does cvfolds mean and why do we use it? 7) What is the output of the estimate function and what does it mean?

```
import pcntoolkit as pcn

# run normative modeling using 2-fold cross-validation

pcn.normative.estimate(covfile = 'covariate_normsample.txt',
                       respfile = 'features_normsample.txt',
                       cvfolds = 2,
                       alg = 'gpr',
                       outputsuffix = '_2fold')

# code by T. Wolfers
```

```
Processing data in features_normsample.txt
Estimating model  1 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 1856.502251
         Iterations: 40
         Function evaluations: 99
         Gradient evaluations: 99
Estimating model  2 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 1596.239263
         Iterations: 42
         Function evaluations: 93
         Gradient evaluations: 93
Estimating model  3 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
```

(continues on next page)

(continued from previous page)

```
Optimization terminated successfully.
         Current function value: 1862.316698
         Iterations: 47
         Function evaluations: 104
         Gradient evaluations: 104
Estimating model  4 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 1587.950935
         Iterations: 30
         Function evaluations: 64
         Gradient evaluations: 64
Estimating model  1 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 1916.461484
         Iterations: 44
         Function evaluations: 94
         Gradient evaluations: 87
Estimating model  2 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 1611.661888
         Iterations: 34
         Function evaluations: 85
         Gradient evaluations: 85
Estimating model  3 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 1912.665851
         Iterations: 61
         Function evaluations: 133
         Gradient evaluations: 126
Estimating model  4 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
```

(continues on next page)

(continued from previous page)

```
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 1619.045647
         Iterations: 43
         Function evaluations: 110
         Gradient evaluations: 105
Evaluating the model ...
Writing outputs ...
```

4.6 TASK 4: Estimate the forward model of the normative model

In order to visulize the normative trajectories you first need to run the forward model. To this end you need to set up an appropriate covariate_forwardmodel file that covers the age range appropriately for both sexes. Save this file as .txt . Then you can input the files you made in TASK 1 as well as the file you made now and run the forward model using the appropriate specifications.

Question for your understaning:

8) What is yhat and ys2? 9) Why does the output of the forward model does not inlcude the Z-scores?

```
# create covariate_forwardmodel.txt file
covariate_forwardmodel = {'sex': [0, 0, 0, 0, 0, 0, 0,
                                  1, 1, 1, 1, 1, 1, 1],
                          'age': [20, 30, 40, 50, 60, 70, 80,
                                  20, 30, 40, 50, 60, 70, 80]}
covariate_forwardmodel = pd.DataFrame(data=covariate_forwardmodel)

covariate_forwardmodel.to_csv('covariate_forwardmodel.txt',
                           sep = ' ',
                           header = False,
                           index = False)

# estimate forward model
pcn.normative.estimate(covfile = 'covariate_normsample.txt',
                       respfile = 'features_normsample.txt',
                       testcov = 'covariate_forwardmodel.txt',
                       cvfolds = None,
                       alg = 'gpr',
                       outputsuffix = '_forward')

# code by T. Wolfers
```

```
Processing data in features_normsample.txt
Estimating model  1 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
```

(continues on next page)

(continued from previous page)

```
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3781.497401
         Iterations: 20
         Function evaluations: 61
         Gradient evaluations: 54
Estimating model  2 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3201.761309
         Iterations: 39
         Function evaluations: 108
         Gradient evaluations: 108
Estimating model  3 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3771.310488
         Iterations: 47
         Function evaluations: 181
         Gradient evaluations: 167
Estimating model  4 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3200.837262
         Iterations: 40
         Function evaluations: 104
```

(continues on next page)

(continued from previous page)

```
         Gradient evaluations: 104
Writing outputs ...
```

4.7 TASK 5: Visualize forward model

Visualize the forward model of the normative model similar to the figure below.

Fig. 1: 1-s2.0-S2451902218303291X-gr2.jpg

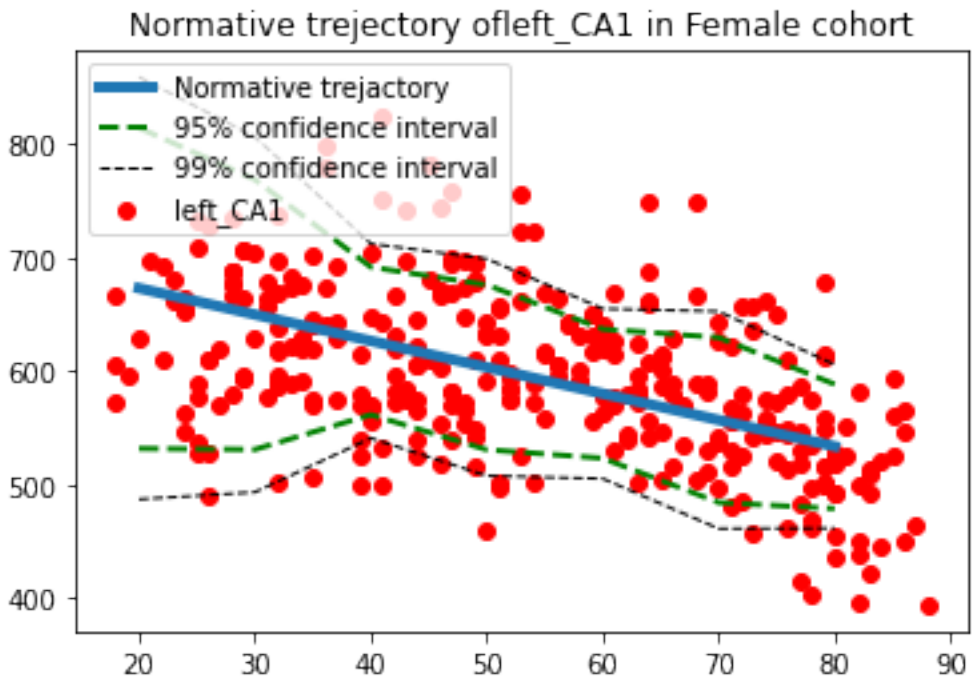
HINT: First create a function that calculates the confidence intervals and then plot yhat, y2 of the forward model. Finally, plot the data of individual participants.

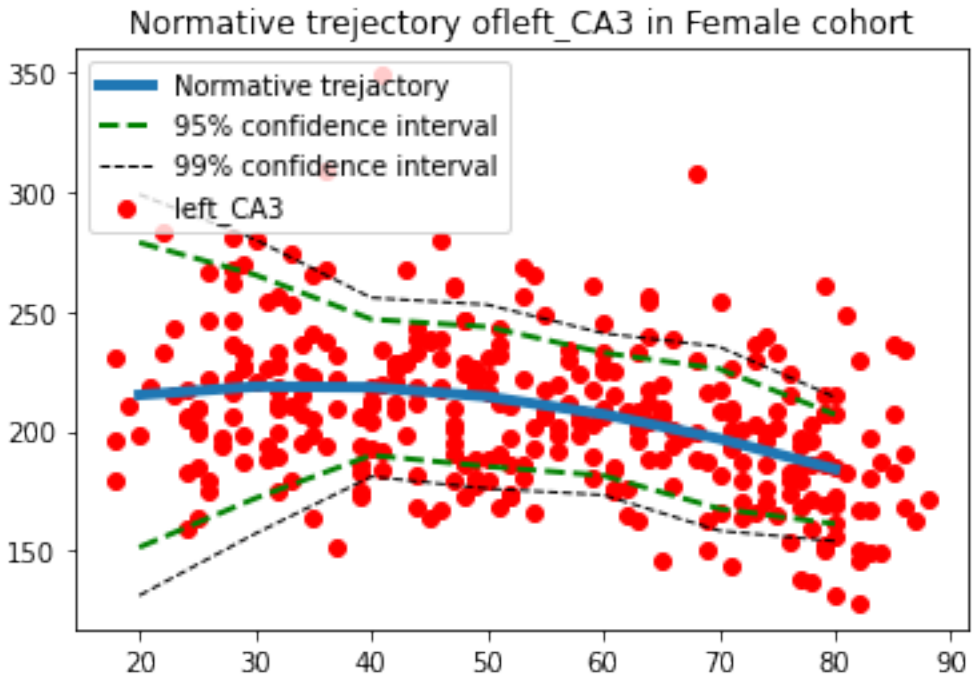
```
import numpy as np
import matplotlib.pyplot as plt

# confidence interval calculation at x_forward
def confidence_interval(s2,x,z):
  CI=np.zeros((len(x_forward),4))
  for i,xdot in enumerate(x_forward):
    ci_inx=np.isin(x,xdot)
    S2=s2[ci_inx]
    S_hat=np.mean(S2,axis=0)
    n=S2.shape[0]
    CI[i,:]=z*np.power(S_hat/n,.5)
  return CI


feature_names=['left_CA1','left_CA3','right_CA1','right_CA3']
sex_covariates=[ 'Female','Male']
# Creating plots for Female and male
for i,sex in enumerate(sex_covariates):
#forward model data
    forward_yhat = pd.read_csv('yhat_forward.txt', sep = ' ', header=None)
    yhat_forward=forward_yhat.values
    yhat_forward=yhat_forward[7*i:7*(i+1)]
    x_forward=[20, 30, 40, 50, 60, 70, 80]

# Find the index of the data exclusively for one sex. Female:0, Male: 1
    inx=np.where(covariate_normsample.sex==i)[0]
    x=covariate_normsample.values[inx,1]
# actual data
    y = pd.read_csv('features_normsample.txt', sep = ' ', header=None)
```

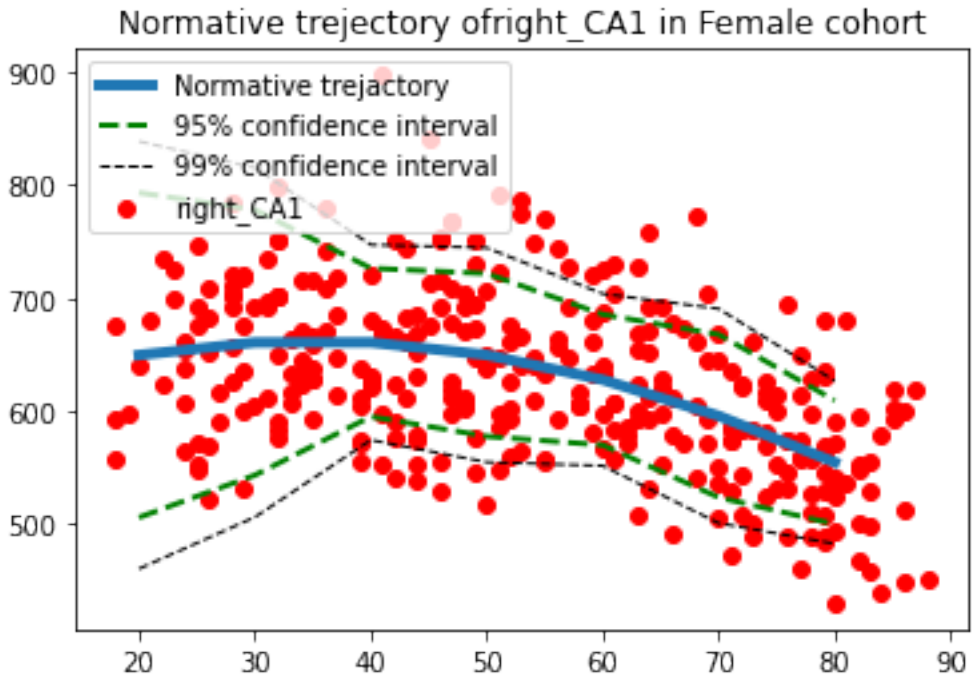
(continues on next page)

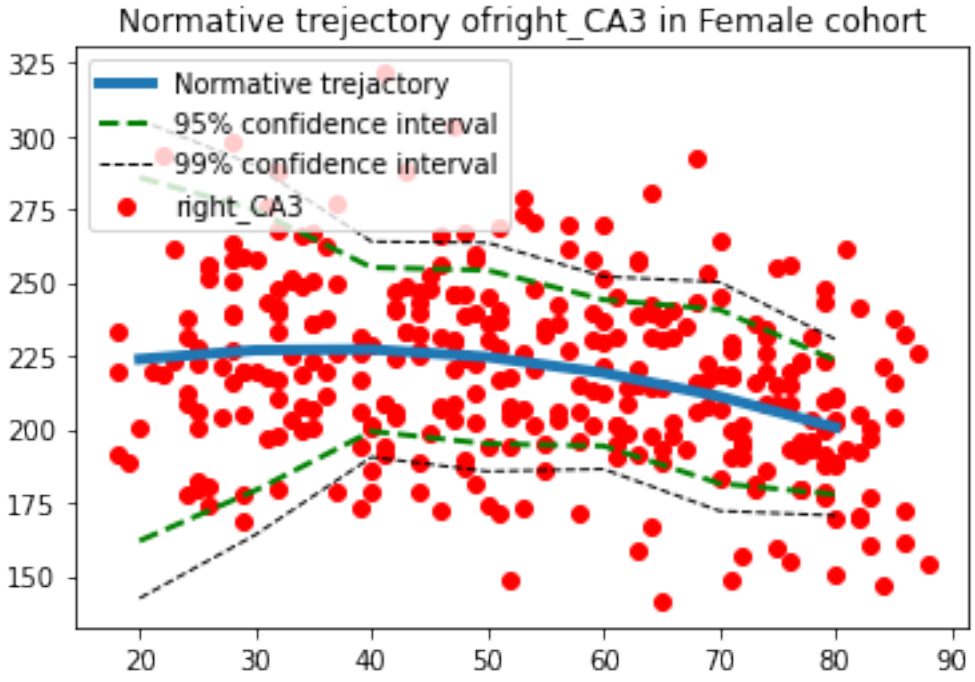
(continued from previous page)

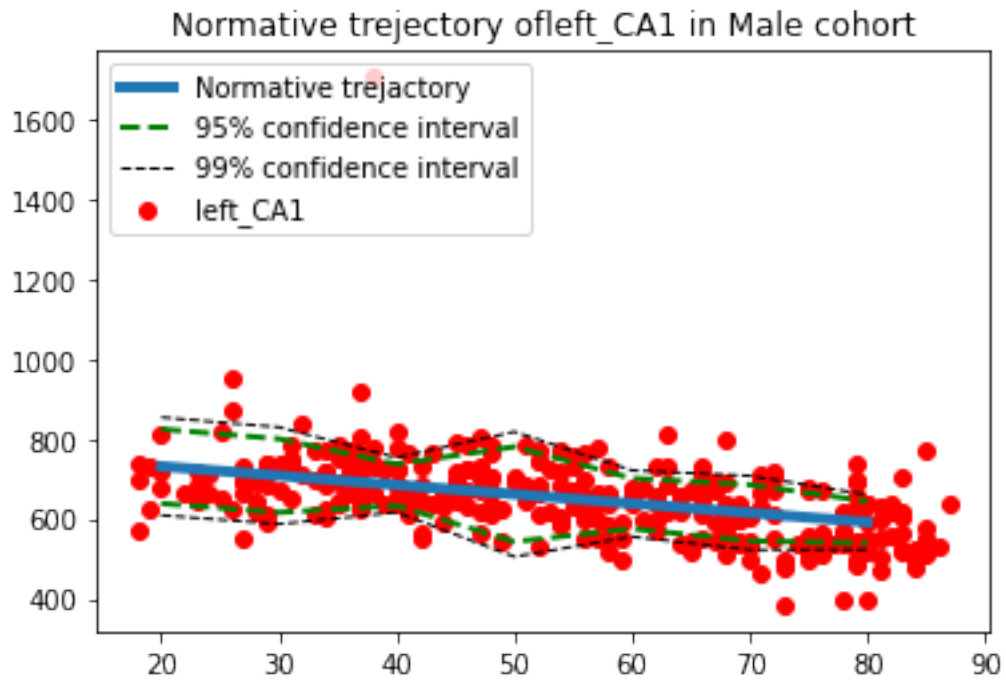
```
    y=y.values[inx]
# confidence Interval yhat+ z *(std/n^.5)-->.95 % CI:z=1.96, 99% CI:z=2.58
    s2= pd.read_csv('ys2_2fold.txt', sep = ' ', header=None)
    s2=s2.values[inx]

    CI_95=confidence_interval(s2,x,1.96)
    CI_99=confidence_interval(s2,x,2.58)

# Creat a trejactroy for each point
    for j,name in enumerate(feature_names):
         fig=plt.figure()
         ax=fig.add_subplot(111)
         ax.plot(x_forward,yhat_forward[:,j], linewidth=4, label='Normative trejactory')


         ax.plot(x_forward,CI_95[:,j]+yhat_forward[:,j], linewidth=2,linestyle='--',c='g
↪', label='95% confidence interval')
         ax.plot(x_forward,-CI_95[:,j]+yhat_forward[:,j], linewidth=2,linestyle='--',c='g
↪')

         ax.plot(x_forward,CI_99[:,j]+yhat_forward[:,j], linewidth=1,linestyle='--',c='k
↪', label='99% confidence interval')
         ax.plot(x_forward,-CI_99[:,j]+yhat_forward[:,j], linewidth=1,linestyle='--',c='k
↪')

         ax.scatter(x,y[:,j],c='r', label=name)
         plt.legend(loc='upper left')
         plt.title('Normative trejectory of' +name+' in '+sex+' cohort')
         plt.show()
         plt.close()

# code by M. Zabihi
```

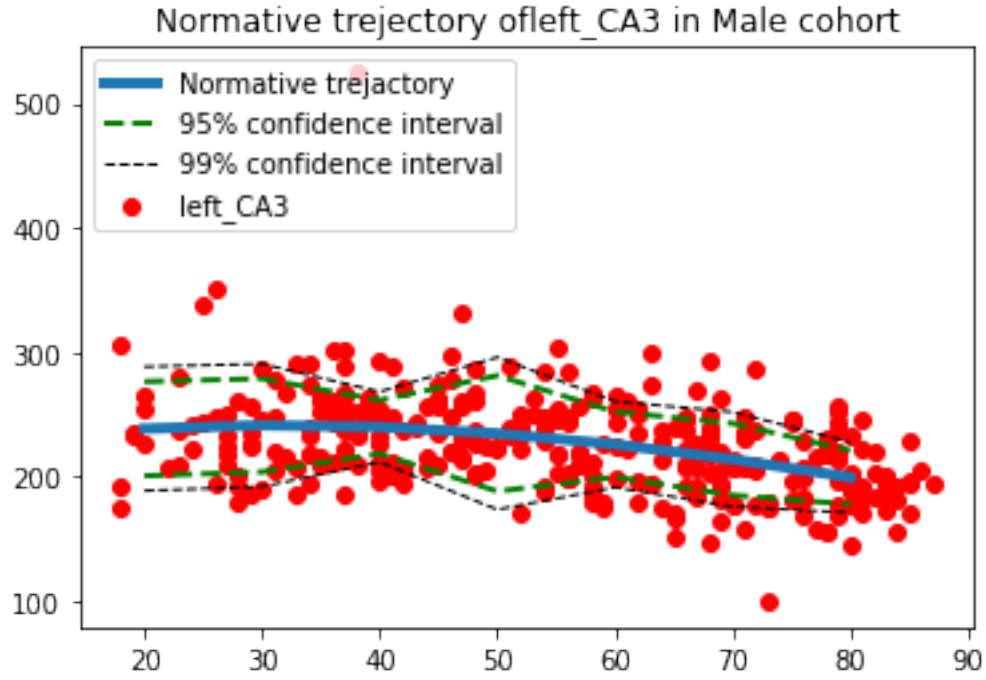

Normative trejectory ofleft_CA1 in Female cohort
Normative trejactory
95% confidence interval
99% confidence interval
left_CA1
800
700
600
500
400
20
30
40
50
60
70
80
90

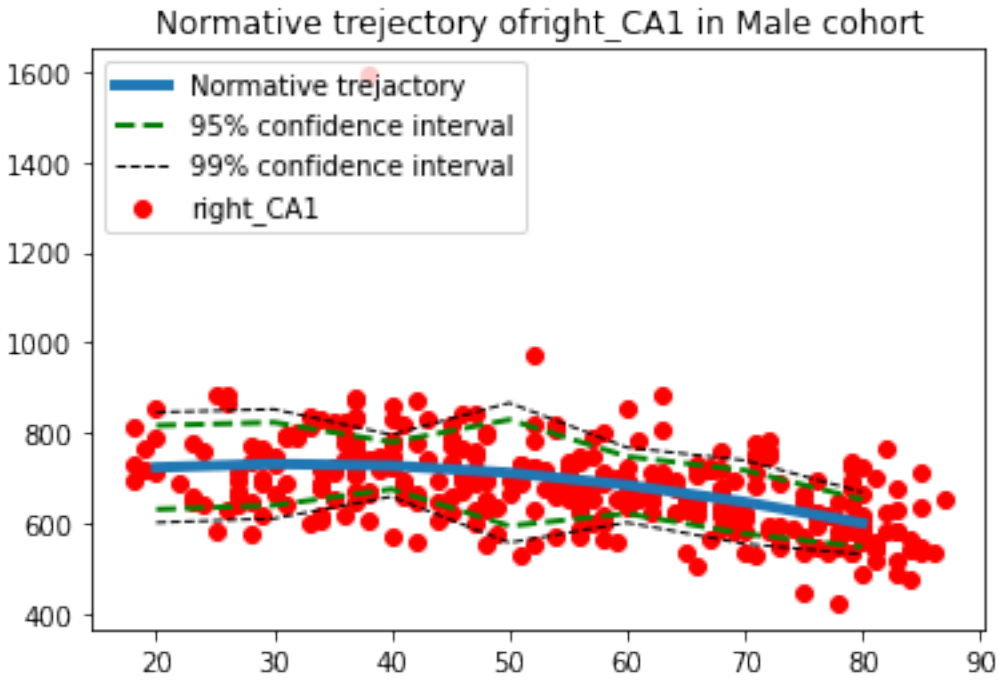

Normative trejectory ofleft_CA3 in Female cohort
Normative trejactory
95% confidence interval
99% confidence interval
left_CA3
350
300
250
200
150
20
30
40
50
60
70
80
90

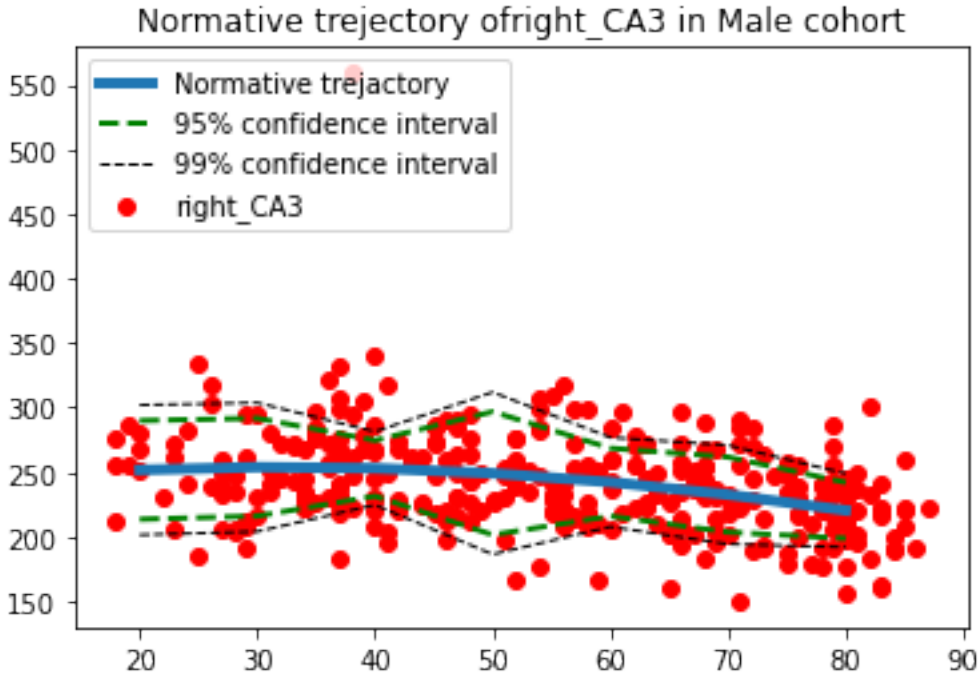

Normative trejectory ofright_CA1 in Female cohort
Normative trejectory
95% confidence interval
99% confidence interval
right_CA1
900
800
700
600
500
20
30
40
50
60
70
80
90


Normative trejectory ofright_CA3 in Female cohort
Normative trejectory
95% confidence interval
99% confidence interval
right_CA3
325
300
275
250
225
200
175
150
20
30
40
50
60
70
80
90


Normative trejectory ofleft_CA1 in Male cohort
Normative trejactory
95% confidence interval
99% confidence interval
left_CA1
1600
1400
1200
1000
800
600
400
20
30
40
50
60
70
80
90


Normative trejectory ofleft_CA3 in Male cohort
Normative trejactory
95% confidence interval
99% confidence interval
left_CA3
500
400
300
200
100
20
30
40
50
60
70
80
90


Normative trejectory ofright_CA1 in Male cohort
Normative trejactory
95% confidence interval
99% confidence interval
right_CA1
1600
1400
1200
1000
800
600
400
20
30
40
50
60
70
80
90


Normative trejectory ofright_CA3 in Male cohort
Normative trejactory
95% confidence interval
99% confidence interval
right_CA3
550
500
450
400
350
300
250
200
150
20
30
40
50
60
70
80
90

4.8 TASK 6: Apply the normative model to Nordan's data and the dementia patients.

```
# read in Nordan's as well as the patient's demographics and features
demographics_nordan = pd.read_csv('camcan_demographics_nordan.csv',
                                        sep= ",",
                                        index_col = 0)
features_nordan = pd.read_csv('camcan_features_nordan.csv',
                              sep=",",
                              index_col = 0)

# create a covariate file for Nordan's as well as the patient's demograhpics
covariate_nordan = demographics_nordan[['sex',
                                         'age']]
covariate_nordan.to_csv('covariate_nordan.txt',
                        sep = ' ',
                        header = False,
                        index = False)

# create the corresponding feature file
features_nordan = features_nordan[['left_CA1',
                                   'left_CA3',
                                   'right_CA1',
                                   'right_CA3']]

features_nordan.to_csv('features_nordan.txt',
                        sep = ' ',
                        header = False,
                        index = False)

# apply normative modeling
pcn.normative.estimate(covfile = 'covariate_normsample.txt',
                       respfile = 'features_normsample.txt',
                       testcov = 'covariate_nordan.txt',
                       testresp = 'features_nordan.txt',
                       cvfolds = None,
                       alg = 'gpr',
                       outputsuffix = '_nordan')

# code by T. Wolfers
```

```
Processing data in features_normsample.txt
Estimating model  1 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3781.497401
         Iterations: 20
         Function evaluations: 61
```

(continues on next page)

(continued from previous page)

```
         Gradient evaluations: 54
Estimating model  2 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3201.761309
         Iterations: 39
         Function evaluations: 108
         Gradient evaluations: 108
Estimating model  3 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3771.310488
         Iterations: 47
         Function evaluations: 181
         Gradient evaluations: 167
Estimating model  4 of 4
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Warning: Estimation of posterior distribution failed
Optimization terminated successfully.
         Current function value: 3200.837262
         Iterations: 40
         Function evaluations: 104
         Gradient evaluations: 104
Evaluating the model ...
Writing outputs ...
```

4.9 TASK 7: In which hyppocampal subfield(s) does Nordan deviate extremely?

No coding necessary just create a presentation which includes recommendations to Nordan and his family. Use i) |Z| > 3.6 ii) |Z| > 1.96 as definitions for extreme normative deviations.

4.10 TASK 8 (OPTIONAL): Implement a function that calculates percentage change.

Percentage change = $\frac{x1-x2}{|x2|} * 100$

```
# function that calculates percentage change
def calculate_percentage_change(x1, x2):
  percentage_change = ((x1 - x2) / abs(x2)) * 100
  return percentage_change

# code by T. Wolfers
```

4.11 TASK 9 (OPTIONAL): Visualize percent change

Plot the prercentage change in Yhat of the forward model in reference to age 20. Do that for both sexes seperately.

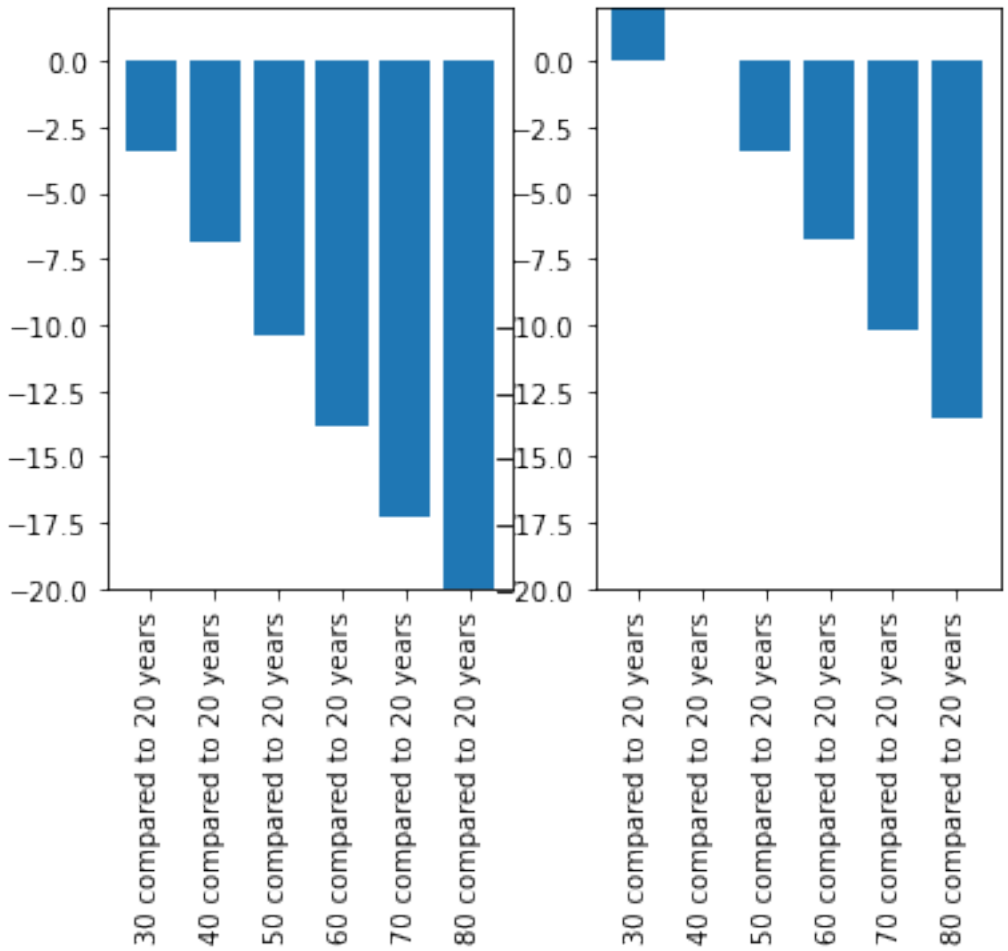
```
import matplotlib.pyplot as plt

forward_yhat = pd.read_csv('yhat_forward.txt', sep = ' ', header=None)

# You can indicate here which hypocampal subfield you like to visualize
hyppocampal_subfield = 0

percentage_change_female = []
percentage_change_male = []
count = 0
lengths = len(forward_yhat[hyppocampal_subfield])
for entry in forward_yhat[hyppocampal_subfield]:
  if count > 0 and count < 7:
    loop_percentage_change_female = calculate_percentage_change(entry,
                                                                 forward_yhat.iloc[0,
                                                                                   ␣
↪hyppocampal_subfield])
    percentage_change_female.append(loop_percentage_change_female)
  elif count > 7:
    loop_percentage_change_male = calculate_percentage_change(entry,
                                                              forward_yhat.iloc[9,
                                                                                ␣
↪hyppocampal_subfield])
    percentage_change_male.append(loop_percentage_change_male)
  count = count + 1
```

(continues on next page)

(continued from previous page)

```
names = ['30 compared to 20 years',
         '40 compared to 20 years',
         '50 compared to 20 years',
         '60 compared to 20 years',
         '70 compared to 20 years',
         '80 compared to 20 years']

# females
plt.subplot(121)
plt.bar(names, percentage_change_female)
plt.xticks(rotation=90)
plt.ylim(-20, 2)

# males
plt.subplot(122)
plt.bar(names, percentage_change_male)
plt.xticks(rotation=90)
plt.ylim(-20, 2)

# code by T. Wolfers
```

```
(-20.0, 2.0)
```


0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−20.0
30 compared to 20 years
40 compared to 20 years
50 compared to 20 years
60 compared to 20 years
70 compared to 20 years
80 compared to 20 years
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−20.0
30 compared to 20 years
40 compared to 20 years
50 compared to 20 years
60 compared to 20 years
70 compared to 20 years
80 compared to 20 years

CHAPTER

FIVE

HIERARCHICAL BAYESIAN REGRESSION

This notebook will go through basic data preparation (training and testing set, see Saige's tutorial on Normative Modelling for more detail), the actual training of the models, and will finally describe how to transfer the trained models onto unseen sites.

Created by Saige Rutherford

Adapted/edited by Andre Marquand and Pierre Berthet.

5.1 Step 0: Install necessary libraries & grab data files

```
! pip install pcntoolkit==0.26
```

For this tutorial we will use data from the Functional Connectom Project FCON1000 to create a multi-site dataset.

The dataset contains some cortical measures (eg thickness), processed by Freesurfer 6.0, and some covariates (eg age, site, gender).

First we import the required package, and create a working directory.

```
import os
import pandas as pd
import pcntoolkit as ptk
import numpy as np
import pickle
from matplotlib import pyplot as plt
```

```
processing_dir = "HBR_demo/"    # replace with a path to your working directory
if not os.path.isdir(processing_dir):
    os.makedirs(processing_dir)
os.chdir(processing_dir)
processing_dir = os.getcwd()
```

5.1.1 Overview

Here we get the FCON dataset, remove the ICBM site for later transfer, assign some site id to the different scanner sites and print an overview of the left hemisphere mean raw cortical thickness as a function of age, color coded by the various sites:

```
fcon = pd.read_csv('https://raw.githubusercontent.com/predictive-clinical-neuroscience/
↪PCNtoolkit-demo/main/data/fcon1000.csv')

icbm = fcon.loc[fcon['site'] == 'ICBM']
icbm['sitenum'] = 0
fcon = fcon.loc[fcon['site'] != 'ICBM']

sites = fcon['site'].unique()
fcon['sitenum'] = 0

f, ax = plt.subplots(figsize=(12, 12))

for i,s in enumerate(sites):
    idx = fcon['site'] == s
    fcon['sitenum'].loc[idx] = i

    print('site',s, sum(idx))
    ax.scatter(fcon['age'].loc[idx], fcon['lh_MeanThickness_thickness'].loc[idx])

ax.legend(sites)
ax.set_ylabel('LH mean cortical thickness [mm]')
ax.set_xlabel('age')
```

5.2 Step 1: Prepare training and testing sets

Then we randomly split half of the samples (participants) to be either in the training or in the testing samples. We do this for the remaing FCON dataset and for the ICBM data. The transfer function will also require a training and a test sample.

The numbers of samples per sites used for training and for testing are then displayed.

```
tr = np.random.uniform(size=fcon.shape[0]) > 0.5
te = ~tr

fcon_tr = fcon.loc[tr]
fcon_te = fcon.loc[te]

tr = np.random.uniform(size=icbm.shape[0]) > 0.5
te = ~tr

icbm_tr = icbm.loc[tr]
icbm_te = icbm.loc[te]

print('sample size check')
for i,s in enumerate(sites):
    idx = fcon_tr['site'] == s
```

(continues on next page)

(continued from previous page)

```
    idxte = fcon_te['site'] == s
    print(i,s, sum(idx), sum(idxte))

fcon_tr.to_csv(processing_dir + '/fcon1000_tr.csv')
fcon_te.to_csv(processing_dir + '/fcon1000_te.csv')
icbm_tr.to_csv(processing_dir + '/fcon1000_icbm_tr.csv')
icbm_te.to_csv(processing_dir + '/fcon1000_icbm_te.csv')
```

Otherwise you can just load these pre defined subsets:

```
# Optional
# fcon_tr = pd.read_csv('https://raw.githubusercontent.com/predictive-clinical-
↪neuroscience/PCNtoolkit-demo/main/data/fcon1000_tr.csv')
# fcon_te = pd.read_csv('https://raw.githubusercontent.com/predictive-clinical-
↪neuroscience/PCNtoolkit-demo/main/data/fcon1000_te.csv')
# icbm_tr = pd.read_csv('https://raw.githubusercontent.com/predictive-clinical-
↪neuroscience/PCNtoolkit-demo/main/data/fcon1000_icbm_tr.csv')
# icbm_te = pd.read_csv('https://raw.githubusercontent.com/predictive-clinical-
↪neuroscience/PCNtoolkit-demo/main/data/fcon1000_icbm_te.csv')
```

5.3 Step 2: Configure HBR inputs: covariates, measures and batch effects

We will here only use the mean cortical thickness for the Right and Left hemisphere: two idps.

```
idps = ['rh_MeanThickness_thickness','lh_MeanThickness_thickness']
```

As input to the model, we need covariates (used to describe predictable source of variability (fixed effects), here 'age'), measures (here cortical thickness on two idps), and batch effects (random source of variability, here 'scanner site' and 'sex').

`X` corresponds to the covariate(s)

`Y` to the measure(s)

`batch_effects` to the random effects

We need these values both for the training (`_train`) and for the testing set (`_test`).

```
X_train = (fcon_tr['age']/100).to_numpy(dtype=float)
Y_train = fcon_tr[idps].to_numpy(dtype=float)
batch_effects_train = fcon_tr[['sitenum','sex']].to_numpy(dtype=int)

with open('X_train.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(X_train), file)
with open('Y_train.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(Y_train), file)
with open('trbefile.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(batch_effects_train), file)


X_test = (fcon_te['age']/100).to_numpy(dtype=float)
```

(continues on next page)

(continued from previous page)

```
Y_test = fcon_te[idps].to_numpy(dtype=float)
batch_effects_test = fcon_te[['sitenum','sex']].to_numpy(dtype=int)

with open('X_test.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(X_test), file)
with open('Y_test.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(Y_test), file)
with open('tsbefile.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(batch_effects_test), file)

# a simple function to quickly load pickle files
def ldpkl(filename: str):
    with open(filename, 'rb') as f:
        return pickle.load(f)
```

5.4 Step 3: Files and Folders grooming

```
respfile = os.path.join(processing_dir, 'Y_train.pkl')       # measurements  (eg␣
↪cortical thickness) of the training samples (columns: the various features/ROIs, rows:␣
↪observations or subjects)
covfile = os.path.join(processing_dir, 'X_train.pkl')        # covariates (eg age) the␣
↪training samples (columns: covariates, rows: observations or subjects)

testrespfile_path = os.path.join(processing_dir, 'Y_test.pkl')       # measurements  for␣
↪the testing samples
testcovfile_path = os.path.join(processing_dir, 'X_test.pkl')        # covariate file␣
↪for the testing samples

trbefile = os.path.join(processing_dir, 'trbefile.pkl')      # training batch effects␣
↪file (eg scanner_id, gender)  (columns: the various batch effects, rows: observations␣
↪or subjects)
tsbefile = os.path.join(processing_dir, 'tsbefile.pkl')      # testing batch effects file

output_path = os.path.join(processing_dir, 'Models/')    #  output path, where the␣
↪models will be written
log_dir = os.path.join(processing_dir, 'log/')           #
if not os.path.isdir(output_path):
    os.mkdir(output_path)
if not os.path.isdir(log_dir):
    os.mkdir(log_dir)

outputsuffix = '_estimate'      # a string to name the output files, of use only to you,␣
↪so adapt it for your needs.
```

5.5 Step 4: Estimating the models

Now we have everything ready to estimate the normative models. The `estimate` function only needs the training and testing sets, each divided in three datasets: covariates, measures and batch effects. We obviously specify `alg=hbr` to use the hierarchical bayesian regression method, well suited for the multi sites datasets. The remaining arguments are basic data management: where the models, logs, and output files will be written and how they will be named.

```
ptk.normative.estimate(covfile=covfile,
                       respfile=respfile,
                       tsbefile=tsbefile,
                       trbefile=trbefile,
                       alg='hbr',
                       log_path=log_dir,
                       binary=True,
                       output_path=output_path, testcov= testcovfile_path,
                       testresp = testrespfile_path,
                       outputsuffix=outputsuffix, savemodel=True)
```

Here some analyses can be done, there are also some error metrics that could be of interest. This is covered in step 6 and in Saige's tutorial on Normative Modelling.

5.6 Step 5: Transfering the models to unseen sites

Similarly to what was done before for the FCON data, we also need to prepare the ICBM specific data, in order to run the transfer function: training and testing set of covariates, measures and batch effects:

```
X_adapt = (icbm_tr['age']/100).to_numpy(dtype=float)
Y_adapt = icbm_tr[idps].to_numpy(dtype=float)
batch_effects_adapt = icbm_tr[['sitenum','sex']].to_numpy(dtype=int)

with open('X_adaptation.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(X_adapt), file)
with open('Y_adaptation.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(Y_adapt), file)
with open('adbefile.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(batch_effects_adapt), file)

# Test data (new dataset)
X_test_txfr = (icbm_te['age']/100).to_numpy(dtype=float)
Y_test_txfr = icbm_te[idps].to_numpy(dtype=float)
batch_effects_test_txfr = icbm_te[['sitenum','sex']].to_numpy(dtype=int)

with open('X_test_txfr.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(X_test_txfr), file)
with open('Y_test_txfr.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(Y_test_txfr), file)
with open('txbefile.pkl', 'wb') as file:
    pickle.dump(pd.DataFrame(batch_effects_test_txfr), file)
```

```
respfile = os.path.join(processing_dir, 'Y_adaptation.pkl')
covfile = os.path.join(processing_dir, 'X_adaptation.pkl')
```

(continues on next page)

(continued from previous page)

```
testrespfile_path = os.path.join(processing_dir, 'Y_test_txfr.pkl')
testcovfile_path = os.path.join(processing_dir, 'X_test_txfr.pkl')
trbefile = os.path.join(processing_dir, 'adbefile.pkl')
tsbefile = os.path.join(processing_dir, 'txbefile.pkl')

log_dir = os.path.join(processing_dir, 'log_transfer/')
output_path = os.path.join(processing_dir, 'Transfer/')
model_path = os.path.join(processing_dir, 'Models/')  # path to the previously trained␣
↪models
outputsuffix = '_transfer'  # suffix added to the output files from the transfer function
```

Here, the difference is that the transfer function needs a model path, which points to the models we just trained, and new site data (training and testing). That is basically the only difference.

```
yhat, s2, z_scores = ptk.normative.transfer(covfile=covfile,
                                            respfile=respfile,
                                            tsbefile=tsbefile,
                                            trbefile=trbefile,
                                            model_path = model_path,
                                            alg='hbr',
                                            log_path=log_dir,
                                            binary=True,
                                            output_path=output_path,
                                            testcov= testcovfile_path,
                                            testresp = testrespfile_path,
                                            outputsuffix=outputsuffix,
                                            savemodel=True)
```

And that is it, you now have models that benefited from prior knowledge about different scanner sites to learn on unseen sites.

5.7 Step 6: Interpreting model performance

Output evaluation metrics definitions

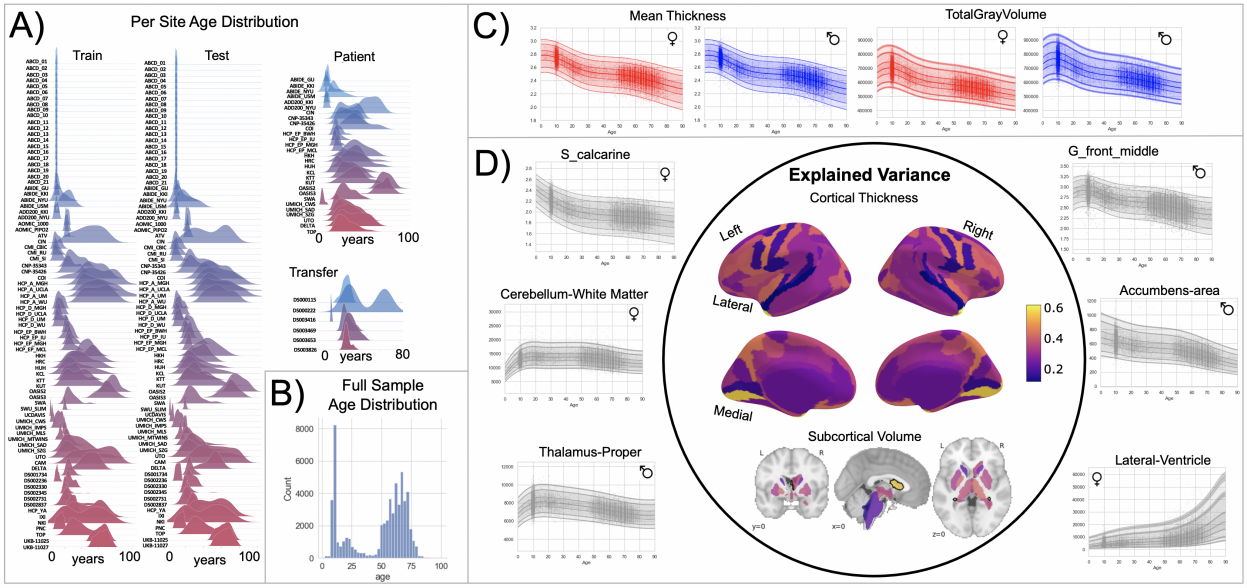
Abbreviation	Full name
NM	Normative Model
EV / EXPV	Explained Variance
MSLL	Mean Standardized Log Loss
SMSE	Standardized Mean Squared Error
RMSE	Root Mean Squared Error between true/predicted responses
Rho	Pearson orrelation between true/predicted responses
pRho	Parametric p-value for this correlation
Z	Z-score or deviation score
yhat	predictive mean
ys2	predictive variance

CHAPTER

SIX

BRAINCHARTS: TRANSFER

Code for transfering the models from Charting Brain Growth and Aging at High Spatial Precision.



This notebook shows how to apply the coefficients from pre-estimated normative models to new data. This can be done in two different ways: (i) using a new set of data derived from the same sites used to estimate the model and (ii) 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.

First, if necessary, we install PCNtoolkit (note: this tutorial requires at least version 0.20)

```
!pip install pcntoolkit==0.26
```

```
! git clone https://github.com/predictive-clinical-neuroscience/braincharts.git
```

```
Cloning into 'braincharts'...
remote: Enumerating objects: 1444, done.[K
remote: Counting objects: 100% (1444/1444), done.[K
remote: Compressing objects: 100% (1365/1365), done.[K
remote: Total 1444 (delta 153), reused 1342 (delta 75), pack-reused 0[K
```

(continues on next page)

(continued from previous page)

```
Receiving objects: 100% (1444/1444), 57.99 MiB | 34.87 MiB/s, done.
Resolving deltas: 100% (153/153), done.
```

```
# we need to be in the scripts folder when we import the libraries in the code block␣
↪below,
# because there is a function 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
```

We need to unzip the models.

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

```
ls
```

```
lifespan_12K_57sites_mqc2_train.zip  lifespan_29K_82sites_train.zip
lifespan_12K_59sites_mqc_train.zip   lifespan_57K_82sites.zip
lifespan_23K_57sites_mqc2.zip        README.md
```

```
# we will use the biggest sample as our training set (approx. N=57000 subjects from 82␣
↪sites)
# for more info on the other pretrained models available in this repository,
# please refer to the accompanying paper https://elifesciences.org/articles/72904
! unzip lifespan_57K_82sites.zip
```

Next, we configure some basic variables, like where we want the analysis to be done and which model we 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 the accompanying paper for more details

---

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

(continues on next page)

(continued from previous page)

```
# where the analysis takes place
root_dir = '/content/braincharts'
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()
```

6.1 Download test dataset

As mentioned above, to demonstrate this tool we will use a test dataset derived from the FCON 1000 dataset. We provide a prepackaged training/test split of these data in the required format (also after removing sites with only a few data points), here. you can get these data by running the following commmands:

```
os.chdir(root_dir)
!wget -nc https://raw.githubusercontent.com/predictive-clinical-neuroscience/braincharts/
↪master/docs/OpenNeuroTransfer_ct_te.csv
!wget -nc https://raw.githubusercontent.com/predictive-clinical-neuroscience/braincharts/
↪master/docs/OpenNeuroTransfer_ct_tr.csv
```

```
--2022-02-17 15:01:31--  https://raw.githubusercontent.com/predictive-clinical-
↪neuroscience/braincharts/master/docs/OpenNeuroTransfer_ct_te.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.
↪199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443.
↪.. connected.
HTTP request sent, awaiting response... 200 OK
Length: 628752 (614K) [text/plain]
Saving to: ‘OpenNeuroTransfer_te.csv’

OpenNeuroTransfer_t 100%[===================>] 614.02K  --.-KB/s    in 0.03s

2022-02-17 15:01:31 (22.0 MB/s) - ‘OpenNeuroTransfer_te.csv’ saved [628752/628752]

--2022-02-17 15:01:31--  https://raw.githubusercontent.com/predictive-clinical-
↪neuroscience/braincharts/master/docs/OpenNeuroTransfer_ct_tr.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.
↪199.110.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443.
↪.. connected.
HTTP request sent, awaiting response... 200 OK
Length: 163753 (160K) [text/plain]
Saving to: ‘OpenNeuroTransfer_tr.csv’

OpenNeuroTransfer_c 100%[===================>] 159.92K  --.-KB/s    in 0.03s

2022-02-17 15:01:32 (6.08 MB/s) - ‘OpenNeuroTransfer_ct_tr.csv’ saved [163753/163753]
```

6.2 Load test data

Now we load the test data and remove some subjects that may have poor scan quality. This asssesment is based on the Freesurfer Euler characteristic as described in the papers below.

Note: For the purposes of this tutorial, we make predictions for all sites in the FCON 1000 dataset, but two of them were also included in the training data (named 'Baltimore' and 'NewYork_a'). In this case, this will only slightly bias the accuracy, but in order to replicate the results in the paper, it would be necessary to additionally remove these sites from the test dataframe.

References - Kia et al 2021 - Rosen et al 2018

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

df_te = pd.read_csv(test_data)

# remove some bad subjects, this requires having a column called "avg_en" that␣
↪corresponds to the average Euler number extracted from Freesurfer
# df_te, bad_sub = remove_bad_subjects(df_te, df_te)

# extract a list of unique site ids from the test set
site_ids_te =  sorted(set(df_te['site'].to_list()))
```

6.3 (Optional) Load adaptation data

If the data you wish to make predictions for is not derived from the same scanning sites as those in the trainig set, it is necessary to learn the site effect so that we can account for it in the predictions. In order to do this in an unbiased way, we use a separate dataset, which we refer to as 'adaptation' data. This must contain data for all the same sites as in the test dataset and we assume these are coded in the same way, based on a the 'sitenum' column in the dataframe.

```
adaptation_data = os.path.join(root_dir, 'OpenNeuroTransfer_ct_tr.csv')

df_ad = pd.read_csv(adaptation_data)

# remove some bad subjects, this requires having a column called "avg_en" that␣
↪corresponds to the average Euler number extracted from Freesurfer
# df_ad, bad_sub = remove_bad_subjects(df_ad, df_ad)

# 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')
```

6.4 Configure which models to fit

Now, we configure which imaging derived phenotypes (IDPs) we would like to process. This is just a list of column names in the dataframe we have loaded above.

We could load the whole set (i.e. all phenotypes for which we have models for ...

```
# load the list of idps for left and right hemispheres, plus subcortical regions
with open(os.path.join(root_dir,'docs','phenotypes_ct_lh.txt')) as f:
    idp_ids_lh = f.read().splitlines()
with open(os.path.join(root_dir,'docs','phenotypes_ct_rh.txt')) as f:
    idp_ids_rh = f.read().splitlines()
with open(os.path.join(root_dir,'docs','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
↪']
```

6.5 Configure covariates

Now, we configure some parameters to fit the model. First, we 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 we loop 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). Since we are sticking with the default value, we do not need to specify any parameters for this, but we do need to specify the limits. We choose to pad the input by a few years either side of the input range. We will also set a couple of options that control the estimation of the model

For further details about the likelihood warping approach, see the accompanying paper and Fraza et al 2021.

```
# 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 treshold above which a sample is considered to be an outlier (without␣
↪fitting any model)
outlier_thresh = 7
```

6.6 Make predictions

This 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. Then, we 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:

- A global intercept (column of ones)
- 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 parameterisation)

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, we have a set of predictions and Z-statistics for the test dataset that we can take forward to further analysis.

When we 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')
```

(continues on next page)

(continued from previous page)

```
        # 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)
```

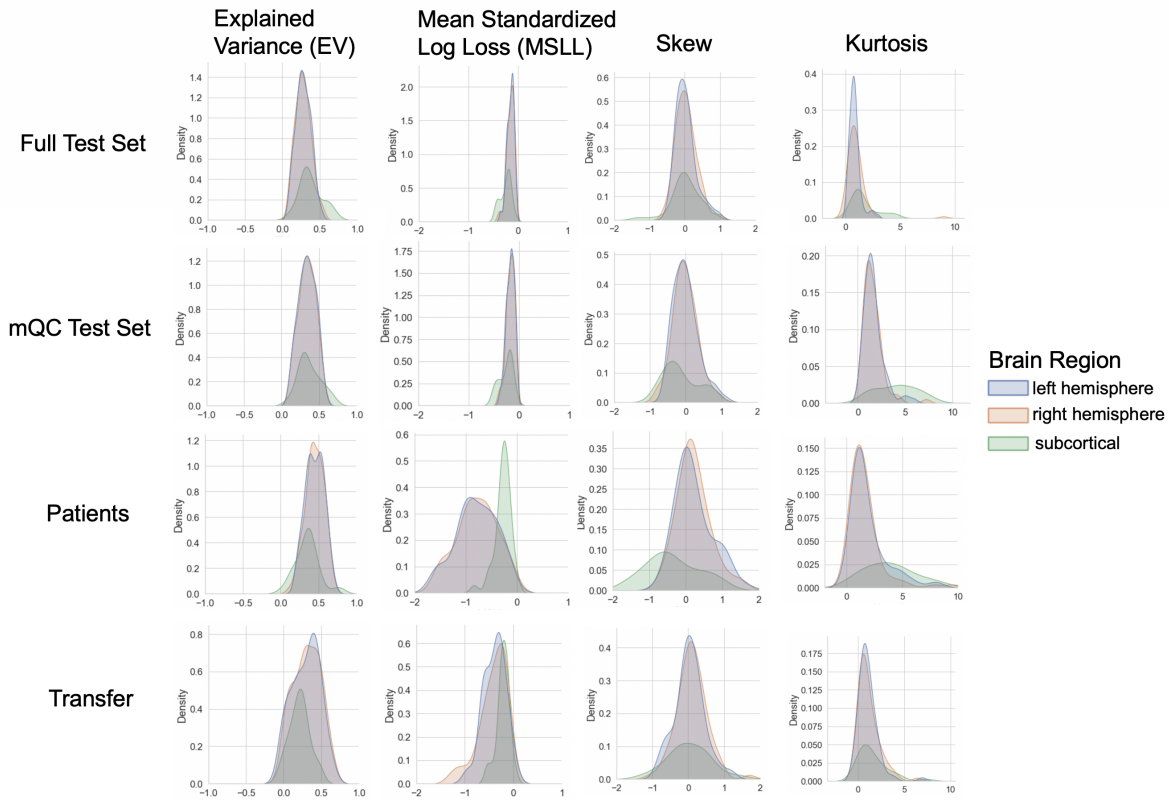
```
Running IDP 0 Left-Thalamus-Proper :
Some sites missing from the training data. Adapting model
Loading data ...
Prediction by model  1 of 1
Evaluating the model ...
Evaluations Writing outputs ...
Writing outputs ...
Running IDP 1 Left-Lateral-Ventricle :
Some sites missing from the training data. Adapting model
Loading data ...
Prediction by model  1 of 1
Evaluating the model ...
Evaluations Writing outputs ...
Writing outputs ...
Running IDP 2 rh_MeanThickness_thickness :
Some sites missing from the training data. Adapting model
Loading data ...
```

(continues on next page)

(continued from previous page)

```
Prediction by model  1 of 1
Evaluating the model ...
Evaluations Writing outputs ...
Writing outputs ...
```

6.7 Evaluate the performance



6.8 Preparing dummy data for plotting

Now, we plot the centiles of variation estimated by the normative model.

We 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'
```

(continues on next page)

(continued from previous page)

```
# create dummy data for visualisation
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)
```

```
configuring dummy data ...
```

6.9 Plotting the normative models

Now 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 visualisation, 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 also that 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
```

(continues on next page)

(continued from previous page)

```
    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]
```

(continues on next page)

(continued from previous page)

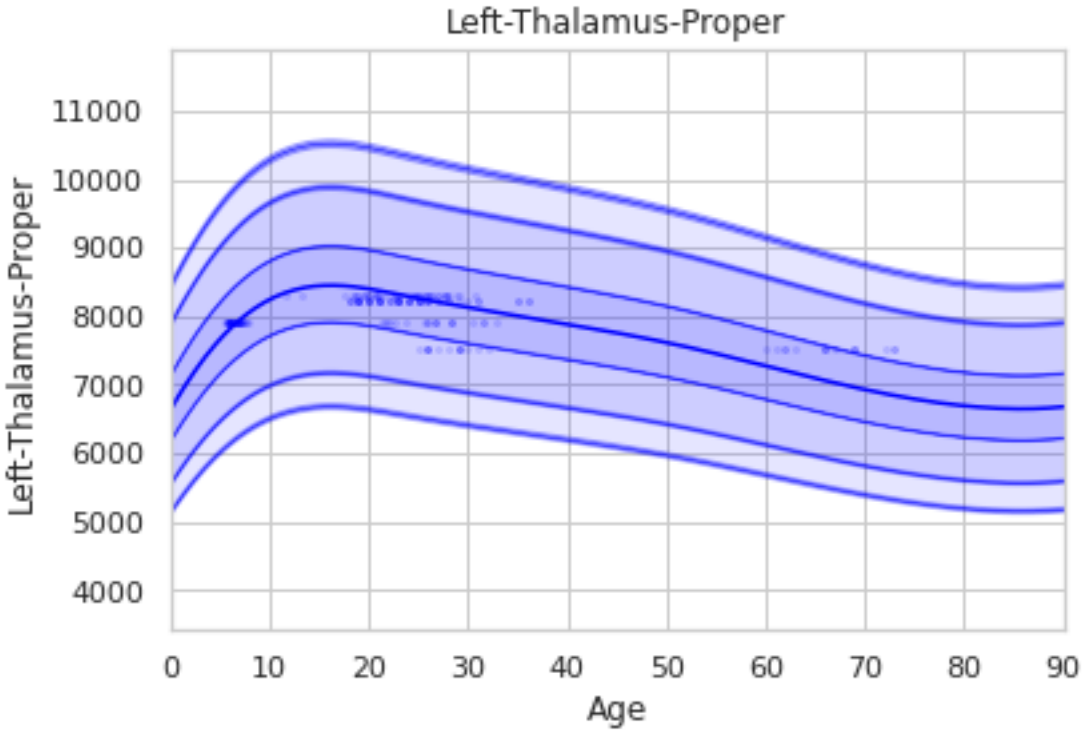
```
            # load the adaptation data
            y_ad = load_2d(os.path.join(idp_dir, 'resp_ad.txt'))
            X_ad = load_2d(os.path.join(idp_dir, 'cov_bspline_ad.txt'))
            idx_a = np.where(np.bitwise_and(X_ad[:,2] == sex, (df_ad['site'] == site).to_
↪numpy()))[0]
            if len(idx) < 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)
```

(continues on next page)

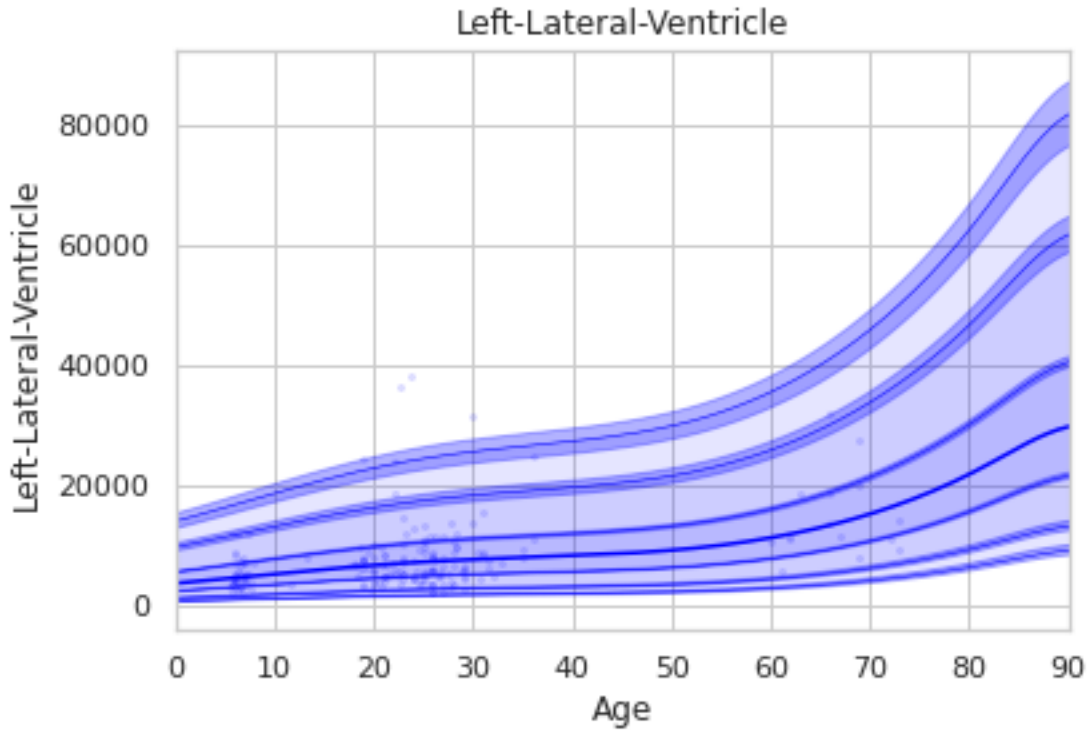
(continued from previous page)

```
    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)
```

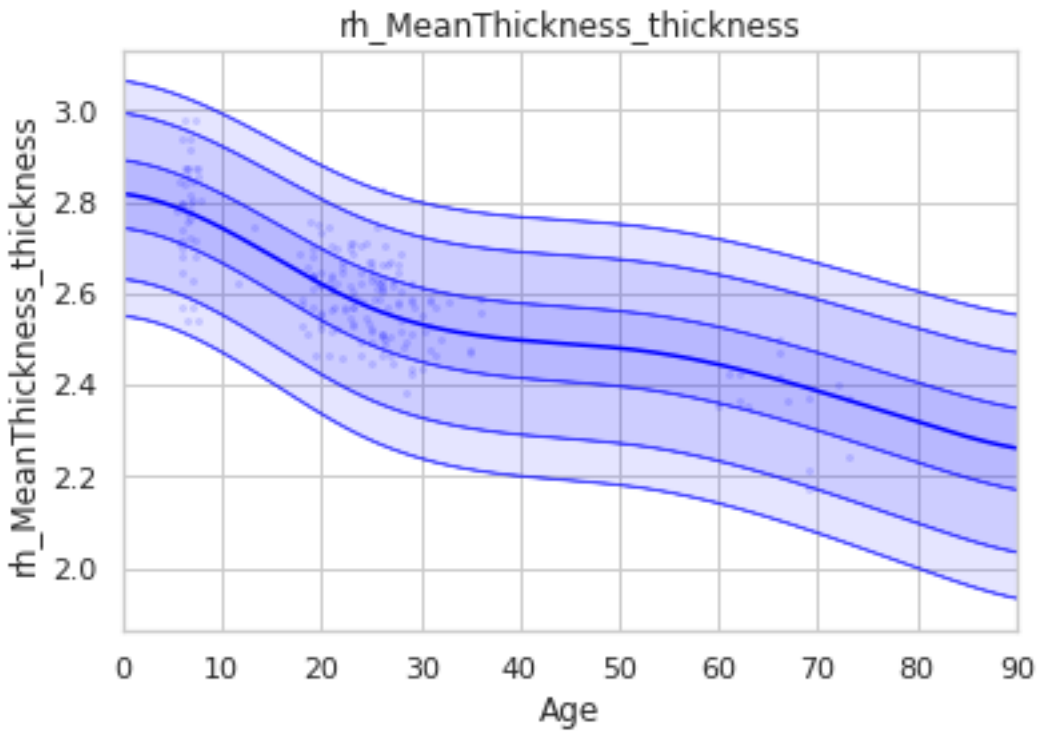
```
Running IDP 0 Left-Thalamus-Proper :
Making predictions with dummy covariates (for visualisation)
Loading data ...
Prediction by model  1 of 1
Writing outputs ...
metrics: {'RMSE': array([0.55690777]), 'Rho': array([0.]), 'pRho': array([1.]), 'SMSE':␣
↪array([0.]), 'EXPV': array([0.])}
```



```
Running IDP 1 Left-Lateral-Ventricle :
Making predictions with dummy covariates (for visualisation)
Loading data ...
Prediction by model  1 of 1
Writing outputs ...
metrics: {'RMSE': array([4205.49266088]), 'Rho': array([0.45898577]), 'pRho': array([5.
↪62632393e-25]), 'SMSE': array([0.81397727]), 'EXPV': array([0.19814613])}
```




```
Running IDP 2 rh_MeanThickness_thickness :
Making predictions with dummy covariates (for visualisation)
Loading data ...
Prediction by model  1 of 1
Writing outputs ...
metrics: {'RMSE': array([0.08652435]), 'Rho': array([0.77666469]), 'pRho': array([2.
↪97430261e-103]), 'SMSE': array([0.40227749]), 'EXPV': array([0.59789079])}
```




```
# explore an example output folder of a single model (one ROI)
# think about what each of these output files represents.
# Hint: look at the variable names and comments in the code block above
! ls rh_MeanThickness_thickness/
```

```
centiles_1.png          MSLL_predict.txt  RMSE_predict.txt  yhat_predict.txt
cov_bspline_ad.txt  pRho_predict.txt  sitenum_ad.txt        ys2_dummy.pkl
cov_bspline_te.txt  resp_ad.txt       sitenum_te.txt        ys2_predict.txt
EXPV_predict.txt    resp_te.txt       SMSE_predict.txt      Z_predict.txt
Models                  Rho_predict.txt   yhat_dummy.pkl
```

```
# 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), so we can
# verify by counting the line numbers in the Z_predict.txt file
! cat rh_MeanThickness_thickness/Z_predict.txt | wc
```

```
436     436   11115
```

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

```
! mkdir deviation_scores
```

```
! 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/lifespan_57K_82sites/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)
```

CHAPTER

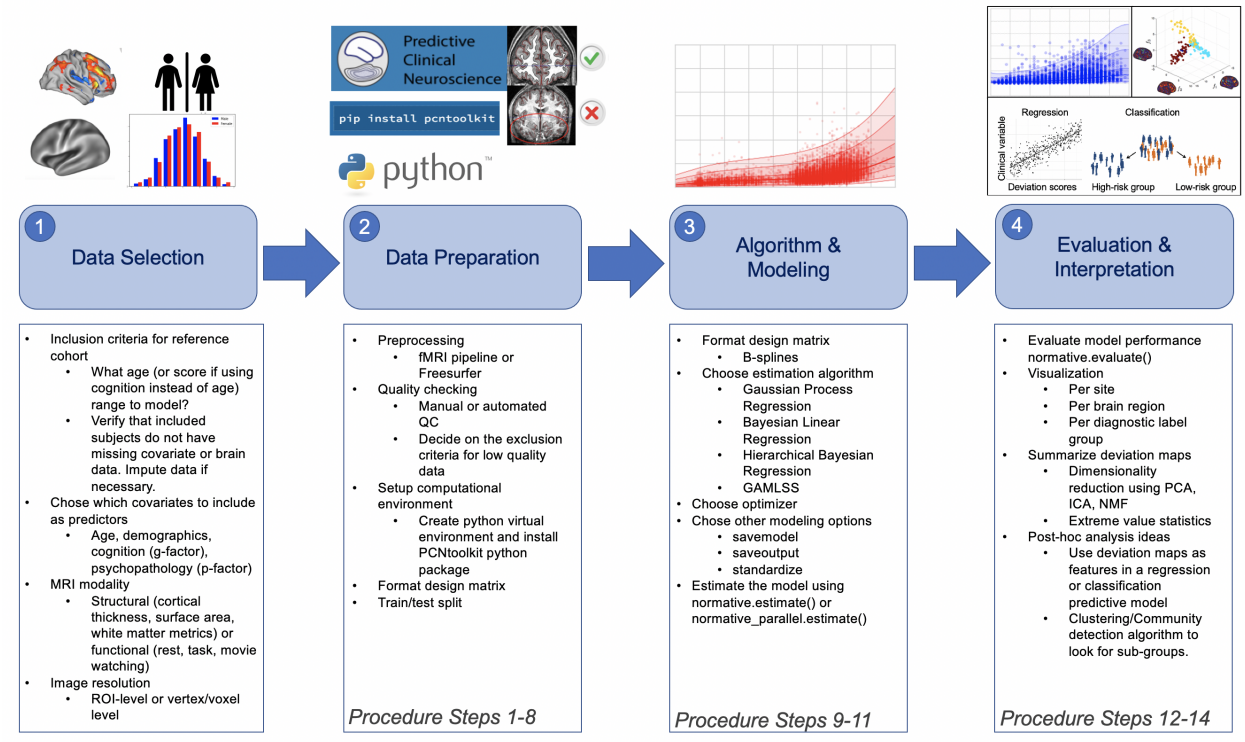
SEVEN

BAYESIAN LINEAR REGRESSION

The Normative Modeling Framework for Computational Psychiatry. Nature Protocols. https://www.nature.com/articles/s41596-022-00696-5.

Created by Saige Rutherford

Using Multi-Site Cortical Thickness Data



7.1 Data Preparation

7.1.1 Install necessary libraries & grab data files

Begin by cloning the GitHub repository using the following commands. This repository contains the necessary code and example data. Then install the python packages using pip and import them into the python environment (either Google Colab or using a local python installation on your computer).

```
! git clone https://github.com/predictive-clinical-neuroscience/PCNtoolkit-demo.git
```

```
Cloning into 'PCNtoolkit-demo'...
remote: Enumerating objects: 855, done.[K
remote: Counting objects: 100% (855/855), done.[K
remote: Compressing objects: 100% (737/737), done.[K
remote: Total 855 (delta 278), reused 601 (delta 101), pack-reused 0[K
Receiving objects: 100% (855/855), 18.07 MiB | 13.53 MiB/s, done.
Resolving deltas: 100% (278/278), done.
```

```
import os

# set this path to the git cloned PCNtoolkit-demo repository --> Uncomment whichever␣
↪line you need for either running on your own computer or on Google Colab.
#os.chdir('/Users/PCNtoolkit-demo/tutorials/BLR_protocol') # if running on your own␣
↪computer, use this line (change the path to match where you cloned the repository)
os.chdir('/content/PCNtoolkit-demo/tutorials/BLR_protocol') # if running on Google Colab,
↪ use this line
```

```
! pip install -r requirements.txt
```

7.1.2 Prepare covariate data

The data set (downloaded in Step 1) includes a multi-site dataset from the Human Connectome Project Young Adult study and IXI. It is also possible to use different datasets (i.e., your own data or additional public datasets) in this step. If using your own data here, it is recommended to load the example data to view the column names in order to match your data to this format. Read in the data files using pandas, then merge the covariate (age & sex) data from each site into a single data frame (named cov). The columns of this covariate data frame represent the predictor variables. Additional columns may be added here, depending on the research question.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joypy
from sklearn.model_selection import train_test_split
from pcntoolkit.normative import estimate, evaluate
from pcntoolkit.util.utils import create_bspline_basis, compute_MSLL
```

```
# if running in Google colab, remove the "data/" folder from the path
hcp = pd.read_csv('/content/PCNtoolkit-demo/data/HCP1200_age_gender.csv')
ixi = pd.read_csv('/content/PCNtoolkit-demo/data/IXI_age_gender.csv')
```

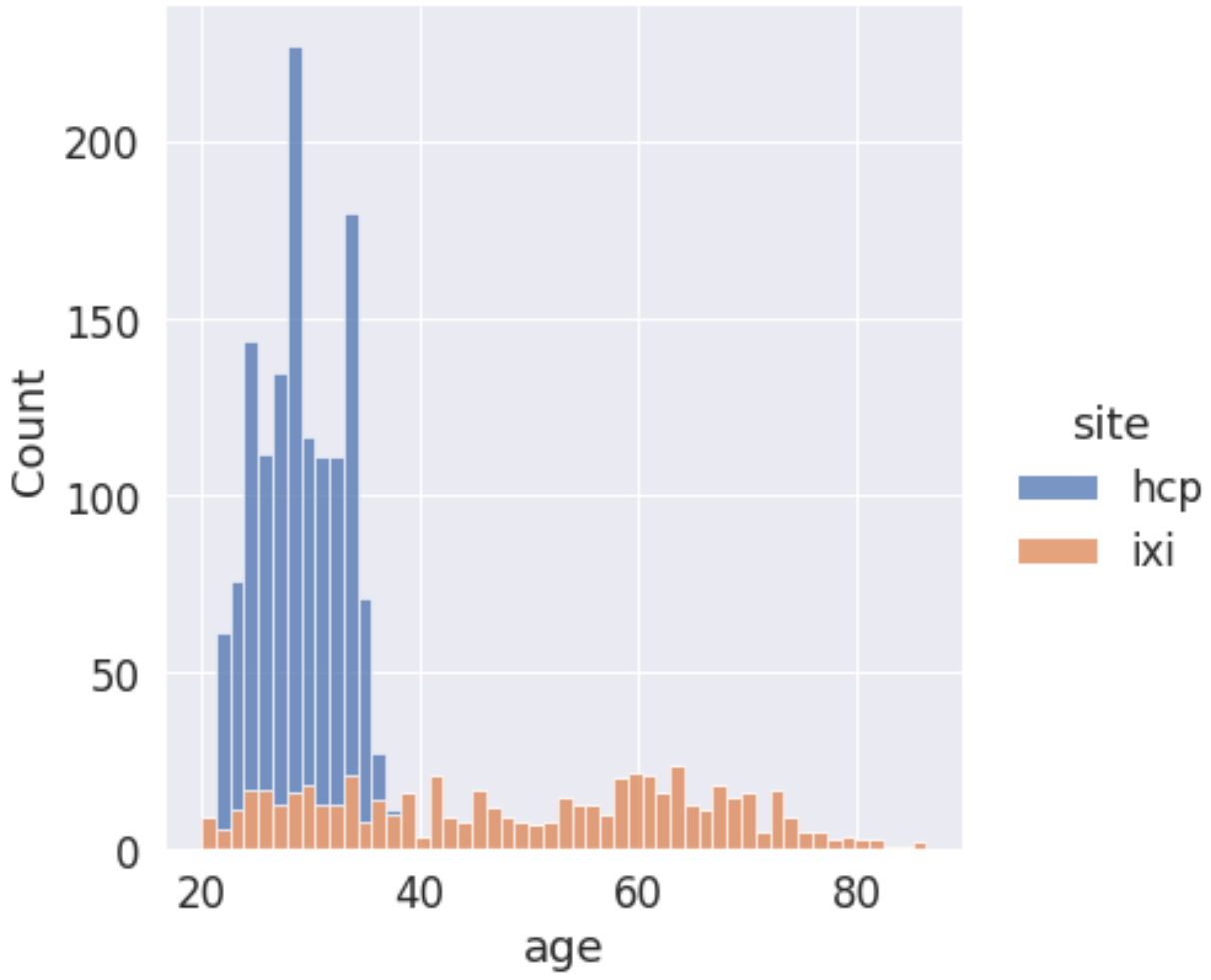
```
cov = pd.merge(hcp, ixi, on=["participant_id", "age", "sex", "site"], how='outer')
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/reshape/merge.py:1218: UserWarning:␣
↪You are merging on int and float columns where the float values are not equal to their␣
↪int representation
  UserWarning,
```

```
sns.set(font_scale=1.5, style='darkgrid')
```

```
sns.displot(cov, x="age", hue="site", multiple="stack", height=6)
```

```
<seaborn.axisgrid.FacetGrid at 0x7ff321c7af90>
```




```
cov.groupby(['site']).describe()
```

7.1.3 Prepare brain data

Next, format and combine the MRI data using the following commands. The example data contains cortical thickness maps estimated by running recon-all from Freesurfer (version 6.0). The dimensionality of the data was reduced by using ROIs from the Desikan-Killiany atlas. Including the Euler number as a covariate is also recommended, as this is a proxy metric for data quality. The Euler number from each subjects recon-all output folder was extracted into a text file and is merged into the cortical thickness data frame. The Euler number is site-specific, thus, to use the same exclusion threshold across sites it is important to center the site by subtracting the site median from all subjects at a site. Then take the square root and multiply by negative one and exclude any subjects with a square root above 10.

Here is some psuedo-code (run from a terminal in the folder that has all subjects recon-all output folders) that was used to extract these ROIs:

```
export SUBJECTS_DIR=/path/to/study/freesurfer_data/

aparcstats2table --subject sub-* --hemi lh --meas thickness --tablefile
HCP1200_aparc_lh_thickness.txt

aparcstats2table --subject sub-* --hemi rh --meas thickness --tablefile
HCP1200_aparc_rh_thickness.txt
```

```
hcpya = pd.read_csv('/content/PCNtoolkit-demo/data/HCP1200_aparc_thickness.csv')
ixi = pd.read_csv('/content/PCNtoolkit-demo/data/IXI_aparc_thickness.csv')
```

```
brain_all = pd.merge(ixi, hcpya, how='outer')
```

We extracted the euler number from each subjects recon-all output folder into a text file and we now need to format and combine these into our brain dataframe.

Below is psuedo code for how we extracted the euler number from the recon-all.log for each subject. Run this from the terminal in the folder where your subjects recon-all output folders are located. This assumes that all of your subject IDs start with "sub-" prefix.

```
for i in sub-*; do if [[ -e ${i}/scripts/recon-all.log ]]; then cat ${i}/scripts/
recon-all.log | grep -A 1 "Computing euler" > temp_log; lh_en=$(cat temp_log | head -2
| tail -1 | awk -F '=' '{print $2}' | awk -F ',' '{print $1}'); rh_en=$(cat temp_log |
head -2 | tail -1 | awk -F '=' '{print $3}'); echo "${i}, ${lh_en}, ${rh_en}" >> euler.
csv; echo ${i}; fi; done
```

```
hcp_euler = pd.read_csv('/content/PCNtoolkit-demo/data/hcp-ya_euler.csv')
ixi_euler = pd.read_csv('/content/PCNtoolkit-demo/data/ixi_euler.csv')
```

```
hcp_euler['site'] = 'hcp'
ixi_euler['site'] = 'ixi'
```

```
hcp_euler.replace(r'^\s*$', np.nan, regex=True, inplace=True)
ixi_euler.replace(r'^\s*$', np.nan, regex=True, inplace=True)
```

```
hcp_euler.dropna(inplace=True)
ixi_euler.dropna(inplace=True)
```

```
hcp_euler['rh_euler'] = hcp_euler['rh_euler'].astype(int)
hcp_euler['lh_euler'] = hcp_euler['lh_euler'].astype(int)
ixi_euler['rh_euler'] = ixi_euler['rh_euler'].astype(int)
ixi_euler['lh_euler'] = ixi_euler['lh_euler'].astype(int)
```

```
df_euler = pd.merge(hcp_euler, ixi_euler, on=['participant_id', 'lh_euler', 'rh_euler',
↪'site'], how='outer')
```

Finally, we need to center the euler number for each site. The euler number is very site-specific so in order to use the same exclusion threshold across sites we need to center the site by subtracting the site median from all subjects at a site. Then we will take the square root and multiply by negative one and exclude any subjects with a square root above 10. This choice of threshold is fairly random. If possible all of your data should be visually inspected to verify that the data inclusion is not too strict or too lenient.

```
df_euler['avg_euler'] = df_euler[['lh_euler','rh_euler']].mean(axis=1)
```

```
df_euler.groupby(by='site').median()
```

```
df_euler['site_median'] = df_euler['site']
```

```
df_euler['site_median'] = df_euler['site_median'].replace({'hcp':-43,'ixi':-56})
```

```
df_euler['avg_euler_centered'] = df_euler['avg_euler'] - df_euler['site_median']
```

```
df_euler['avg_euler_centered_neg'] = df_euler['avg_euler_centered']*-1
```

```
df_euler['avg_euler_centered_neg_sqrt'] = np.sqrt(np.absolute(df_euler['avg_euler_
↪centered_neg']))
```

```
brain = pd.merge(df_euler, brain_all, on=['participant_id'], how='inner')
```

```
brain_good = brain.query('avg_euler_centered_neg_sqrt < 10')
```

Warning: **CRITICAL STEP:** If possible, data should be visually inspected to verify that the data inclusion is not too strict or too lenient. Subjects above the Euler number threshold should be manually checked to verify and justify their exclusion due to poor data quality. This is just one approach for automated QC used by the developers of the PCNtoolkit. Other approaches such as the ENIGMA QC pipeline or UK Biobanks QC pipeline are also viable options for automated QC.

7.1.4 Combine covariate & cortical thickness dataframes

The normative modeling function requires the covariate predictors and brain features to be in separate text files. However, it is important to first (inner) merge them together, using the following commands, to confirm that the same subjects are in each file and that the rows (representing subjects) align. This requires that both data frames have 'subject_id' as a column name. Once this is confirmed, exclude rows with NaN values and separate the brain features and covariate predictors into their own dataframes, using the commands below.

```
# make sure to use how="inner" so that we only include subjects that have data in both␣
↪the covariate and the cortical thickness files
all_data = pd.merge(brain_good, cov, how='inner')
```

```
# Create a list of all the ROIs you want to run a normative model for (add additional␣
↪names to this list if you would like to include other brain regions from the Desikan-
```

(continues on next page)

(continued from previous page)

```
↪Killian atlas)
roi_ids = ['lh_MeanThickness_thickness',
           'rh_MeanThickness_thickness',
           'lh_bankssts_thickness',
           'lh_caudalanteriorcingulate_thickness',
           'lh_superiorfrontal_thickness',
           'rh_superiorfrontal_thickness']
```

```
# Remove any subjects that have NaN variables in any of the columns
all_data.dropna(subset=roi_ids, inplace=True)
```

```
all_data_features = all_data[roi_ids]
```

```
all_data_covariates = all_data[['age','sex','site']]
```

Warning: **CRITICAL STEP:** `roi_ids` is a variable that represents which brain areas will be modeled and can be used to select subsets of the data frame if you do not wish to run models for the whole brain.

7.1.5 Add variable to model site/scanner effects

Currently, the different sites are coded in a single column (named 'site') and are represented as a string data type. However, the PCNtoolkit requires binary variables. Use the pandas package as follows to address this, which has a built-in function, `pd.get_dummies`, that takes in the string 'site' column and dummy encodes the site variable so that there is now a column for each site and the columns contain binary variables (0=not in this site, 1=present in this site).

```
all_data_covariates = pd.get_dummies(all_data_covariates, columns=['site'])
```

```
all_data['Average_Thickness'] = all_data[['lh_MeanThickness_thickness','rh_MeanThickness_
↪thickness']].mean(axis=1)
```

7.1.6 Train/test split

In this example, we use 80% of the data for training and 20% for testing. Please carefully read the experimental design section on train/test split considerations when using your own data in this step. Using a function from scikit-learn (`train_test_split`), stratify the train/test split using the site variable to make sure that the train/test sets both contain data from all sites, using the following commands. Next, confirm that your train and test arrays are the same size (rows), using the following commands. You do not need the same size columns (subjects) in the train and test arrays, but the rows represent the covariate and responses which should be the same across train and test arrays.

```
X_train, X_test, y_train, y_test = train_test_split(all_data_covariates, all_data_
↪features, stratify=all_data['site'], test_size=0.2, random_state=42)
```

Verify that your train & test arrays are the same size

```
tr_cov_size = X_train.shape
tr_resp_size = y_train.shape
te_cov_size = X_test.shape
```

(continues on next page)

(continued from previous page)

```
te_resp_size = y_test.shape
print("Train covariate size is: ", tr_cov_size)
print("Test covariate size is: ", te_cov_size)
print("Train response size is: ", tr_resp_size)
print("Test response size is: ", te_resp_size)
```

```
Train covariate size is:  (1353, 4)
Test covariate size is:  (339, 4)
Train response size is:  (1353, 6)
Test response size is:  (339, 6)
```

Warning: CRITICAL STEP: The model would not learn the site effects if all the data from one site was only in the test set. Therefore, we stratify the train/test split using the site variable.

When the data were split into train and test sets, the row index was not reset. This means that the row index in the train and test data frames still correspond to the full data frame (before splitting the data occurred). The test set row index informs which subjects belong to which site, and this information is needed to evaluate per site performance metrics. Resetting the row index of the train/test data frames fixes this issue. Then extract the site row indices to a list (one list per site) and create a list called `site_names` that is used to decide which sites to evaluate model performance for, as follows:

```
X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

```
# Get indices of all the subejcts in each site so that we can evaluate the test set␣
↪metrics per site
hcp_idx = X_test.index[X_test['site_hcp'] == 1].to_list()
ixi_idx = X_test.index[X_test['site_ixi'] == 1].to_list()
```

```
# Save the site indices into a single list
sites = [hcp_idx, ixi_idx]
```

```
# Create a list with sites names to use in evaluating per-site metrics
site_names = ['hcp', 'ixi']
```

7.1.7 Setup output directories

Save each brain region to its own text file (organized in separate directories) using the following commands, because for each response variable, Y (e.g., brain region) we fit a separate normative model.

```
for c in y_train.columns:
    y_train[c].to_csv('resp_tr_' + c + '.txt', header=False, index=False)
```

```
X_train.to_csv('cov_tr.txt', sep = '\t', header=False, index = False)
```

```
y_train.to_csv('resp_tr.txt', sep = '\t', header=False, index = False)
```

```
for c in y_test.columns:
    y_test[c].to_csv('resp_te_' + c + '.txt', header=False, index=False)
```

```
X_test.to_csv('cov_te.txt', sep = '\t', header=False, index = False)
```

```
y_test.to_csv('resp_te.txt', sep = '\t', header=False, index = False)
```

```
! if [[ ! -e ROI_models/ ]]; then mkdir ROI_models; fi
```

```
! for i in `cat /content/PCNtoolkit-demo/data/roi_dir_names`; do if [[ -e resp_tr_${i}.
↪txt ]]; then cd ROI_models; mkdir ${i}; cd ../; cp resp_tr_${i}.txt ROI_models/${i}/
↪resp_tr.txt; cp resp_te_${i}.txt ROI_models/${i}/resp_te.txt; cp cov_tr.txt ROI_models/
↪${i}/cov_tr.txt; cp cov_te.txt ROI_models/${i}/cov_te.txt; fi; done
```

```
# clean up files
! rm resp_*.txt
```

```
# clean up files
! rm cov_t*.txt
```

7.2 Algorithm & Modeling

7.2.1 Basis expansion using B-Splines

Now, set up a B-spline basis set that allows us to perform nonlinear regression using a linear model, using the following commands. This basis is deliberately chosen to not to be too flexible so that it can only model relatively slowly varying trends. To increase the flexibility of the model you can change the parameterization (e.g., by adding knot points to the B-spline basis or increasing the order of the interpolating polynomial). Note that in the neuroimaging literature, it is more common to use a polynomial basis expansion for this. Piecewise polynomials like B-splines are superior to polynomial basis expansions because they do not introduce a global curvature. For further details on the use of B-splines see Fraza et al.

```
# set this path to wherever your ROI_models folder is located (where you copied all of␣
↪the covariate & response text files to in Step 4)
data_dir = '/content/PCNtoolkit-demo/tutorials/BLR_protocol/ROI_models/'

# Create a cubic B-spline basis (used for regression)
xmin = 10#16 # xmin & xmax are the boundaries for ages of participants in the dataset
xmax = 95#90
B = create_bspline_basis(xmin, xmax)
# create the basis expansion for the covariates for each of the
for roi in roi_ids:
    print('Creating basis expansion for ROI:', roi)
    roi_dir = os.path.join(data_dir, roi)
    os.chdir(roi_dir)
    # create output dir
```

(continues on next page)

(continued from previous page)

```
    os.makedirs(os.path.join(roi_dir,'blr'), exist_ok=True)
    # load train & test covariate data matrices
    X_tr = np.loadtxt(os.path.join(roi_dir, 'cov_tr.txt'))
    X_te = np.loadtxt(os.path.join(roi_dir, 'cov_te.txt'))
    # add intercept column
    X_tr = np.concatenate((X_tr, np.ones((X_tr.shape[0],1))), axis=1)
    X_te = np.concatenate((X_te, np.ones((X_te.shape[0],1))), axis=1)
    np.savetxt(os.path.join(roi_dir, 'cov_int_tr.txt'), X_tr)
    np.savetxt(os.path.join(roi_dir, 'cov_int_te.txt'), X_te)

    # create Bspline basis set
    Phi = np.array([B(i) for i in X_tr[:,0]])
    Phis = np.array([B(i) for i in X_te[:,0]])
    X_tr = np.concatenate((X_tr, Phi), axis=1)
    X_te = np.concatenate((X_te, Phis), axis=1)
    np.savetxt(os.path.join(roi_dir, 'cov_bspline_tr.txt'), X_tr)
    np.savetxt(os.path.join(roi_dir, 'cov_bspline_te.txt'), X_te)
```

```
Creating basis expansion for ROI: lh_MeanThickness_thickness
Creating basis expansion for ROI: rh_MeanThickness_thickness
Creating basis expansion for ROI: lh_bankssts_thickness
Creating basis expansion for ROI: lh_caudalanteriorcingulate_thickness
Creating basis expansion for ROI: lh_superiorfrontal_thickness
Creating basis expansion for ROI: rh_superiorfrontal_thickness
```

7.2.2 Estimate normative model

Set up a variable (`data_dir`) that specifies the path to the ROI directories that were created in Step 7. Initiate two empty pandas data frames where the evaluation metrics are the column names, as follows; one will be used for overall test set evaluation (`blr_metrics`) and one will be used for site-specific test set evaluation (`blr_site_metrics`). After the normative model has been estimated, these data frames will be saved as individual csv files.

```
# Create pandas dataframes with header names to save out the overall and per-site model␣
↪evaluation metrics
blr_metrics = pd.DataFrame(columns = ['ROI', 'MSLL', 'EV', 'SMSE', 'RMSE', 'Rho'])
blr_site_metrics = pd.DataFrame(columns = ['ROI', 'site', 'MSLL', 'EV', 'SMSE', 'RMSE',
↪'Rho'])
```

Estimate the normative models using a for loop to iterate over brain regions. An important consideration is whether to re-scale or standardize the covariates or responses. Whilst this generally only has a minor effect on the final model accuracy, it has implications for the interpretation of models and how they are configured. If the covariates and responses are both standardized (`standardize = True`), the model will return standardized coefficients. If (as in this case) the response variables are not standardized (`standardized = False`), then the scaling both covariates and responses will be reflected in the estimated coefficients. Also, under the linear modeling approach employed here, if the coefficients are unstandardized and do not have a zero mean, it is necessary to add an intercept column to the design matrix (this is done above in step 9 (B-spline)). The estimate function uses a few specific arguments that are worthy of commenting on:

```
- alg = 'blr': specifies we should use Bayesian Linear Regression.
- optimizer = 'powell': use Powell's derivative-free optimization method (faster in this␣
↪case than L-BFGS)
```

(continues on next page)

(continued from previous page)

```
- savemodel = False: do not write out the final estimated model to disk
- saveoutput = False: return the outputs directly rather than writing them to disk
- standardize = False: Do not standardize the covariates or response variables
```

Warning: CRITICAL STEP: This code fragment will loop through each region of interest in the `roi_ids` list (created in step 4) using Bayesian Linear Regression and evaluate the model on the independent test set. In principle, we could estimate the normative models on the whole data matrix at once (e.g., with the response variables stored in a `n_subjects` by `n_brain_measures` NumPy array or a text file instead of saved out into separate directories). However, running the models iteratively gives some extra flexibility in that it does not require that the included subjects are the same for each of the brain measures.

```
# Loop through ROIs
for roi in roi_ids:
    print('Running ROI:', roi)
    roi_dir = os.path.join(data_dir, roi)
    os.chdir(roi_dir)

    # configure the covariates to use. Change *_bspline_* to *_int_* to
    cov_file_tr = os.path.join(roi_dir, 'cov_bspline_tr.txt')
    cov_file_te = os.path.join(roi_dir, 'cov_bspline_te.txt')

    # load train & test response files
    resp_file_tr = os.path.join(roi_dir, 'resp_tr.txt')
    resp_file_te = os.path.join(roi_dir, 'resp_te.txt')

    # run a basic model
    yhat_te, s2_te, nm, Z, metrics_te = estimate(cov_file_tr,
                                                 resp_file_tr,
                                                 testresp=resp_file_te,
                                                 testcov=cov_file_te,
                                                 alg = 'blr',
                                                 optimizer = 'powell',
                                                 savemodel = True,
                                                 saveoutput = False,
                                                 standardize = False)
    # save metrics
    blr_metrics.loc[len(blr_metrics)] = [roi, metrics_te['MSLL'][0], metrics_te['EXPV
↪'][0], metrics_te['SMSE'][0], metrics_te['RMSE'][0], metrics_te['Rho'][0]]

    # Compute metrics per site in test set, save to pandas df
    # load true test data
    X_te = np.loadtxt(cov_file_te)
    y_te = np.loadtxt(resp_file_te)
    y_te = y_te[:, np.newaxis] # make sure it is a 2-d array

    # load training data (required to compute the MSLL)
    y_tr = np.loadtxt(resp_file_tr)
    y_tr = y_tr[:, np.newaxis]

    for num, site in enumerate(sites):
```

(continues on next page)

(continued from previous page)

```
        y_mean_te_site = np.array([[np.mean(y_te[site])]])
        y_var_te_site = np.array([[np.var(y_te[site])]])
        yhat_mean_te_site = np.array([[np.mean(yhat_te[site])]])
        yhat_var_te_site = np.array([[np.var(yhat_te[site])]])

        metrics_te_site = evaluate(y_te[site], yhat_te[site], s2_te[site], y_mean_te_
↪site, y_var_te_site)

        site_name = site_names[num]
        blr_site_metrics.loc[len(blr_site_metrics)] = [roi, site_names[num], metrics_te_
↪site['MSLL'][0], metrics_te_site['EXPV'][0], metrics_te_site['SMSE'][0], metrics_te_
↪site['RMSE'][0], metrics_te_site['Rho'][0]]
```

```
Running ROI: lh_MeanThickness_thickness
Processing data in /content/PCNtoolkit-demo/tutorials/BLR_protocol/ROI_models/lh_
↪MeanThickness_thickness/resp_tr.txt
Estimating model  1 of 1
configuring BLR ( order 1 )
Using default hyperparameters
Optimization terminated successfully.
         Current function value: -1162.792820
         Iterations: 2
         Function evaluations: 47
Saving model meta-data...
Evaluating the model ...
```

```
/usr/local/lib/python3.7/dist-packages/pcntoolkit/model/bayesreg.py:187: LinAlgWarning:␣
↪Ill-conditioned matrix (rcond=1.15485e-18): result may not be accurate.
  invAXt = linalg.solve(self.A, X.T, check_finite=False)
/usr/local/lib/python3.7/dist-packages/pcntoolkit/model/bayesreg.py:187: LinAlgWarning:␣
↪Ill-conditioned matrix (rcond=4.51813e-19): result may not be accurate.
  invAXt = linalg.solve(self.A, X.T, check_finite=False)
```

```
Running ROI: rh_MeanThickness_thickness
Processing data in /content/PCNtoolkit-demo/tutorials/BLR_protocol/ROI_models/rh_
↪MeanThickness_thickness/resp_tr.txt
Estimating model  1 of 1
configuring BLR ( order 1 )
Using default hyperparameters
Optimization terminated successfully.
         Current function value: -1187.621858
         Iterations: 2
         Function evaluations: 47
Saving model meta-data...
Evaluating the model ...
Running ROI: lh_bankssts_thickness
Processing data in /content/PCNtoolkit-demo/tutorials/BLR_protocol/ROI_models/lh_
↪bankssts_thickness/resp_tr.txt
Estimating model  1 of 1
configuring BLR ( order 1 )
Using default hyperparameters
```

(continues on next page)

(continued from previous page)

```
Optimization terminated successfully.
         Current function value: -578.945257
         Iterations: 2
         Function evaluations: 46
Saving model meta-data...
Evaluating the model ...
Running ROI: lh_caudalanteriorcingulate_thickness
Processing data in /content/PCNtoolkit-demo/tutorials/BLR_protocol/ROI_models/lh_
↪caudalanteriorcingulate_thickness/resp_tr.txt
Estimating model  1 of 1
configuring BLR ( order 1 )
Using default hyperparameters
Optimization terminated successfully.
         Current function value: -235.509099
         Iterations: 3
         Function evaluations: 75
Saving model meta-data...
Evaluating the model ...
Running ROI: lh_superiorfrontal_thickness
Processing data in /content/PCNtoolkit-demo/tutorials/BLR_protocol/ROI_models/lh_
↪superiorfrontal_thickness/resp_tr.txt
Estimating model  1 of 1
configuring BLR ( order 1 )
Using default hyperparameters
Optimization terminated successfully.
         Current function value: -716.547377
         Iterations: 3
         Function evaluations: 91
Saving model meta-data...
Evaluating the model ...
Running ROI: rh_superiorfrontal_thickness
Processing data in /content/PCNtoolkit-demo/tutorials/BLR_protocol/ROI_models/rh_
↪superiorfrontal_thickness/resp_tr.txt
Estimating model  1 of 1
configuring BLR ( order 1 )
Using default hyperparameters
Optimization terminated successfully.
         Current function value: -730.639309
         Iterations: 2
         Function evaluations: 45
Saving model meta-data...
Evaluating the model ...
```

7.3 Evaluation & Interpretation

7.3.1 Describe the normative model performance

In step 11, when we looped over each region of interest in the `roi_ids` list (created in step 4) and evaluated the normative model on the independent test set, it also computed the evaluation metrics such as the explained variance, mean standardized log-loss and Pearson correlation between true and predicted test responses. The evaluation metrics were calculated for the full test set and calculated separately for each scanning site. The metrics were saved out to a csv file. In this step we load the evaluation metrics into a panads data frame and use the describe function to show the range, mean, and standard deviation of each of the evaluation metrics. Table 2 shows how to interpret the ranges/directions of good model fit.

```
# Overall test set evaluation metrics
print(blr_metrics['EV'].describe())
print(blr_metrics['MSLL'].describe())
print(blr_metrics['SMSE'].describe())
print(blr_metrics['Rho'].describe())
```

```
count    6.000000
mean     0.216747
std      0.114371
min      0.063284
25%      0.161901
50%      0.204015
75%      0.264058
max      0.397232
Name: EV, dtype: float64
count    6.000000
mean    -0.131996
std      0.080019
min     -0.267055
25%     -0.157321
50%     -0.120775
75%     -0.089765
max     -0.034441
Name: MSLL, dtype: float64
count    6.000000
mean     0.784798
std      0.114679
min      0.603410
25%      0.736912
50%      0.798426
75%      0.841000
max      0.936928
Name: SMSE, dtype: float64
count    6.000000
mean     0.452088
std      0.126840
min      0.257838
25%      0.403631
50%      0.450319
75%      0.513867
```

(continues on next page)

(continued from previous page)

```
max      0.630935
Name: Rho, dtype: float64
```

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

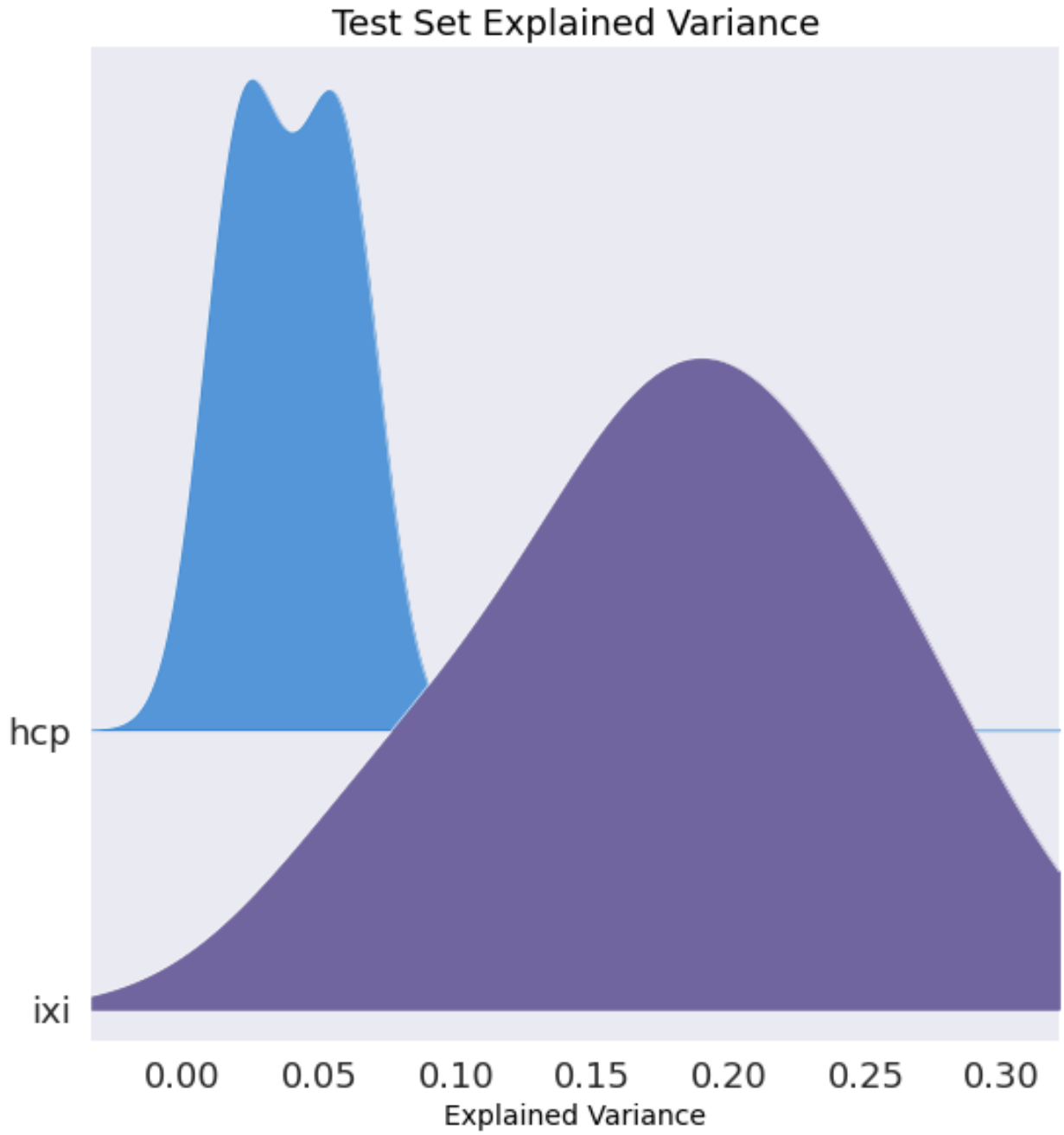
7.3.2 Visualize normative model outputs

7.3.3 Figure 4A viz

```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

def color_gradient(x=0.0, start=(0, 0, 0), stop=(1, 1, 1)):
    r = np.interp(x, [0, 1], [start[0], stop[0]])
    g = np.interp(x, [0, 1], [start[1], stop[1]])
    b = np.interp(x, [0, 1], [start[2], stop[2]])
    return r, g, b

plt.figure(dpi=380)
fig, axes = joypy.joyplot(blr_site_metrics, column=['EV'], overlap=2.5, by="site", ylim=
↪'own', fill=True, figsize=(8,8)
                          , legend=False, xlabels=True, ylabels=True, colormap=lambda x:␣
↪color_gradient(x, start=(.08, .45, .8),stop=(.8, .34, .44))
                          , alpha=0.6, linewidth=.5, linecolor='w', fade=True)
plt.title('Test Set Explained Variance', fontsize=18, color='black', alpha=1)
plt.xlabel('Explained Variance', fontsize=14, color='black', alpha=1)
plt.ylabel('Site', fontsize=14, color='black', alpha=1)
plt.show
```

```
<function matplotlib.pyplot.show>
```

```
<Figure size 2280x1520 with 0 Axes>
```



The code used to create the visualizations shown in Figure 4 panels B-F, can be found in this notebook.

7.3.4 Post-Hoc analysis ideas

The code for running SVM classification and classical case vs. control t-testing on the outputs of normative modeling can be found in this notebook.

The code for running other predictive models (regression, using the outputs of normative modeling as predictive features) can be found in this notebook.

The code for transfering a pre-trained normative model to a new dataset can be found in this notebook.

CHAPTER

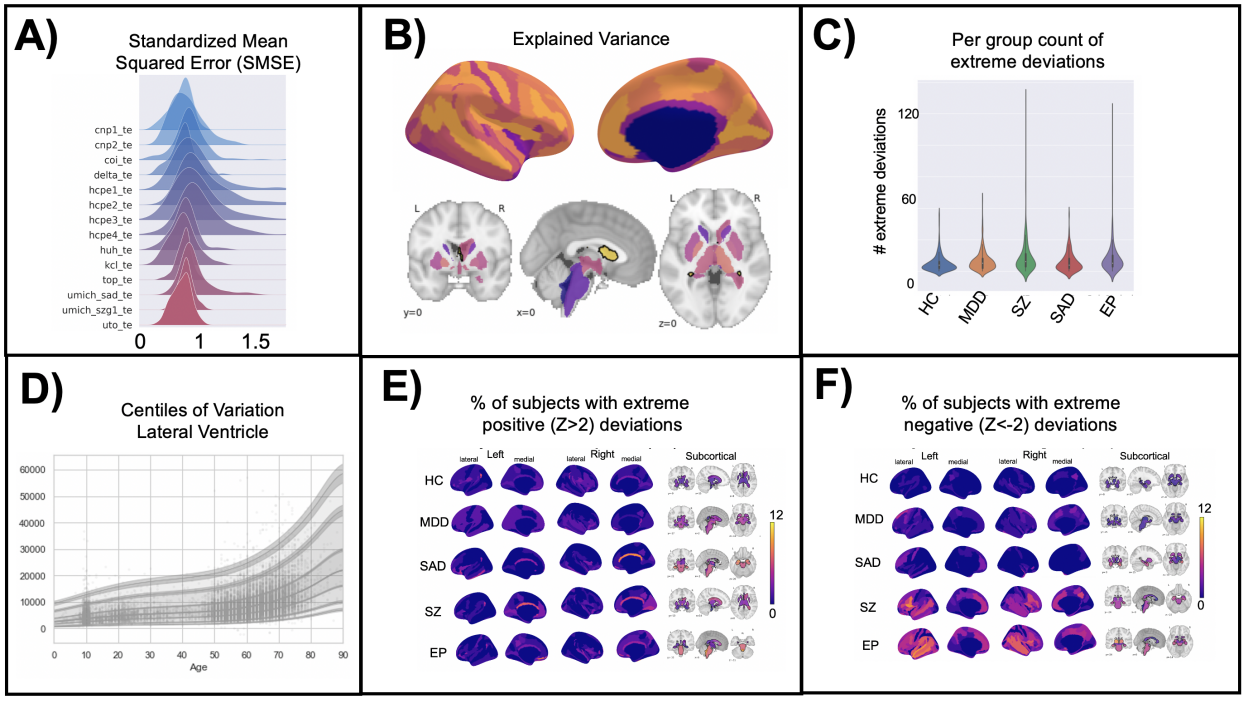
EIGHT

VISUALIZATION OF NORMATIVE MODELING OUTPUTS

The Normative Modeling Framework for Computational Psychiatry. Nature Protocols. https://www.nature.com/articles/s41596-022-00696-5.

Created by Saige Rutherford

We have also built an app for interactively viewing the evaluation metrics.



8.1 Brain space extreme deviation counts

Count the number of extreme (positive & negative) deviations at each brain region and visualize the count for each hemisphere.

```
! 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
```

```
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 with your partner 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')
```

```
positive_left_z2.describe()
```

```
positive_right_z2.describe()
```

```
positive_sc_z2.describe()
```

```
negative_left_z2.describe()
```

```
negative_right_z2.describe()
```

```
negative_sc_z2.describe()
```

```
! pip install nilearn
```

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

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

```
Dataset created in /root/nilearn_data/destrieux_surface

Downloading data from https://www.nitrc.org/frs/download.php/9343/lh.aparc.a2009s.annot .
↪..
```

```
...done. (1 seconds, 0 min)
```

```
Downloading data from https://www.nitrc.org/frs/download.php/9342/rh.aparc.a2009s.annot .
↪..
```

```
...done. (1 seconds, 0 min)
```

```
# 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()
```

8.1.1 Extreme positive deviation viz

```
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)
```

```
# you can click around in 3D space on this visualization. Scroll in/out, move the brain␣
↪around, etc. Have fun with it :)
view = plotting.view_surf(fsaverage.infl_right, parcellation_positive_r, threshold=None,␣
↪symmetric_cmap=False, cmap='plasma', bg_map=fsaverage.sulc_right)

view
```

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

view
```

8.1.2 Extreme negative deviation viz

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

view
```

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

view
```

8.2 Violin plots of extreme deviations

We will 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')]
```

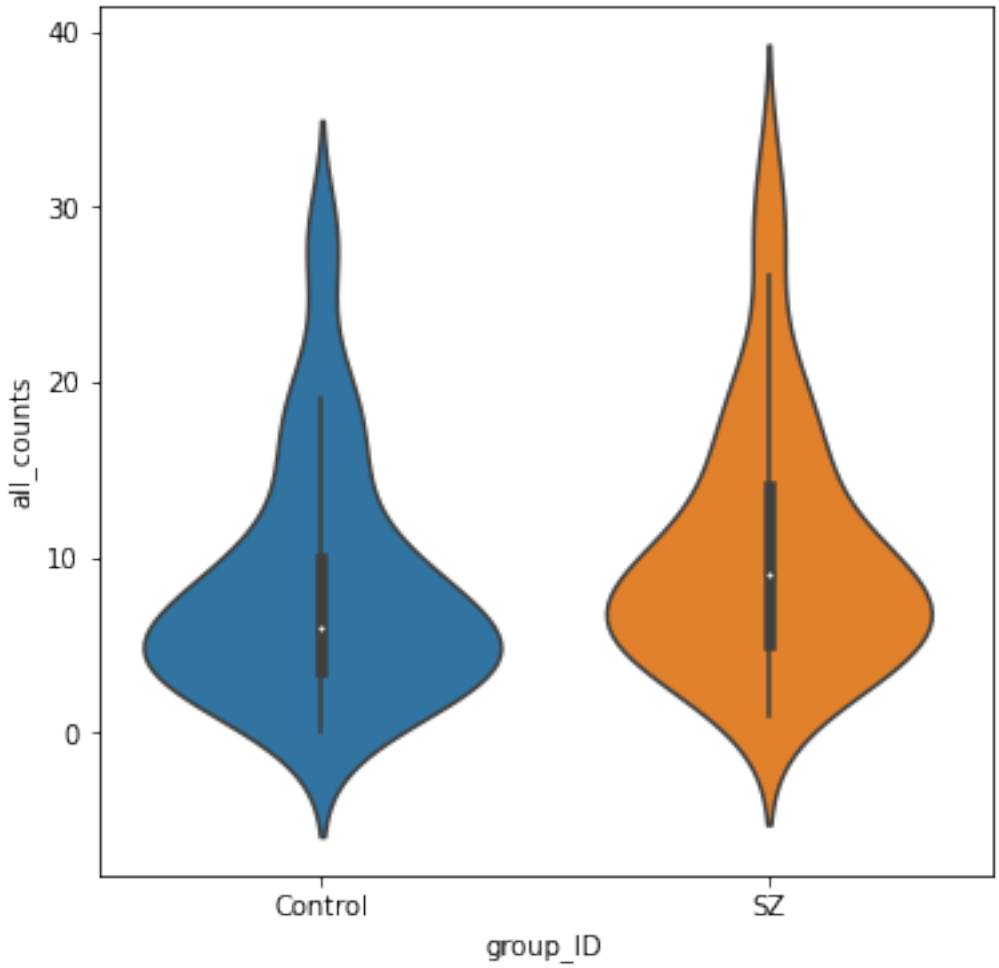
```
deviation_counts['positive_count'] = deviation_counts[deviation_counts >= 2].
↪count(axis=1)
```

```
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
```



8.3 Centile visualization

The code used to visualize the centiles of variation can be found in this notebook.

CHAPTER

NINE

POST-HOC ANALYSIS ON NORMATIVE MODELING OUTPUTS

The Normative Modeling Framework for Computational Psychiatry. Nature Protocols. https://www.nature.com/articles/s41596-022-00696-5.

Created by Saige Rutherford

9.1 SVM classification

Classify schizophrenia group from controls using cortical thickness deviation scores (z-scores) and then the true cortical thickness data to see which type of data better separates the groups.

```
! git clone https://github.com/predictive-clinical-neuroscience/PCNtoolkit-demo.git
```

```
Cloning into 'PCNtoolkit-demo'...
remote: Enumerating objects: 855, done.[K
remote: Counting objects: 100% (855/855), done.[K
remote: Compressing objects: 100% (737/737), done.[K
remote: Total 855 (delta 278), reused 601 (delta 101), pack-reused 0[K
Receiving objects: 100% (855/855), 18.07 MiB | 16.65 MiB/s, done.
Resolving deltas: 100% (278/278), done.
```

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
os.chdir('/content/PCNtoolkit-demo/')
```

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

```
from sklearn import svm
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold
```

```
Z_df.dropna(subset=['group'], inplace=True)
```

```
Z_df['group'] = Z_df['group'].replace("SZ",0)
```

```
Z_df['group'] = Z_df['group'].replace("Control",1)
```

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

```
cortical_thickness = Z_df.loc[:, Z_df.columns.str.endswith('_thickness')]
```

```
# Data IO and generation
X1 = deviations
X2 = cortical_thickness
y = Z_df['group']
n_samples, n_features = X1.shape
random_state = np.random.RandomState(0)
```

```
X1 = X1.to_numpy()
```

```
X2 = X2.to_numpy()
```

```
y = y.astype(int)
```

```
y = y.to_numpy()
```

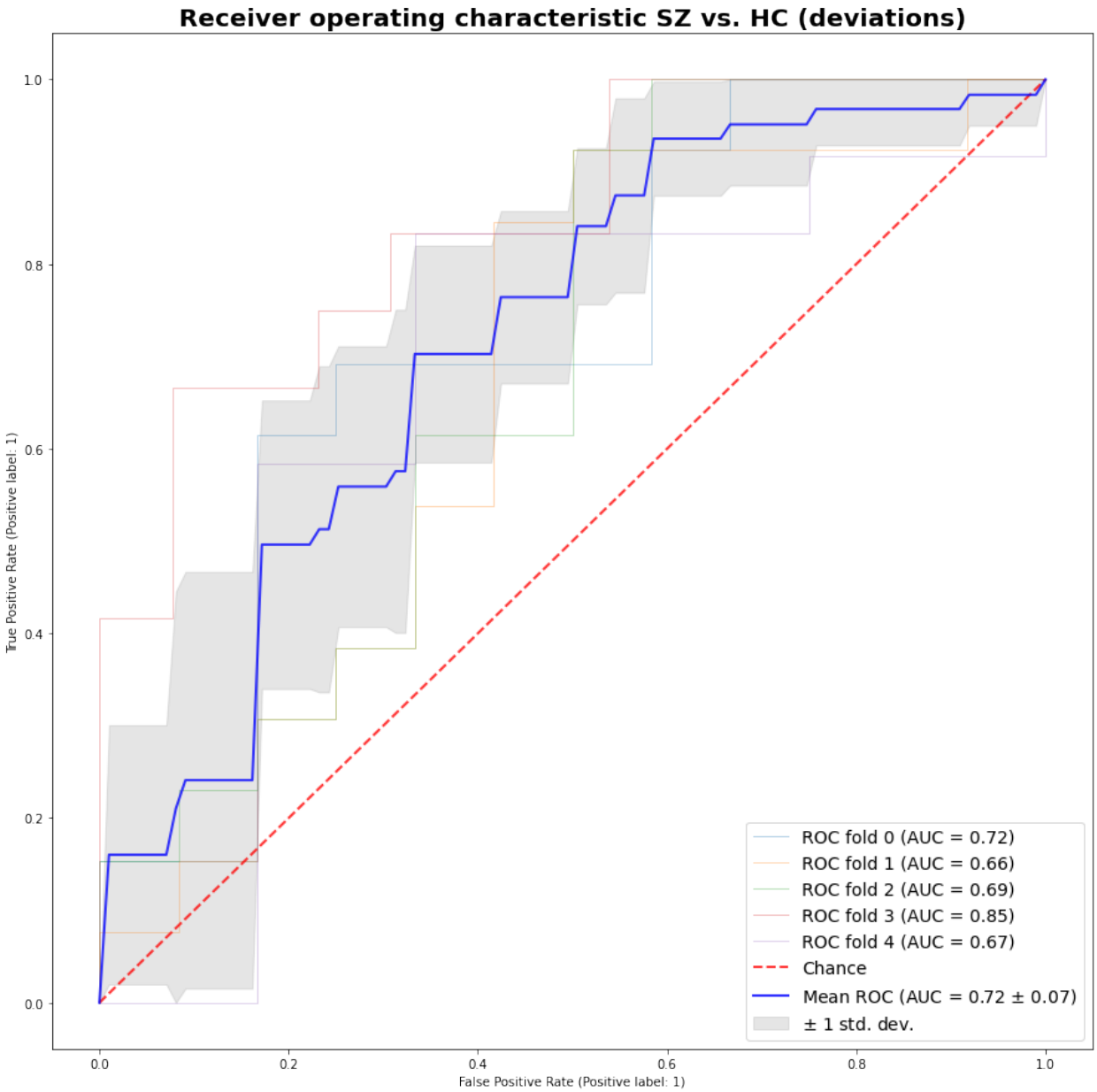
9.1.1 SVM using deviation scores as features

```
# Run classifier with cross-validation and plot ROC curves
cv = StratifiedKFold(n_splits=5)
classifier = svm.SVC(kernel='linear', probability=True,
                     random_state=random_state)

tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots(figsize=(15,15))
parameters = {'axes.labelsize': 20,
          'axes.titlesize': 25, 'xtick.labelsize':16,'ytick.labelsize':16,'legend.
↪fontsize':14,'legend.title_fontsize':16}
plt.rcParams.update(parameters)

for i, (train, test) in enumerate(cv.split(X1, y)):
    classifier.fit(X1[train], y[train])
    viz = plot_roc_curve(classifier, X1[test], y[test],
                         name='ROC fold {}'.format(i),
                         alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
```

(continues on next page)

(continued from previous page)

```
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05])
ax.set_title('Receiver operating characteristic SZ vs. HC (deviations)', fontweight="bold
↪", size=20)
ax.legend(loc="lower right")
plt.show()
```



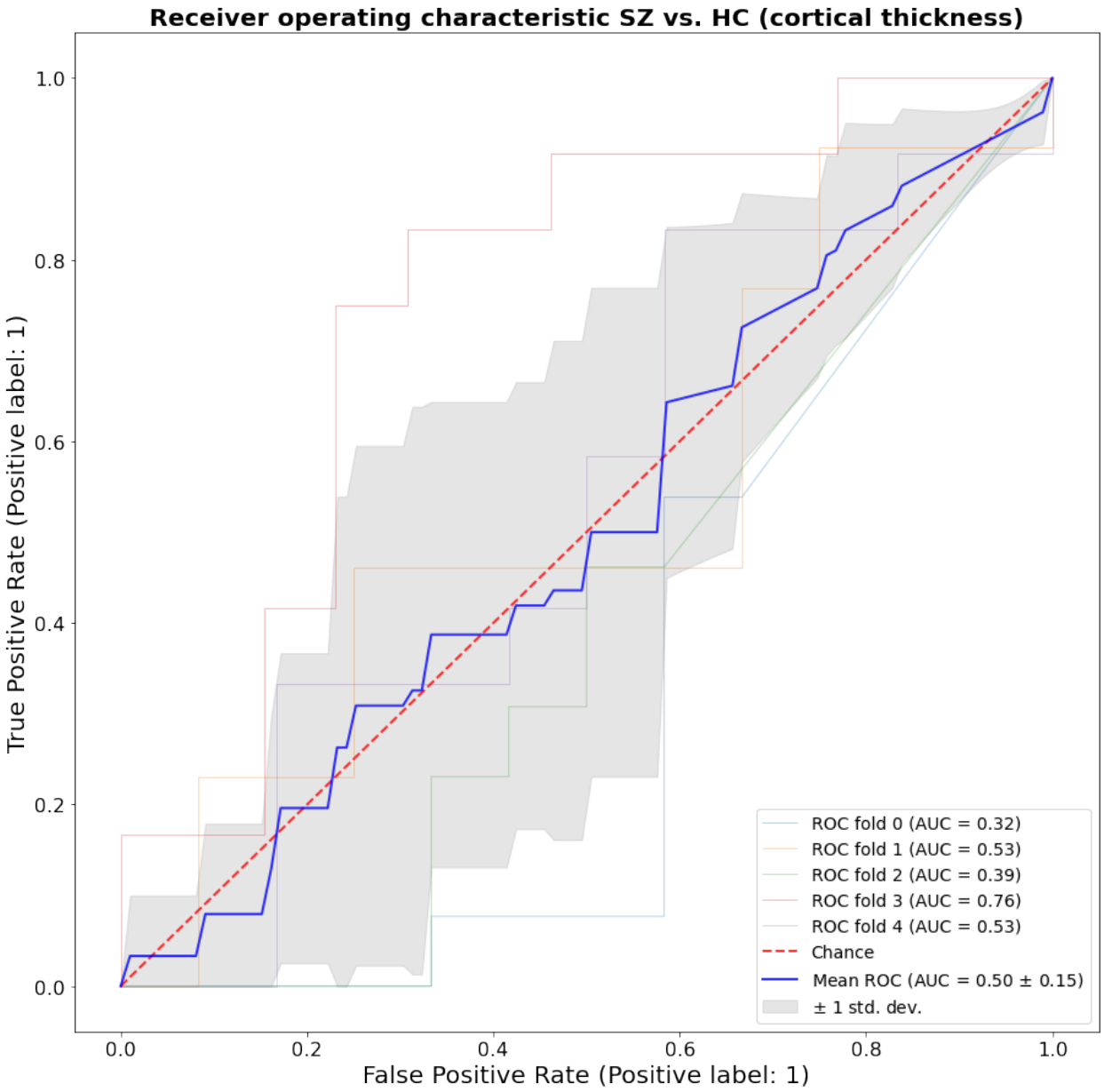
9.1.2 SVM using true cortical thickness data as features

```
# Run classifier with cross-validation and plot ROC curves
cv = StratifiedKFold(n_splits=5)
classifier = svm.SVC(kernel='linear', probability=True,
                     random_state=random_state)

tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots(figsize=(15,15))
```

(continues on next page)

(continued from previous page)

```
parameters = {'axes.labelsize': 20,
          'axes.titlesize': 25, 'xtick.labelsize':16,'ytick.labelsize':16,'legend.
↪fontsize':14,'legend.title_fontsize':16}
plt.rcParams.update(parameters)

for i, (train, test) in enumerate(cv.split(X2, y)):
    classifier.fit(X2[train], y[train])
    viz = plot_roc_curve(classifier, X2[test], y[test],
                         name='ROC fold {}'.format(i),
                         alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.2f $\pm$ %0.2f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05])
ax.set_title('Receiver operating characteristic SZ vs. HC (cortical thickness)',␣
↪fontweight="bold", size=20)
ax.legend(loc="lower right")
plt.show()
```



9.2 Classical case-control testing

```
! pip install statsmodels
```

```
from scipy.stats import ttest_ind
from statsmodels.stats import multitest
```

```
SZ = Z_df.query('group == 0')
HC = Z_df.query('group == 1')
```

9.2.1 Mass univariate two sample t-tests on deviation score maps

```
SZ_deviations = SZ.loc[:, SZ.columns.str.contains('Z_predict')]
```

```
HC_deviations = HC.loc[:, HC.columns.str.contains('Z_predict')]
```

```
z_cols = SZ_deviations.columns
```

```
sz_hc_pvals_z = pd.DataFrame(columns={'roi','pval', 'tstat','fdr_pval'})
for index, column in enumerate(z_cols):
    test = ttest_ind(SZ_deviations[column], HC_deviations[column])
    sz_hc_pvals_z.loc[index, 'pval'] = test.pvalue
    sz_hc_pvals_z.loc[index, 'tstat'] = test.statistic
    sz_hc_pvals_z.loc[index, 'roi'] = column
```

```
sz_hc_fdr_z = multitest.fdrcorrection(sz_hc_pvals_z['pval'], alpha=0.05, method='indep',␣
↪is_sorted=False)
```

```
sz_hc_pvals_z['fdr_pval'] = sz_hc_fdr_z[1]
```

```
sz_hc_z_sig_diff = sz_hc_pvals_z.query('pval < 0.05')
```

```
sz_hc_z_sig_diff
```

```
sz_hc_z_sig_diff.shape
```

```
(96, 4)
```

9.2.2 Mass univariate two sample t-tests on true cortical thickness data

```
SZ_cortical_thickness = SZ.loc[:, SZ.columns.str.endswith('_thickness')]
```

```
HC_cortical_thickness = HC.loc[:, HC.columns.str.endswith('_thickness')]
```

```
ct_cols = SZ_cortical_thickness.columns
```

```
sz_hc_pvals_ct = pd.DataFrame(columns={'roi','pval', 'tstat','fdr_pval'})
for index, column in enumerate(ct_cols):
    test = ttest_ind(SZ_cortical_thickness[column], HC_cortical_thickness[column])
    sz_hc_pvals_ct.loc[index, 'pval'] = test.pvalue
    sz_hc_pvals_ct.loc[index, 'tstat'] = test.statistic
    sz_hc_pvals_ct.loc[index, 'roi'] = column
```

```
sz_hc_fdr_ct = multitest.fdrcorrection(sz_hc_pvals_ct['pval'], alpha=0.05, method='indep
↪', is_sorted=False)
```

```
sz_hc_pvals_ct['fdr_pval'] = sz_hc_fdr_ct[1]
```

```
sz_hc_ct_sig_diff = sz_hc_pvals_ct.query('pval < 0.05')
```

```
sz_hc_ct_sig_diff
```

```
sz_hc_ct_sig_diff.shape
```

```
(67, 4)
```

CHAPTER

TEN

PREDICTIVE MODELING USING DEVIATION SCORES

The Normative Modeling Framework for Computational Psychiatry. Nature Protocols. https://www.nature.com/articles/s41596-022-00696-5.

Created by Saige Rutherford

```
! git clone https://github.com/predictive-clinical-neuroscience/PCNtoolkit-demo.git
```

```
import os
```

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

```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:95% !important; }</style>"))
```

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import stats, linalg
from sklearn import preprocessing, decomposition, linear_model, metrics
import warnings
```

```
# set fontsizes for matplotlib plots
baseline_fontsize = 12
SMALL_SIZE = 8 + baseline_fontsize
MEDIUM_SIZE = 10 + baseline_fontsize
BIGGER_SIZE = 12 + baseline_fontsize

plt.rc('font', size=SMALL_SIZE)          # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)     # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)    # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)    # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title
```

10.1 Load Data

```
hcp_z = np.load('data/hcpya_z.npy')
hcp_ct = np.load('data/hcpya_ct.npy')
gscores = np.load('data/hcpya_g.npy')
```

```
print(hcp_z.shape)
print(hcp_ct.shape)
print(gscores.shape)
```

```
(946, 187)
(946, 151)
(946,)
```

10.2 Create Train/Test Splits

```
# generate train/test splits
np.random.seed(42)
n_train = int(0.8 * hcp_z.shape[0])

train_idxs = np.random.choice(range(hcp_z.shape[0]), size=n_train, replace=False)
test_idxs = np.array([x for x in range(hcp_z.shape[0]) if x not in train_idxs])
```

```
train_data_z = hcp_z[train_idxs, :]
test_data_z = hcp_z[test_idxs, :]

train_data_ct = hcp_ct[train_idxs, :]
test_data_ct = hcp_ct[test_idxs, :]

train_phen = gscores[train_idxs]
test_phen = gscores[test_idxs]
```

```
# mean center train/test data (using train means)
train_mu_centered_z = (train_data_z - train_data_z.mean(axis=0))
test_mu_centered_z = (test_data_z - train_data_z.mean(axis=0))

train_mu_centered_ct = (train_data_ct - train_data_ct.mean(axis=0))
test_mu_centered_ct = (test_data_ct - train_data_ct.mean(axis=0))
```

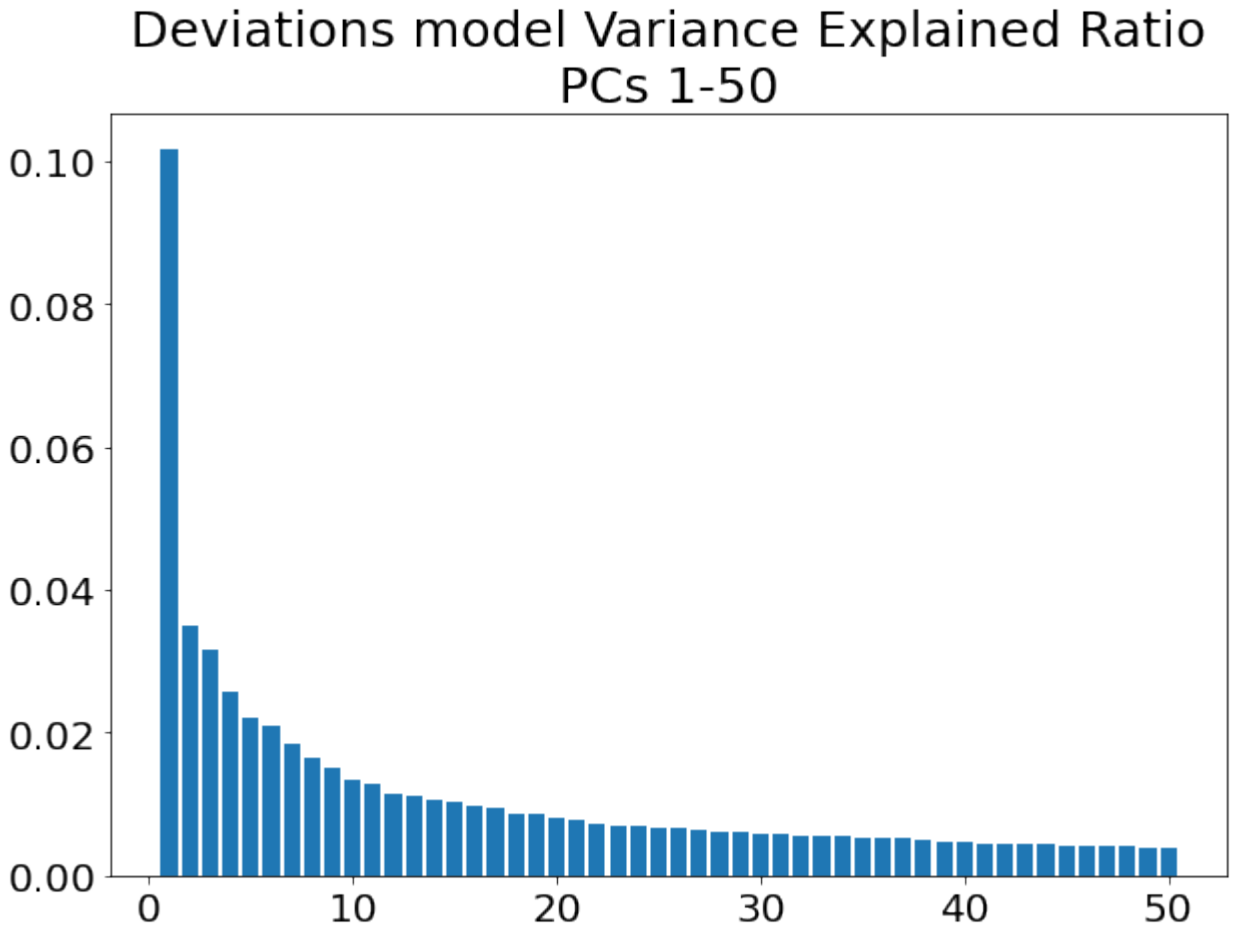
10.3 Principal Component Regression (BBS)

```
pca_model_z = decomposition.PCA(n_components=75).fit(train_data_z)
# from pca documentation, "the input data is centered but not scaled for each feature␣
↪before applying the SVD"
```

```
pca_model_ct = decomposition.PCA(n_components=75).fit(train_data_ct)
# from pca documentation, "the input data is centered but not scaled for each feature␣
↪before applying the SVD"
```

```
print(f'First PC explains {pca_model_z.explained_variance_ratio_[0]*100:.2f}% of the␣
↪total variance.')
plt.figure(figsize=(10, 7))
plt.bar(range(1, 51), pca_model_z.explained_variance_ratio_[1:51])
plt.title('Deviations model Variance Explained Ratio\nPCs 1-50', fontsize=25)
plt.show()
```

```
First PC explains 23.41% of the total variance.
```



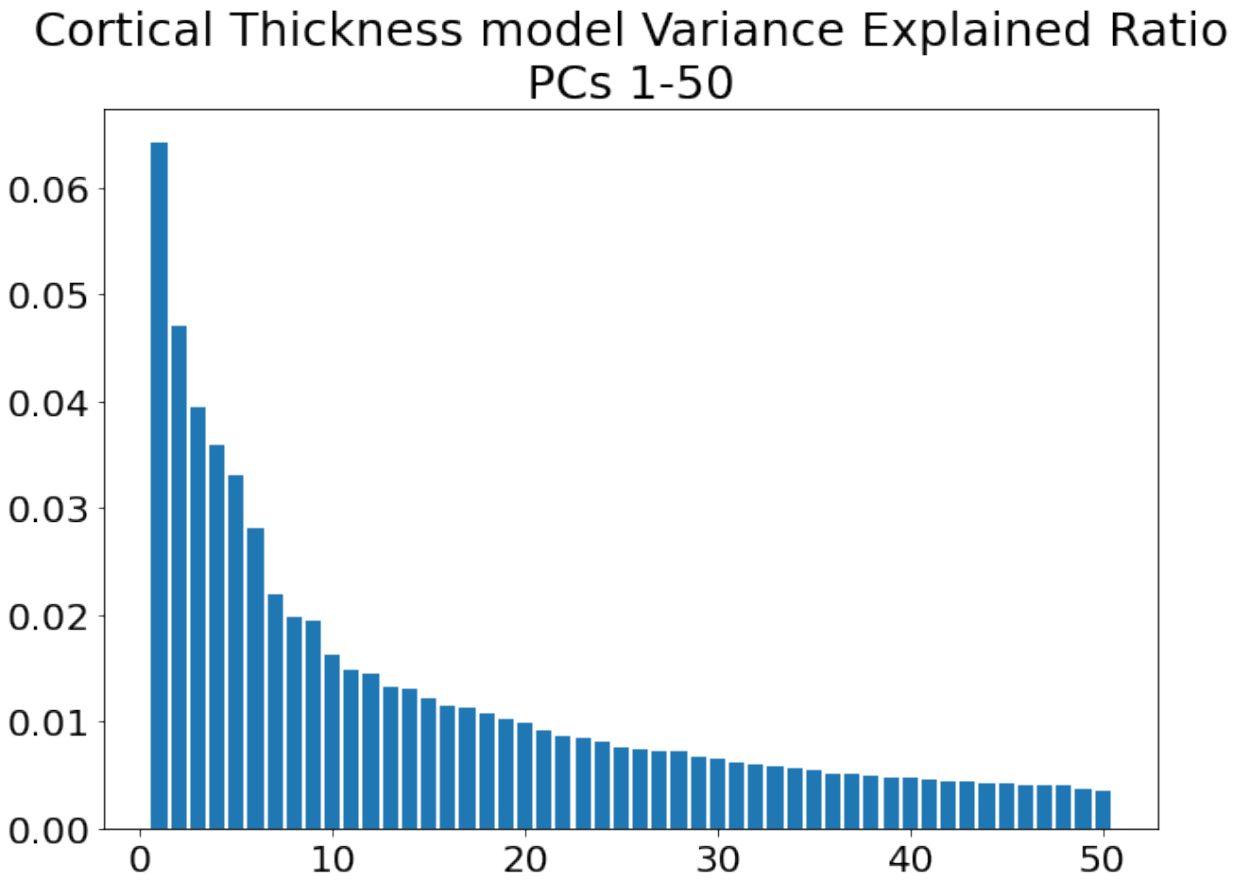

```
print(f'First PC explains {pca_model_ct.explained_variance_ratio_[0]*100:.2f}% of the␣
↪total variance.')
```

(continues on next page)

(continued from previous page)

```
plt.figure(figsize=(10, 7))
plt.bar(range(1, 51), pca_model_ct.explained_variance_ratio_[1:51])
plt.title('Cortical Thickness model Variance Explained Ratio\nPCs 1-50', fontsize=25)
plt.show()
```

```
First PC explains 24.28% of the total variance.
```




```
train_transformed_z = pca_model_z.transform(train_data_z)
test_transformed_z = pca_model_z.transform(test_data_z)
```

```
train_transformed_ct = pca_model_ct.transform(train_data_ct)
test_transformed_ct = pca_model_ct.transform(test_data_ct)
```

10.4 Fit Linear Regression Model

```
# fast OLS using matrix math
# we will check that this matches sklearn results later

# fit ols model on dimension reduced train data
train_features_z = np.hstack([np.ones((train_transformed_z.shape[0], 1)),
                              train_transformed_z])
train_features_inv_z = linalg.pinv2(train_features_z)
train_betas_z = np.dot(train_features_inv_z, train_phen)
train_pred_phen_z = np.dot(train_features_z, train_betas_z)

# fit ols model on dimension reduced test data
test_features_z = np.hstack([np.ones((test_transformed_z.shape[0], 1)),
                             test_transformed_z])
test_pred_phen_z = np.dot(test_features_z, train_betas_z)
```

```
# fast OLS using matrix math
# we will check that this matches sklearn results later

# fit ols model on dimension reduced train data
train_features_ct = np.hstack([np.ones((train_transformed_ct.shape[0], 1)),
                              train_transformed_ct])
train_features_inv_ct = linalg.pinv2(train_features_ct)
train_betas_ct = np.dot(train_features_inv_ct, train_phen)
train_pred_phen_ct = np.dot(train_features_ct, train_betas_ct)

# fit ols model on dimension reduced test data
test_features_ct = np.hstack([np.ones((test_transformed_ct.shape[0], 1)),
                             test_transformed_ct])
test_pred_phen_ct = np.dot(test_features_ct, train_betas_ct)
```

```
# OLS using sklearn

lr_model_z = linear_model.LinearRegression(fit_intercept=True, normalize=False)
lr_model_z.fit(train_transformed_z, train_phen)
train_pred_phen_lr_model_z = lr_model_z.predict(train_transformed_z)
test_pred_phen_lr_model_z = lr_model_z.predict(test_transformed_z)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:155: FutureWarning:
↪'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave the␣
↪normalize parameter to its default value to silence this warning. The default behavior␣
↪of this estimator is to not do any normalization. If normalization is needed please␣
↪use sklearn.preprocessing.StandardScaler instead.
  FutureWarning,
```

```
# OLS using sklearn

lr_model_ct = linear_model.LinearRegression(fit_intercept=True, normalize=False)
lr_model_ct.fit(train_transformed_ct, train_phen)
train_pred_phen_lr_model_ct = lr_model_ct.predict(train_transformed_ct)
test_pred_phen_lr_model_ct = lr_model_ct.predict(test_transformed_ct)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:155: FutureWarning:
↪'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave the␣
↪normalize parameter to its default value to silence this warning. The default behavior␣
↪of this estimator is to not do any normalization. If normalization is needed please␣
↪use sklearn.preprocessing.StandardScaler instead.
  FutureWarning,
```

```
# ensure matrix math predictions and sklearn predictions are accurate to 5 decimals
assert np.allclose(np.round(train_pred_phen_z - train_pred_phen_lr_model_z, 5), 0),
↪'Failed'
assert np.allclose(np.round(test_pred_phen_z - test_pred_phen_lr_model_z, 5), 0), 'Failed
↪'
print('Passed')
```

```
Passed
```

```
# ensure matrix math predictions and sklearn predictions are accurate to 5 decimals
assert np.allclose(np.round(train_pred_phen_ct - train_pred_phen_lr_model_ct, 5), 0),
↪'Failed'
assert np.allclose(np.round(test_pred_phen_ct - test_pred_phen_lr_model_ct, 5), 0),
↪'Failed'
print('Passed')
```

```
Passed
```

10.5 Mean Squared/Absolute Error of Predictions

```
train_r2_z = metrics.r2_score(train_phen, train_pred_phen_lr_model_z)
train_mae_z = metrics.mean_absolute_error(train_phen, train_pred_phen_lr_model_z)
test_mae_z = metrics.mean_absolute_error(test_phen, test_pred_phen_lr_model_z)
train_mae_z = metrics.mean_squared_error(train_phen, train_pred_phen_lr_model_z)
test_mae_z = metrics.mean_squared_error(test_phen, test_pred_phen_lr_model_z)
print(f'Deviation model Train R^2: {train_r2_z:.3f}')
print(f'Deviation model Train MAE: {train_mae_z:.3f}')
print(f'Deviation model Test MAE: {test_mae_z:.3f}')
print(f'Deviation model Train MSE: {train_mae_z:.3f}')
print(f'Deviation model Test MSE: {test_mae_z:.3f}')
```

```
Deviation model Train R^2: 0.255
Deviation model Train MAE: 0.532
Deviation model Test MAE: 0.741
Deviation model Train MSE: 0.532
Deviation model Test MSE: 0.741
```

```
train_r2_ct = metrics.r2_score(train_phen, train_pred_phen_lr_model_ct)
train_mae_ct = metrics.mean_absolute_error(train_phen, train_pred_phen_lr_model_ct)
test_mae_ct = metrics.mean_absolute_error(test_phen, test_pred_phen_lr_model_ct)
train_mae_ct = metrics.mean_squared_error(train_phen, train_pred_phen_lr_model_ct)
test_mae_ct = metrics.mean_squared_error(test_phen, test_pred_phen_lr_model_ct)
```

(continues on next page)

(continued from previous page)

```
print(f'Cortical thickness model Train R^2: {train_r2_ct:.3f}')
print(f'Cortical thickness model Train MAE: {train_mae_ct:.3f}')
print(f'Cortical thickness model Test MAE: {test_mae_ct:.3f}')
print(f'Cortical thickness model Train MSE: {train_mae_ct:.3f}')
print(f'Cortical thickness model Test MSE: {test_mae_ct:.3f}')
```

```
Cortical thickness model Train R^2: 0.185
Cortical thickness model Train MAE: 0.582
Cortical thickness model Test MAE: 0.830
Cortical thickness model Train MSE: 0.582
Cortical thickness model Test MSE: 0.830
```

10.6 BBS Cross Validation

```
def bbs(X, y, n_components, n_cv_splits, pred_summary_function, verbose=False):
    assert X.shape[0] == y.shape[0]

    fold_accs_train = []
    fold_accs_test = []
    np.random.seed(42)
    shuffled_idxs = np.random.choice(range(X.shape[0]), size=X.shape[0], replace=False)
    for fold_i, test_idxs in enumerate(np.array_split(shuffled_idxs, n_cv_splits)):
        train_mask = np.ones(X.shape[0], np.bool)
        train_mask[test_idxs] = 0

        # create train/text X, y
        train_X, test_X = X[train_mask, :], X[test_idxs, :]
        train_y, test_y = y[train_mask], y[test_idxs]

        # mean center columns using train data only
        train_X_mu = train_X.mean(axis=0)
        train_X = train_X - train_X_mu
        test_X = test_X - train_X_mu

        # fit pca
        if verbose:
            print(f'CV Fold: {fold_i+1:<10} Fitting PCA model...')
        pca_model = decomposition.PCA(n_components=n_components).fit(train_X)

        # dimension reduce train/test data
        train_X = pca_model.transform(train_X)
        test_X = pca_model.transform(test_X)

        # fit OLS model
        if verbose:
            print(f'CV Fold: {fold_i+1:<10} Fitting Linear Regression model...')
        lr_model = linear_model.LinearRegression(fit_intercept=True, normalize=False)
        lr_model.fit(train_X, train_y)
```

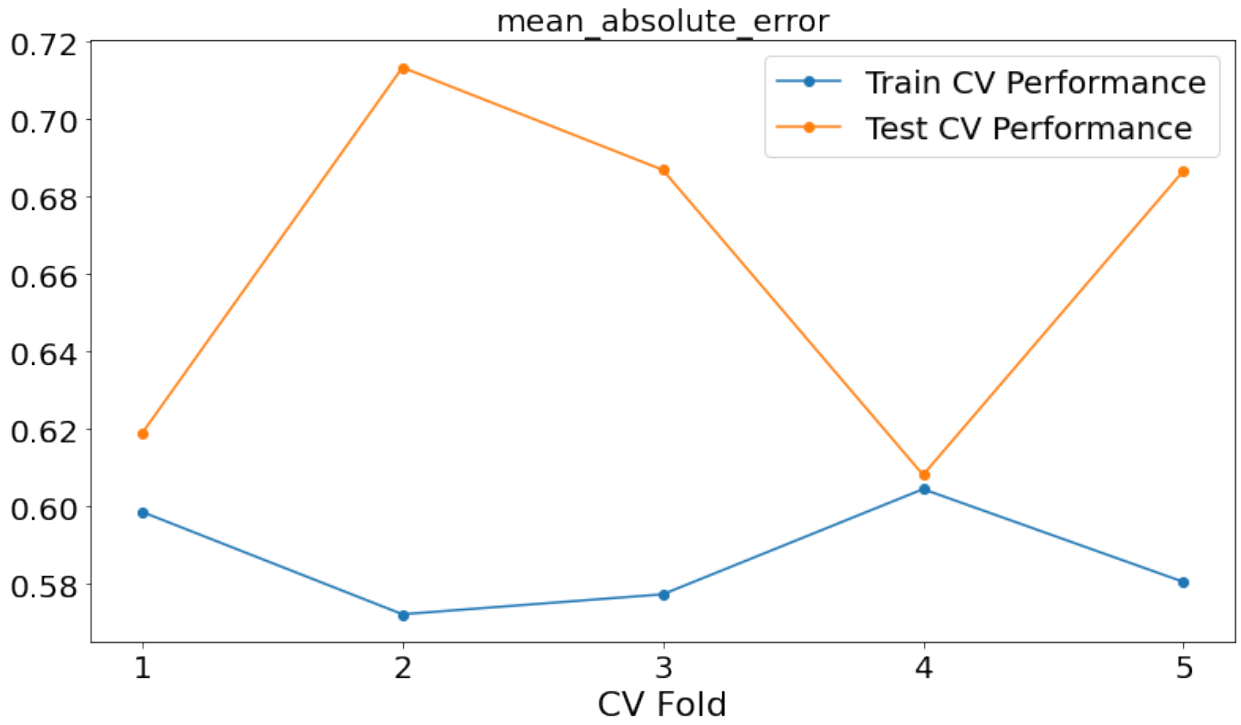
(continues on next page)

(continued from previous page)

```
        train_pred = lr_model.predict(train_X)
        test_pred = lr_model.predict(test_X)

        fold_accs_train.append(pred_summary_function(train_y, train_pred))
        fold_accs_test.append(pred_summary_function(test_y, test_pred))

        if verbose:
            print(f'CV Fold: {fold_i+1:<10} Train MAE: {round(fold_accs_train[-1], 3):
↪<10} Test MAE: {round(fold_accs_test[-1], 3):<10}')


    plt.figure(figsize=(13, 7))
    plt.plot(range(1, len(fold_accs_train)+1), fold_accs_train, linestyle='-', marker='o
↪', color='C0', label='Train CV Performance')
    plt.plot(range(1, len(fold_accs_test)+1), fold_accs_test, linestyle='-', marker='o',␣
↪color='C1', label='Test CV Performance')
    plt.title(pred_summary_function.__name__, fontsize=20)
    plt.xticks(range(1, len(fold_accs_test)+1))
    plt.xlabel('CV Fold')
    plt.legend(fontsize=20)
    plt.show()

    return fold_accs_train, fold_accs_test
```

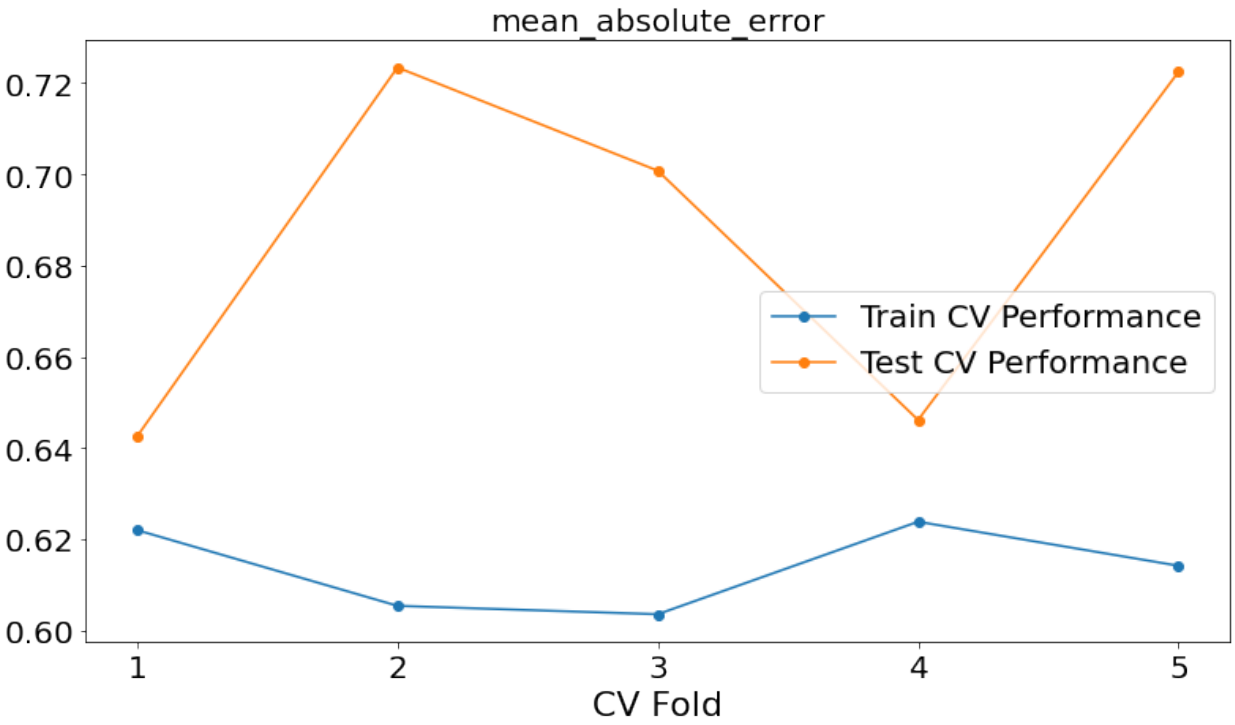
```
fold_accs_train_z, fold_accs_test_z = bbs(hcp_z, gscores, n_components=75, n_cv_splits=5,
↪ pred_summary_function=metrics.mean_absolute_error, verbose=True)
```

```
CV Fold: 1          Fitting PCA model...
CV Fold: 1          Fitting Linear Regression model...
CV Fold: 1          Train MAE: 0.599      Test MAE: 0.619
CV Fold: 2          Fitting PCA model...
CV Fold: 2          Fitting Linear Regression model...
CV Fold: 2          Train MAE: 0.572      Test MAE: 0.713
CV Fold: 3          Fitting PCA model...
CV Fold: 3          Fitting Linear Regression model...
CV Fold: 3          Train MAE: 0.577      Test MAE: 0.687
CV Fold: 4          Fitting PCA model...
CV Fold: 4          Fitting Linear Regression model...
CV Fold: 4          Train MAE: 0.604      Test MAE: 0.608
CV Fold: 5          Fitting PCA model...
CV Fold: 5          Fitting Linear Regression model...
CV Fold: 5          Train MAE: 0.581      Test MAE: 0.687
```




```
fold_accs_train_ct, fold_accs_test_ct = bbs(hcp_ct, gscores, n_components=75, n_cv_
↪splits=5, pred_summary_function=metrics.mean_absolute_error, verbose=True)
```

```
CV Fold: 1          Fitting PCA model...
CV Fold: 1          Fitting Linear Regression model...
CV Fold: 1          Train MAE: 0.622      Test MAE: 0.643
CV Fold: 2          Fitting PCA model...
CV Fold: 2          Fitting Linear Regression model...
CV Fold: 2          Train MAE: 0.605      Test MAE: 0.723
CV Fold: 3          Fitting PCA model...
CV Fold: 3          Fitting Linear Regression model...
CV Fold: 3          Train MAE: 0.604      Test MAE: 0.701
CV Fold: 4          Fitting PCA model...
CV Fold: 4          Fitting Linear Regression model...
CV Fold: 4          Train MAE: 0.624      Test MAE: 0.646
CV Fold: 5          Fitting PCA model...
CV Fold: 5          Fitting Linear Regression model...
CV Fold: 5          Train MAE: 0.614      Test MAE: 0.722
```



10.7 Connectome Predictive Modelling

```
# correlation train_brain with train_phenotype
train_z_pheno_corr_p = [stats.pearsonr(train_data_z[:, i], train_phen) for i in␣
↪range(train_data_z.shape[1])]  # train_pheno_corr_p: (259200, )
```

```
# correlation train_brain with train_phenotype
train_ct_pheno_corr_p = [stats.pearsonr(train_data_ct[:, i], train_phen) for i in␣
↪range(train_data_ct.shape[1])]  # train_pheno_corr_p: (259200, )
```

```
# split into positive and negative correlations
# and keep edges with p values below threshold
pval_threshold = 0.01

train_z_corrs = np.array([x[0] for x in train_z_pheno_corr_p])
train_z_pvals = np.array([x[1] for x in train_z_pheno_corr_p])

keep_edges_pos_z = (train_z_corrs > 0) & (train_z_pvals < pval_threshold)
keep_edges_neg_z = (train_z_corrs < 0) & (train_z_pvals < pval_threshold)

train_ct_corrs = np.array([x[0] for x in train_ct_pheno_corr_p])
train_ct_pvals = np.array([x[1] for x in train_ct_pheno_corr_p])

keep_edges_pos_ct = (train_ct_corrs > 0) & (train_ct_pvals < pval_threshold)
keep_edges_neg_ct = (train_ct_corrs < 0) & (train_ct_pvals < pval_threshold)
```

```
print(f'number of positive Z features kept = {np.sum(keep_edges_pos_z)}')
print(f'number of negative Z features kept = {np.sum(keep_edges_neg_z)}')
print(f'number of positive CT features kept = {np.sum(keep_edges_pos_ct)}')
print(f'number of negative CT features kept = {np.sum(keep_edges_neg_ct)}')
```

```
number of positive Z features kept = 37
number of negative Z features kept = 2
number of positive CT features kept = 15
number of negative CT features kept = 1
```

```
train_pos_edges_sum_z = train_data_z[:, keep_edges_pos_z].sum(1)
train_neg_edges_sum_z = train_data_z[:, keep_edges_neg_z].sum(1)
```

```
train_pos_edges_sum_ct = train_data_ct[:, keep_edges_pos_ct].sum(1)
train_neg_edges_sum_ct = train_data_ct[:, keep_edges_neg_ct].sum(1)
```

```
fit_pos_z = linear_model.LinearRegression(fit_intercept=True, normalize=False).fit(train_
↪pos_edges_sum_z.reshape(-1, 1), train_phen)
fit_neg_z = linear_model.LinearRegression(fit_intercept=True, normalize=False).fit(train_
↪neg_edges_sum_z.reshape(-1, 1), train_phen)
```

```
fit_pos_ct = linear_model.LinearRegression(fit_intercept=True, normalize=False).
↪fit(train_pos_edges_sum_ct.reshape(-1, 1), train_phen)
fit_neg_ct = linear_model.LinearRegression(fit_intercept=True, normalize=False).
↪fit(train_neg_edges_sum_ct.reshape(-1, 1), train_phen)
```

```
pos_error_z = metrics.mean_absolute_error(train_phen, fit_pos_z.predict(train_pos_edges_
↪sum_z.reshape(-1, 1)))
neg_error_z = metrics.mean_absolute_error(train_phen, fit_neg_z.predict(train_neg_edges_
↪sum_z.reshape(-1, 1)))
pos_error_ct = metrics.mean_absolute_error(train_phen, fit_pos_ct.predict(train_pos_
↪edges_sum_ct.reshape(-1, 1)))
neg_error_ct = metrics.mean_absolute_error(train_phen, fit_neg_ct.predict(train_neg_
↪edges_sum_ct.reshape(-1, 1)))

print(f'Training Error (MAE) (Positive Z Features Model) = {pos_error_z:.3f}')
print(f'Training Error (MAE) (Negative Z Features Model) = {neg_error_z:.3f}')
print(f'Training Error (MAE) (Positive CT Features Model) = {pos_error_ct:.3f}')
print(f'Training Error (MAE) (Negative CT Features Model) = {neg_error_ct:.3f}')
```

```
Training Error (MAE) (Positive Z Features Model) = 0.631
Training Error (MAE) (Negative Z Features Model) = 0.666
Training Error (MAE) (Positive CT Features Model) = 0.662
Training Error (MAE) (Negative CT Features Model) = 0.665
```

```
# combine positive/negative edges in one linear regression model
fit_pos_neg_z = linear_model.LinearRegression(fit_intercept=True, normalize=False).
↪fit(np.stack((train_pos_edges_sum_z, train_neg_edges_sum_z)).T, train_phen)
```

```
# combine positive/negative edges in one linear regression model
fit_pos_neg_ct = linear_model.LinearRegression(fit_intercept=True, normalize=False).
↪fit(np.stack((train_pos_edges_sum_ct, train_neg_edges_sum_ct)).T, train_phen)
```

```
pos_neg_error_z = metrics.mean_absolute_error(train_phen, fit_pos_neg_z.predict(np.
↪stack((train_pos_edges_sum_z, train_neg_edges_sum_z)).T))
pos_neg_error_ct = metrics.mean_absolute_error(train_phen, fit_pos_neg_ct.predict(np.
↪stack((train_pos_edges_sum_ct, train_neg_edges_sum_ct)).T))

print(f'Training Error (MAE) (Positive/Negative Z Features Model) = {pos_neg_error_z:.3f}
↪')
print(f'Training Error (MAE) (Positive/Negative CT Features Model) = {pos_neg_error_ct:.
↪3f}')
```

```
Training Error (MAE) (Positive/Negative Z Features Model) = 0.620
Training Error (MAE) (Positive/Negative CT Features Model) = 0.642
```

```
# evaluate out of sample performance
test_pos_edges_sum_z = test_data_z[:, keep_edges_pos_z].sum(1)
test_neg_edges_sum_z = test_data_z[:, keep_edges_neg_z].sum(1)

pos_test_error_z = metrics.mean_absolute_error(test_phen, fit_pos_z.predict(test_pos_
↪edges_sum_z.reshape(-1, 1)))
neg_test_error_z = metrics.mean_absolute_error(test_phen, fit_neg_z.predict(test_neg_
↪edges_sum_z.reshape(-1, 1)))
pos_neg_test_error_z = metrics.mean_absolute_error(test_phen, fit_pos_neg_z.predict(np.
↪stack((test_pos_edges_sum_z, test_neg_edges_sum_z)).T))

test_pos_edges_sum_ct = test_data_ct[:, keep_edges_pos_ct].sum(1)
test_neg_edges_sum_ct = test_data_ct[:, keep_edges_neg_ct].sum(1)

pos_test_error_ct = metrics.mean_absolute_error(test_phen, fit_pos_ct.predict(test_pos_
↪edges_sum_ct.reshape(-1, 1)))
neg_test_error_ct = metrics.mean_absolute_error(test_phen, fit_neg_ct.predict(test_neg_
↪edges_sum_ct.reshape(-1, 1)))
pos_neg_test_error_ct = metrics.mean_absolute_error(test_phen, fit_pos_neg_ct.predict(np.
↪stack((test_pos_edges_sum_ct, test_neg_edges_sum_ct)).T))

print(f'Testing Error (MAE) (Positive Z Features Model) = {pos_test_error_z:.3f}')
print(f'Testing Error (MAE) (Negative Z Features Model) = {neg_test_error_z:.3f}')
print(f'Testing Error (MAE) (Positive/Negative Z Features Model) = {pos_neg_test_error_
↪z:.3f}')
print(f'Testing Error (MAE) (Positive CT Features Model) = {pos_test_error_ct:.3f}')
print(f'Testing Error (MAE) (Negative CT Features Model) = {neg_test_error_ct:.3f}')
print(f'Testing Error (MAE) (Positive/Negative CT Features Model) = {pos_neg_test_error_
↪ct:.3f}')
```

```
Testing Error (MAE) (Positive Z Features Model) = 0.705
Testing Error (MAE) (Negative Z Features Model) = 0.696
Testing Error (MAE) (Positive/Negative Z Features Model) = 0.697
Testing Error (MAE) (Positive CT Features Model) = 0.710
Testing Error (MAE) (Negative CT Features Model) = 0.695
```

(continues on next page)

(continued from previous page)

```
Testing Error (MAE) (Positive/Negative CT Features Model) = 0.701
```

10.8 CPM Cross Validation

```
def cpm(X, y, p_threshold, n_cv_splits, pred_summary_function, verbose=False):
    assert X.shape[0] == y.shape[0]

    fold_accs_train = []
    fold_accs_test = []
    np.random.seed(42)
    shuffled_idxs = np.random.choice(range(X.shape[0]), size=X.shape[0], replace=False)
    for fold_i, test_idxs in enumerate(np.array_split(shuffled_idxs, n_cv_splits)):
        train_mask = np.ones(X.shape[0], np.bool)
        train_mask[test_idxs] = 0

        # create train/text X, y
        train_X, test_X = X[train_mask, :], X[test_idxs, :]
        train_y, test_y = y[train_mask], y[test_idxs]

        # create correlation matrix between train_X and train_y
        if verbose:
            print(f'CV Fold: {fold_i+1:<10} Computing correlations between train_X and␣
↪train_y...')
        with warnings.catch_warnings():
            # we expect pearsonr to throw PearsonRConstantInputWarning because of␣
↪contant valued columns in X
            warnings.simplefilter("ignore")
            train_pheno_corr_p = [stats.pearsonr(train_X[:, i], train_y) for i in␣
↪range(train_X.shape[1])]
            train_corrs = np.array([x[0] for x in train_pheno_corr_p])
            train_pvals = np.array([x[1] for x in train_pheno_corr_p])
            # create masks for edges below p-threshold and split pos/neg correlations
            keep_edges_pos = (train_corrs > 0) & (train_pvals < p_threshold)
            keep_edges_neg = (train_corrs < 0) & (train_pvals < p_threshold)

        # sum X entries with significant correlations with y
        train_pos_edges_sum = train_X[:, keep_edges_pos].sum(1)
        train_neg_edges_sum = train_X[:, keep_edges_neg].sum(1)
        test_pos_edges_sum = test_X[:, keep_edges_pos].sum(1)
        test_neg_edges_sum = test_X[:, keep_edges_neg].sum(1)

        # fit linear regression models based on summed values
        fit_pos = linear_model.LinearRegression(fit_intercept=True, normalize=False).
↪fit(train_pos_edges_sum.reshape(-1, 1), train_y)
        fit_neg = linear_model.LinearRegression(fit_intercept=True, normalize=False).
↪fit(train_neg_edges_sum.reshape(-1, 1), train_y)
        fit_pos_neg = linear_model.LinearRegression(fit_intercept=True, normalize=False).
↪fit(np.stack((train_pos_edges_sum, train_neg_edges_sum)).T, train_y)

        # compute train errors
```

(continues on next page)

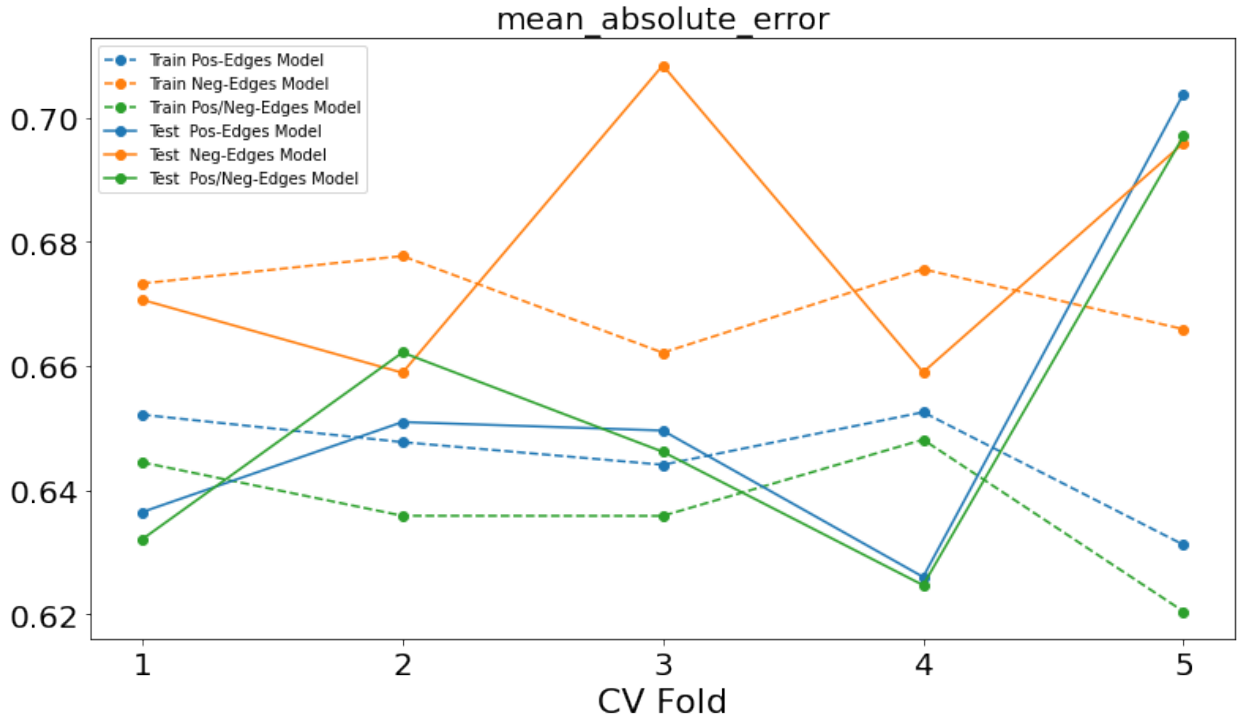
(continued from previous page)

```
        train_pos_error = pred_summary_function(train_y, fit_pos.predict(train_pos_edges_
↪sum.reshape(-1, 1)))
        train_neg_error = pred_summary_function(train_y, fit_neg.predict(train_neg_edges_
↪sum.reshape(-1, 1)))
        train_posneg_error = pred_summary_function(train_y, fit_pos_neg.predict(np.
↪stack((train_pos_edges_sum, train_neg_edges_sum)).T))

        # compute testing errors
        test_pos_error = pred_summary_function(test_y, fit_pos.predict(test_pos_edges_
↪sum.reshape(-1, 1)))
        test_neg_error = pred_summary_function(test_y, fit_neg.predict(test_neg_edges_
↪sum.reshape(-1, 1)))
        test_posneg_error = pred_summary_function(test_y, fit_pos_neg.predict(np.
↪stack((test_pos_edges_sum, test_neg_edges_sum)).T))

        fold_accs_train.append((train_pos_error, train_neg_error, train_posneg_error))
        fold_accs_test.append((test_pos_error, test_neg_error, test_posneg_error))

        if verbose:
            print(f'CV Fold: {fold_i+1:<10} Train Pos-Edges Model MAE: {round(train_pos_
↪error, 3):<10} Train Neg-Edges Model Accuracy: {round(train_neg_error, 3):<10} Train␣
↪Pos/Neg-Edges Model Accuracy: {round(train_posneg_error, 3):<10}')
            print(f'CV Fold: {fold_i+1:<10} Test  Pos-Edges Model MAE: {round(test_pos_
↪error, 3):<10} Test  Neg-Edges Model Accuracy: {round(test_neg_error, 3):<10} Test ␣
↪Pos/Neg-Edges Model Accuracy: {round(test_posneg_error, 3):<10}')


    plt.figure(figsize=(13, 7))
    plt.plot(range(1, len(fold_accs_train)+1), [x[0] for x in fold_accs_train],␣
↪linestyle='--', marker='o', color='C0', label='Train Pos-Edges Model')
    plt.plot(range(1, len(fold_accs_train)+1), [x[1] for x in fold_accs_train],␣
↪linestyle='--', marker='o', color='C1', label='Train Neg-Edges Model')
    plt.plot(range(1, len(fold_accs_train)+1), [x[2] for x in fold_accs_train],␣
↪linestyle='--', marker='o', color='C2', label='Train Pos/Neg-Edges Model')

    plt.plot(range(1, len(fold_accs_test)+1), [x[0] for x in fold_accs_test], linestyle=
↪'-', marker='o', color='C0', label='Test  Pos-Edges Model')
    plt.plot(range(1, len(fold_accs_test)+1), [x[1] for x in fold_accs_test], linestyle=
↪'-', marker='o', color='C1', label='Test  Neg-Edges Model')
    plt.plot(range(1, len(fold_accs_test)+1), [x[2] for x in fold_accs_test], linestyle=
↪'-', marker='o', color='C2', label='Test  Pos/Neg-Edges Model')

    plt.title(pred_summary_function.__name__, fontsize=20)
    plt.xticks(range(1, len(fold_accs_test)+1))
    plt.xlabel('CV Fold')
    plt.legend(fontsize=10)
    plt.show()

    return fold_accs_train, fold_accs_test
```

```
fold_accs_train_z, fold_accs_test_z = cpm(hcp_z, gscores, p_threshold=0.01, n_cv_
↪splits=5, pred_summary_function=metrics.mean_absolute_error, verbose=True)
```

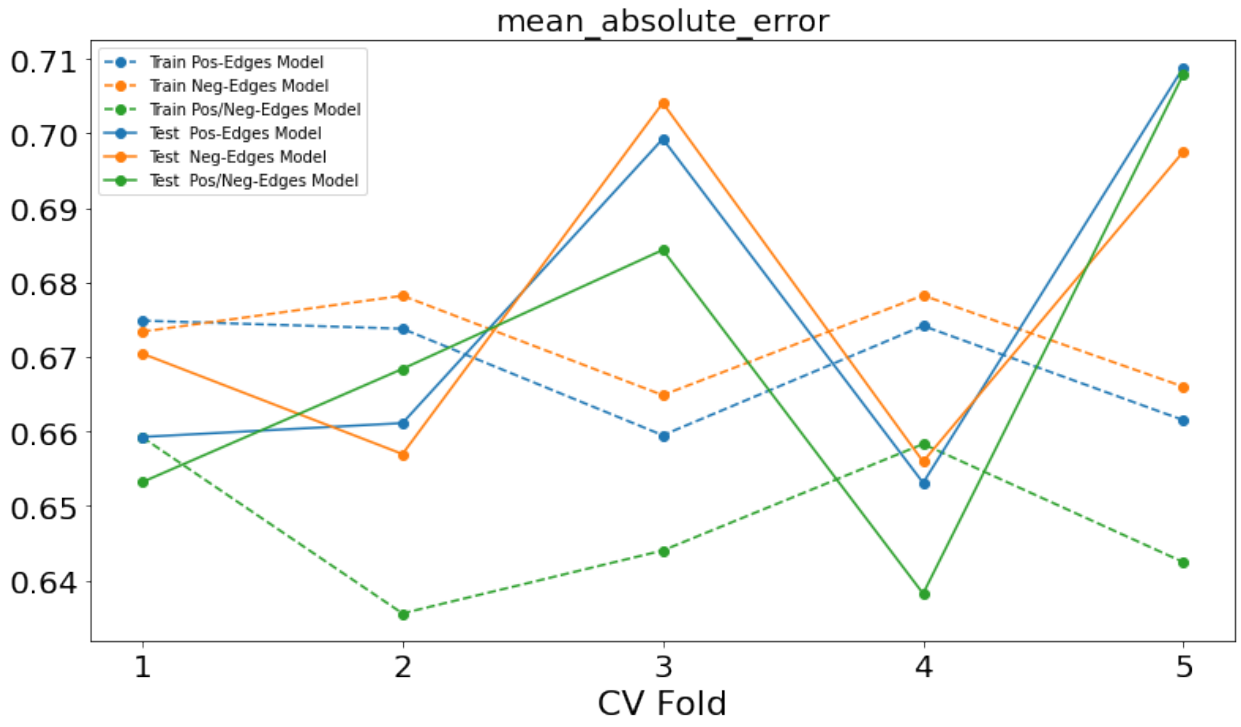
```
CV Fold: 1          Computing correlations between train_X and train_y...
CV Fold: 1          Train Pos-Edges Model MAE: 0.652      Train Neg-Edges Model MAE: 0.
↪673      Train Pos/Neg-Edges Model MAE: 0.644
CV Fold: 1          Test  Pos-Edges Model MAE: 0.636      Test  Neg-Edges Model MAE: 0.
↪671      Test  Pos/Neg-Edges Model MAE: 0.632
CV Fold: 2          Computing correlations between train_X and train_y...
CV Fold: 2          Train Pos-Edges Model MAE: 0.648      Train Neg-Edges Model MAE: 0.
↪678      Train Pos/Neg-Edges Model MAE: 0.636
CV Fold: 2          Test  Pos-Edges Model MAE: 0.651      Test  Neg-Edges Model MAE: 0.
↪659      Test  Pos/Neg-Edges Model MAE: 0.662
CV Fold: 3          Computing correlations between train_X and train_y...
CV Fold: 3          Train Pos-Edges Model MAE: 0.644      Train Neg-Edges Model MAE: 0.
↪662      Train Pos/Neg-Edges Model MAE: 0.636
CV Fold: 3          Test  Pos-Edges Model MAE: 0.65       Test  Neg-Edges Model MAE: 0.
↪708      Test  Pos/Neg-Edges Model MAE: 0.646
CV Fold: 4          Computing correlations between train_X and train_y...
CV Fold: 4          Train Pos-Edges Model MAE: 0.653      Train Neg-Edges Model MAE: 0.
↪676      Train Pos/Neg-Edges Model MAE: 0.648
CV Fold: 4          Test  Pos-Edges Model MAE: 0.626      Test  Neg-Edges Model MAE: 0.
↪659      Test  Pos/Neg-Edges Model MAE: 0.625
CV Fold: 5          Computing correlations between train_X and train_y...
```

```
CV Fold: 5          Train Pos-Edges Model MAE: 0.631      Train Neg-Edges Model MAE: 0.
↪666      Train Pos/Neg-Edges Model MAE: 0.62
CV Fold: 5          Test  Pos-Edges Model MAE: 0.704      Test  Neg-Edges Model MAE: 0.
↪696      Test  Pos/Neg-Edges Model MAE: 0.697
```




```
fold_accs_train_ct, fold_accs_test_ct = cpm(hcp_ct, gscores, p_threshold=0.01, n_cv_
↪splits=5, pred_summary_function=metrics.mean_absolute_error, verbose=True)
```

```
CV Fold: 1          Computing correlations between train_X and train_y...
CV Fold: 1          Train Pos-Edges Model MAE: 0.675      Train Neg-Edges Model MAE: 0.
↪673      Train Pos/Neg-Edges Model MAE: 0.659
CV Fold: 1          Test  Pos-Edges Model MAE: 0.659      Test  Neg-Edges Model MAE: 0.
↪67       Test  Pos/Neg-Edges Model MAE: 0.653
CV Fold: 2          Computing correlations between train_X and train_y...
CV Fold: 2          Train Pos-Edges Model MAE: 0.674      Train Neg-Edges Model MAE: 0.
↪678      Train Pos/Neg-Edges Model MAE: 0.636
CV Fold: 2          Test  Pos-Edges Model MAE: 0.661      Test  Neg-Edges Model MAE: 0.
↪657      Test  Pos/Neg-Edges Model MAE: 0.668
CV Fold: 3          Computing correlations between train_X and train_y...
CV Fold: 3          Train Pos-Edges Model MAE: 0.659      Train Neg-Edges Model MAE: 0.
↪665      Train Pos/Neg-Edges Model MAE: 0.644
CV Fold: 3          Test  Pos-Edges Model MAE: 0.699      Test  Neg-Edges Model MAE: 0.
↪704      Test  Pos/Neg-Edges Model MAE: 0.684
CV Fold: 4          Computing correlations between train_X and train_y...
CV Fold: 4          Train Pos-Edges Model MAE: 0.674      Train Neg-Edges Model MAE: 0.
↪678      Train Pos/Neg-Edges Model MAE: 0.658
CV Fold: 4          Test  Pos-Edges Model MAE: 0.653      Test  Neg-Edges Model MAE: 0.
↪656      Test  Pos/Neg-Edges Model MAE: 0.638
CV Fold: 5          Computing correlations between train_X and train_y...
CV Fold: 5          Train Pos-Edges Model MAE: 0.662      Train Neg-Edges Model MAE: 0.
↪666      Train Pos/Neg-Edges Model MAE: 0.642
CV Fold: 5          Test  Pos-Edges Model MAE: 0.709      Test  Neg-Edges Model MAE: 0.
↪698      Test  Pos/Neg-Edges Model MAE: 0.708
```



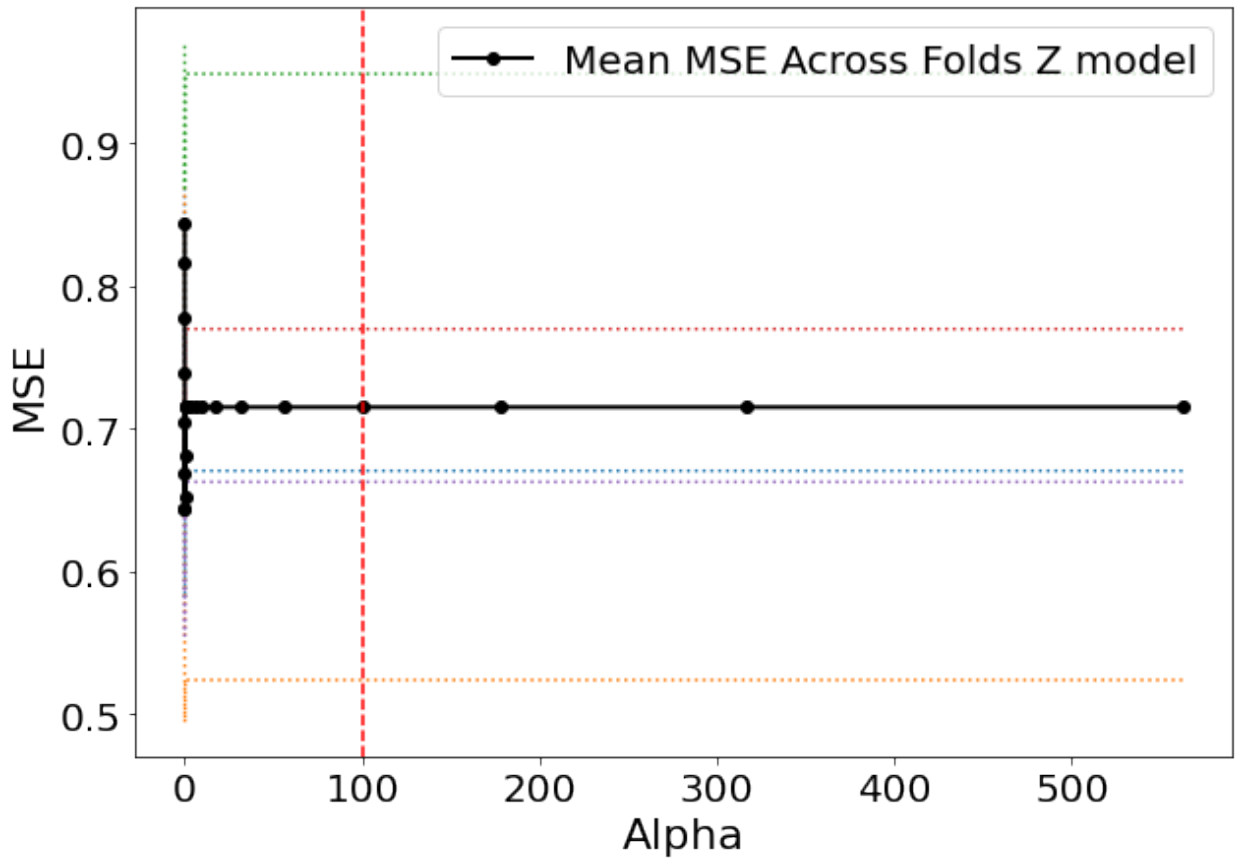
10.9 Lasso (Linear Regression + L1 Regularization)

```
# LassoCV uses coordinate descent to select hyperparameter alpha
alpha_grid = np.array([10**a for a in np.arange(-3, 3, 0.25)])
lassoCV_model_z = linear_model.LassoCV(cv=5, n_alphas=len(alpha_grid), alphas=alpha_grid,
↪ fit_intercept=True, normalize=False, random_state=42, verbose=True, n_jobs=5).
↪fit(train_data_z, train_phen)
```

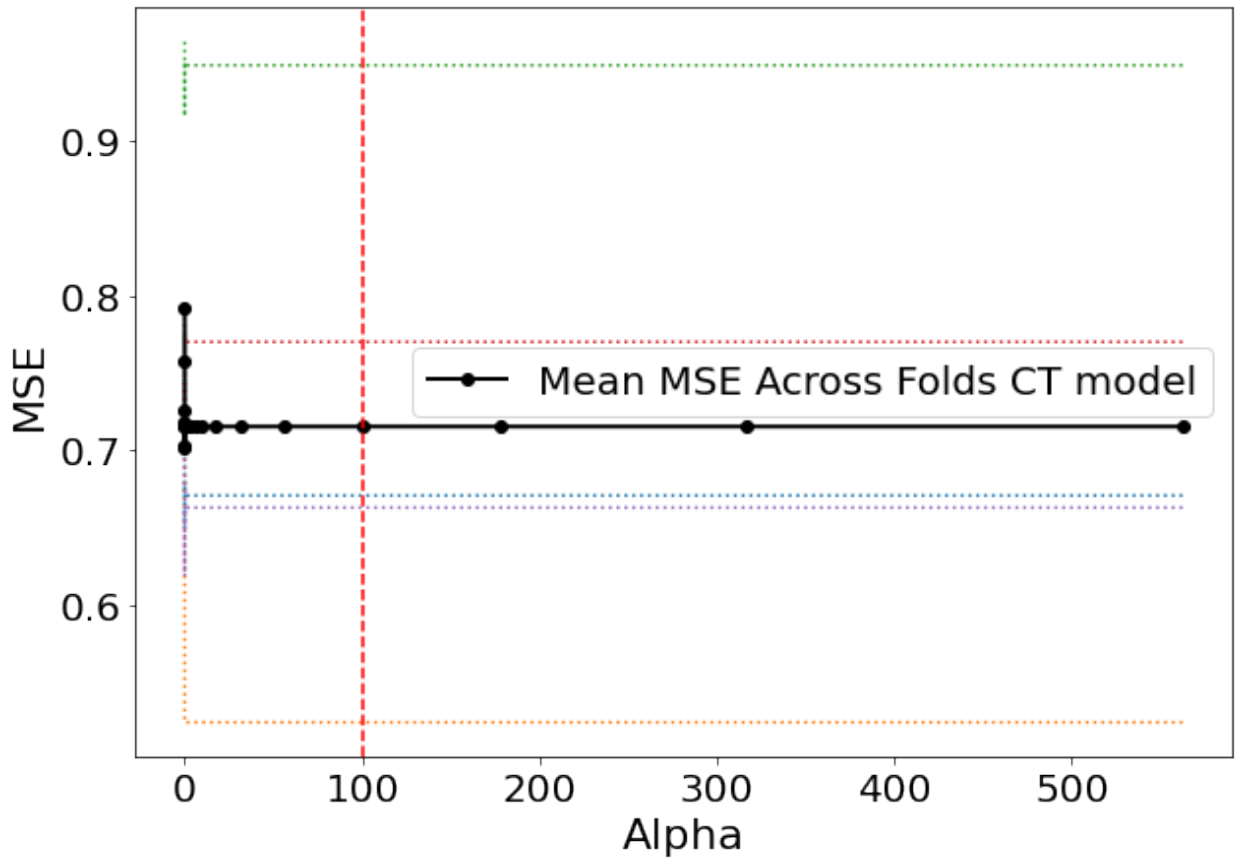
```
# LassoCV uses coordinate descent to select hyperparameter alpha
alpha_grid = np.array([10**a for a in np.arange(-3, 3, 0.25)])
lassoCV_model_ct = linear_model.LassoCV(cv=5, n_alphas=len(alpha_grid), alphas=alpha_
↪grid, fit_intercept=True, normalize=False, random_state=42, verbose=True, n_jobs=5).
↪fit(train_data_ct, train_phen)
```

```
[Parallel(n_jobs=5)]: Using backend ThreadingBackend with 5 concurrent workers.
.....
↪.....[Parallel(n_jobs=5)]: Done   2 out of   5 | elapsed:    0.3s␣
↪remaining:    0.5s
.....[Parallel(n_jobs=5)]: Done   5 out of   5 | elapsed:    0.3s finished
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_base.py:155: FutureWarning:
↪'normalize' was deprecated in version 1.0 and will be removed in 1.2. Please leave the␣
↪normalize parameter to its default value to silence this warning. The default behavior␣
↪of this estimator is to not do any normalization. If normalization is needed please␣
↪use sklearn.preprocessing.StandardScaler instead.
  FutureWarning,
```

```
plt.figure(figsize=(10, 7))
plt.plot(lassoCV_model_z.alphas_, lassoCV_model_z.mse_path_, ':')
plt.plot(lassoCV_model_z.alphas_, lassoCV_model_z.mse_path_.mean(axis=-1), color='k',␣
↪marker='o', label='Mean MSE Across Folds Z model', linewidth=2)
plt.axvline(x=100, linestyle='--', c='r')
plt.xlabel('Alpha')
plt.ylabel('MSE')
plt.legend()
plt.show()
```




```
plt.figure(figsize=(10, 7))
plt.plot(lassoCV_model_ct.alphas_, lassoCV_model_ct.mse_path_, ':')
plt.plot(lassoCV_model_ct.alphas_, lassoCV_model_ct.mse_path_.mean(axis=-1), color='k',␣
↪marker='o', label='Mean MSE Across Folds CT model', linewidth=2)
plt.axvline(x=100, linestyle='--', c='r')
plt.xlabel('Alpha')
plt.ylabel('MSE')
plt.legend()
plt.show()
```




```
# based on cv results above, set alpha=100
lasso_model_z = linear_model.Lasso(alpha=lassoCV_model_z.alpha_, fit_intercept=True,␣
↪normalize=False).fit(train_data_z, train_phen)
```

```
# based on cv results above, set alpha=100
lasso_model_ct = linear_model.Lasso(alpha=lassoCV_model_ct.alpha_, fit_intercept=True,␣
↪normalize=False).fit(train_data_ct, train_phen)
```

```
train_preds_lasso_model_z = lasso_model_z.predict(train_data_z)
test_preds_lasso_model_z = lasso_model_z.predict(test_data_z)

train_mae_z = metrics.mean_absolute_error(train_phen, train_preds_lasso_model_z)
test_mae_z = metrics.mean_absolute_error(test_phen, test_preds_lasso_model_z)

train_preds_lasso_model_ct = lasso_model_ct.predict(train_data_ct)
test_preds_lasso_model_ct = lasso_model_ct.predict(test_data_ct)

train_mae_ct = metrics.mean_absolute_error(train_phen, train_preds_lasso_model_ct)
test_mae_ct = metrics.mean_absolute_error(test_phen, test_preds_lasso_model_ct)

print(f'Train MAE Z model: {train_mae_z:.3f}')
print(f'Test MAE Z model: {test_mae_z:.3f}')
print(f'Train MAE CT model: {train_mae_ct:.3f}')
print(f'Test MAE CT model: {test_mae_ct:.3f}')
```

```
Train MAE Z model: 0.620
Test MAE Z model: 0.682
Train MAE CT model: 0.650
Test MAE CT model: 0.697
```

10.10 Ridge (Linear Regression + L2 Regularization)

```
# RidgeCV uses generalized cross validation to select hyperparameter alpha
with warnings.catch_warnings():
    # ignore matrix decomposition errors
    warnings.simplefilter("ignore")
    ridgeCV_model_z = linear_model.RidgeCV(alphas=(0.1, 1.0, 10.0), fit_intercept=True,␣
↪normalize=False, cv=5).fit(train_data_z, train_phen)
```

```
# RidgeCV uses generalized cross validation to select hyperparameter alpha
with warnings.catch_warnings():
    # ignore matrix decomposition errors
    warnings.simplefilter("ignore")
    ridgeCV_model_ct = linear_model.RidgeCV(alphas=(0.1, 1.0, 10.0), fit_intercept=True,␣
↪normalize=False, cv=5).fit(train_data_ct, train_phen)
```

```
ridge_alpha_z = ridgeCV_model_z.alpha_
print(f'CV Selected Alpha Z model = {ridge_alpha_z:.3f}')
```

```
CV Selected Alpha Z model = 10.000
```

```
ridge_alpha_ct = ridgeCV_model_ct.alpha_
print(f'CV Selected Alpha CT model = {ridge_alpha_ct:.3f}')
```

```
CV Selected Alpha CT model = 10.000
```

```
ridge_model_z = linear_model.Ridge(alpha=ridge_alpha_z, fit_intercept=True,␣
↪normalize=False).fit(train_data_z, train_phen)
```

```
ridge_model_ct = linear_model.Ridge(alpha=ridge_alpha_ct, fit_intercept=True,␣
↪normalize=False).fit(train_data_ct, train_phen)
```

```
train_preds_ridge_model_z = ridge_model_z.predict(train_data_z)
test_preds_ridge_model_z = ridge_model_z.predict(test_data_z)

train_mae_z = metrics.mean_absolute_error(train_phen, train_preds_ridge_model_z)
test_mae_z = metrics.mean_absolute_error(test_phen, test_preds_ridge_model_z)

train_preds_ridge_model_ct = ridge_model_ct.predict(train_data_ct)
test_preds_ridge_model_ct = ridge_model_ct.predict(test_data_ct)

train_mae_ct = metrics.mean_absolute_error(train_phen, train_preds_ridge_model_ct)
test_mae_ct = metrics.mean_absolute_error(test_phen, test_preds_ridge_model_ct)
```

(continues on next page)

(continued from previous page)

```
print(f'Train MAE Z model: {train_mae_z:.3f}')
print(f'Test MAE Z model: {test_mae_z:.3f}')
print(f'Train MAE CT model: {train_mae_ct:.3f}')
print(f'Test MAE CT model: {test_mae_ct:.3f}')
```

```
Train MAE Z model: 0.527
Test MAE Z model: 0.734
Train MAE CT model: 0.600
Test MAE CT model: 0.692
```

10.11 Elastic Net (Linear Regression + L1/L2 Regularization)

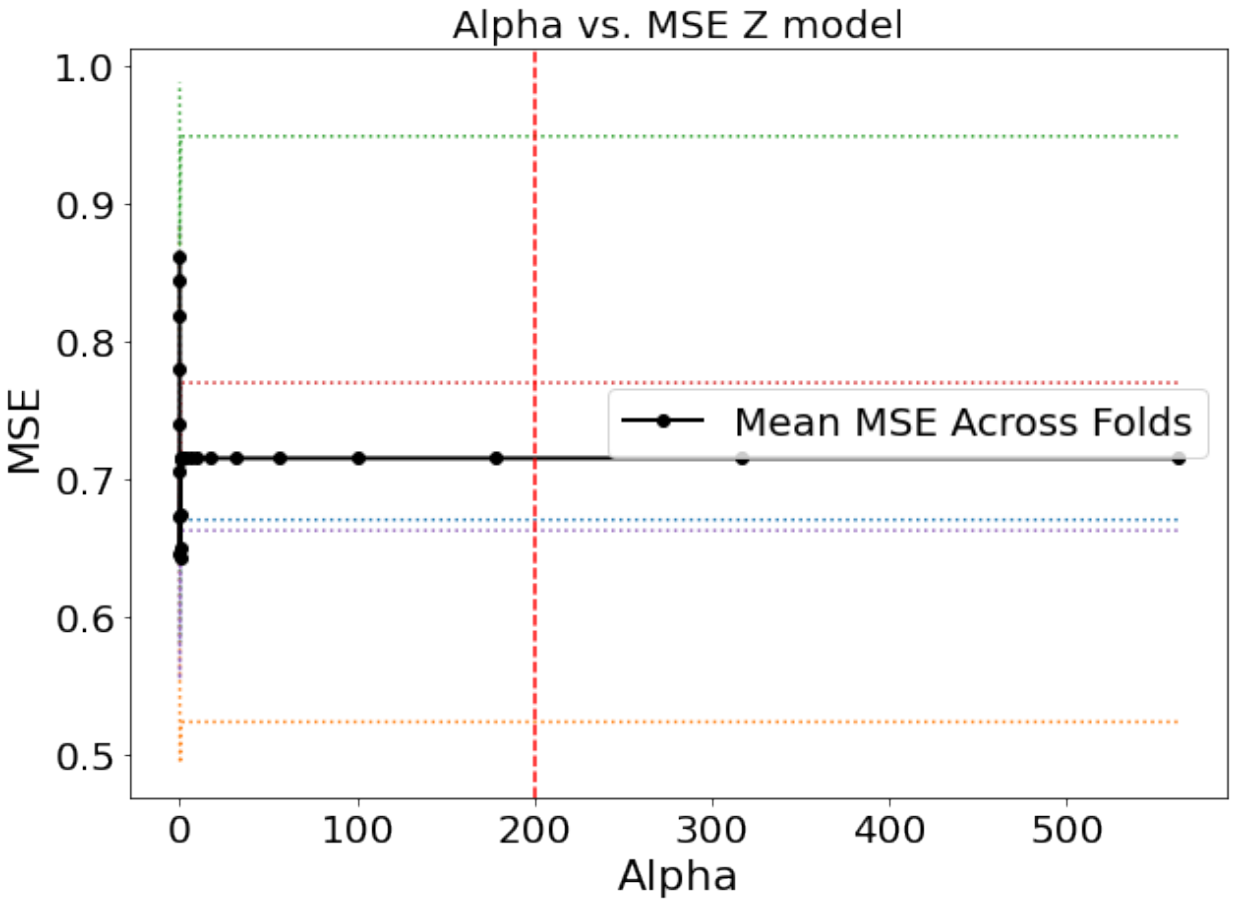
```
# RidgeCV uses generalized cross validation to select hyperparameter alpha
elasticnetCV_model_z = linear_model.ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, 1],␣
↪cv=5, n_alphas=len(alpha_grid), alphas=alpha_grid, random_state=42, verbose=True, n_
↪jobs=5).fit(train_data_z, train_phen)
```

```
# RidgeCV uses generalized cross validation to select hyperparameter alpha
elasticnetCV_model_ct = linear_model.ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, 1],
↪ cv=5, n_alphas=len(alpha_grid), alphas=alpha_grid, random_state=42, verbose=True, n_
↪jobs=5).fit(train_data_ct, train_phen)
```

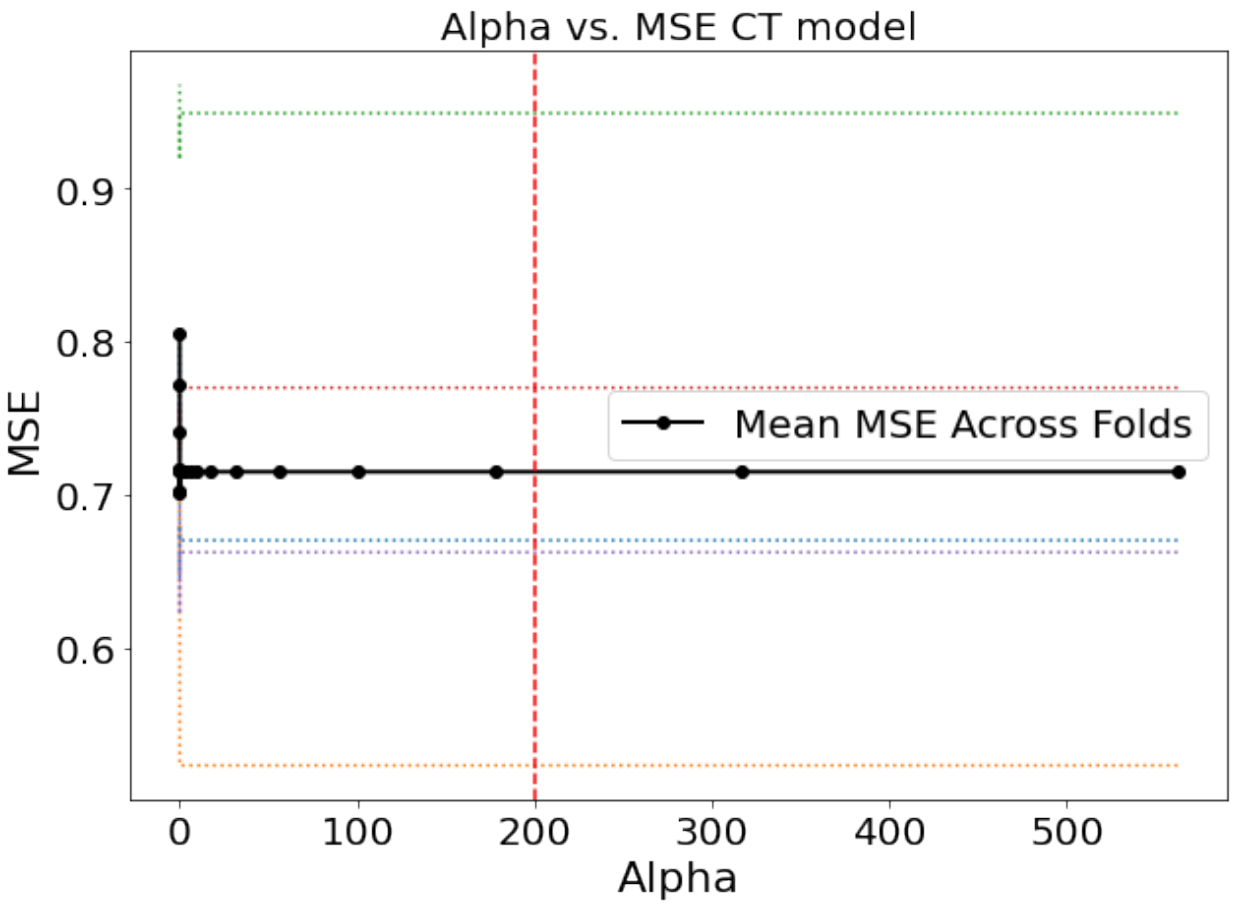
```
print(f'CV selected alpha Z model {elasticnetCV_model_z.alpha_:.3f}')
print(f'Elastic net L1 ratio Z model {elasticnetCV_model_z.l1_ratio_:.3f}')
print(f'CV selected alpha CT model {elasticnetCV_model_ct.alpha_:.3f}')
print(f'Elastic net L1 ratio CT model {elasticnetCV_model_ct.l1_ratio_:.3f}')
```

```
CV selected alpha Z model 0.056
Elastic net L1 ratio Z model 0.700
CV selected alpha CT model 0.032
Elastic net L1 ratio CT model 0.100
```

```
plt.figure(figsize=(10, 7))
plt.plot(elasticnetCV_model_z.alphas_, elasticnetCV_model_z.mse_path_[1, :, :], ':')
plt.plot(elasticnetCV_model_z.alphas_, elasticnetCV_model_z.mse_path_[1, :, :].
↪mean(axis=-1), color='k', marker='o', label='Mean MSE Across Folds', linewidth=2)
plt.axvline(x=200, linestyle='--', c='r')
plt.title('Alpha vs. MSE Z model')
plt.xlabel('Alpha')
plt.ylabel('MSE')
plt.legend()
plt.show()
```




```
plt.figure(figsize=(10, 7))
plt.plot(elasticnetCV_model_ct.alphas_, elasticnetCV_model_ct.mse_path_[1, :, :], ':')
plt.plot(elasticnetCV_model_ct.alphas_, elasticnetCV_model_ct.mse_path_[1, :, :].
↪mean(axis=-1), color='k', marker='o', label='Mean MSE Across Folds', linewidth=2)
plt.axvline(x=200, linestyle='--', c='r')
plt.title('Alpha vs. MSE CT model')
plt.xlabel('Alpha')
plt.ylabel('MSE')
plt.legend()
plt.show()
```




```
elasticnet_model_z = linear_model.ElasticNet(alpha=elasticnetCV_model_z.alpha_, l1_
↪ratio=elasticnetCV_model_z.l1_ratio_, fit_intercept=True, normalize=False, random_
↪state=42).fit(train_data_z, train_phen)

train_preds_en_model_z = elasticnet_model_z.predict(train_data_z)
test_preds_en_model_z = elasticnet_model_z.predict(test_data_z)

train_mae_z = metrics.mean_absolute_error(train_phen, train_preds_en_model_z)
test_mae_z = metrics.mean_absolute_error(test_phen, test_preds_en_model_z)

elasticnet_model_ct = linear_model.ElasticNet(alpha=elasticnetCV_model_ct.alpha_, l1_
↪ratio=elasticnetCV_model_ct.l1_ratio_, fit_intercept=True, normalize=False, random_
↪state=42).fit(train_data_ct, train_phen)

train_preds_en_model_ct = elasticnet_model_ct.predict(train_data_ct)
test_preds_en_model_ct = elasticnet_model_ct.predict(test_data_ct)

train_mae_ct = metrics.mean_absolute_error(train_phen, train_preds_en_model_ct)
test_mae_ct = metrics.mean_absolute_error(test_phen, test_preds_en_model_ct)

print(f'Train MAE Z model: {train_mae_z:.3f}')
print(f'Test MAE Z model: {test_mae_z:.3f}')
print(f'Train MAE CT model: {train_mae_ct:.3f}')
print(f'Test MAE CT model: {test_mae_ct:.3f}')
```

```
Train MAE Z model: 0.611
Test MAE Z model: 0.680
Train MAE CT model: 0.633
Test MAE CT model: 0.692
```

CHAPTER

ELEVEN

FREQUENTLY ASKED QUESTIONS

Most of the questions we recieve are about interpretation of normative modeling outputs.

The PCNtoolkit develoers have written a protocol for how to run a normative modeling analysis which should be helpful to you if you are just getting started.

Rutherford, S., Kia, S. M., Wolfers, T., … Beckmann, C. F., & Marquand, A. F. (2022). The Normative Modeling Framework for Computational Psychiatry. Nature Protocols. https://www.nature.com/articles/s41596-022-00696-5.

Feel free to interact with the normative modeling community and browse the existing questions/answers on Gitter

CHAPTER

TWELVE

GLOSSARY

12.1 Key abbreviations

Abbreviation	Full name
BLR	Bayesian Linear Regression
GPR	Gaussian Process Regression
HBR	Hierarchical Bayesian Regression
NM	Normative Model
EV / EXPV	Explained Variance
MSLL	Mean Standardized Log Loss
SMSE	Standardized Mean Squared Error
RMSE	Root Mean Squared Error between true/predicted responses
Rho	Pearson orrelation between true/predicted responses
pRho	Parametric p-value for this correlation
Z	Z-score or deviation score
yhat	predictive mean
ys2	predictive variance

CHAPTER

THIRTEEN

HOW TO CITE PCNTOOLKIT

If you use the PCNtoolkit, please consider citing some of the following work:

Marquand, A. F., Wolfers, T., Mennes, M., Buitelaar, J., & Beckmann, C. F. (2016). Beyond Lumping and Splitting: A Review of Computational Approaches for Stratifying Psychiatric Disorders. Biological Psychiatry: Cognitive Neuroscience and Neuroimaging. https://doi.org/10.1016/j.bpsc.2016.04.002

Marquand, A. F., Rezek, I., Buitelaar, J., & Beckmann, C. F. (2016). Understanding Heterogeneity in Clinical Cohorts Using Normative Models: Beyond Case-Control Studies. Biological Psychiatry. https://doi.org/10.1016/j.biopsych.2015.12.023

Marquand, A. F., Kia, S. M., Zabihi, M., Wolfers, T., Buitelaar, J. K., & Beckmann, C. F. (2019). Conceptualizing mental disorders as deviations from normative functioning. Molecular Psychiatry. https://doi.org/10.1038/s41380-019-0441-1

Marquand, A. F., Haak, K. V., & Beckmann, C. F. (2017). Functional corticostriatal connection topographies predict goal directed behaviour in humans. Nature Human Behaviour. https://doi.org/10.1038/s41562-017-0146

Wolfers, T., Beckmann, C. F., Hoogman, M., Buitelaar, J. K., Franke, B., & Marquand, A. F. (2020). Individual differences v. the average patient: Mapping the heterogeneity in ADHD using normative models. Psychological Medicine. https://doi.org/10.1017/S0033291719000084

Wolfers, T., Rokicki, J., Alnæs, D., Berthet, P., Agartz, I., Kia, S. M., Kaufmann, T., Zabihi, M., Moberget, T., Melle, I., Beckmann, C. F., Andreassen, O. A., Marquand, A. F., & Westlye, L. T. (n.d.). Replicating extensive brain structural heterogeneity in individuals with schizophrenia and bipolar disorder. Human Brain Mapping. https://doi.org/10.1002/hbm.25386

Zabihi, M., Floris, D. L., Kia, S. M., Wolfers, T., Tillmann, J., Arenas, A. L., Moessnang, C., Banaschewski, T., Holt, R., Baron-Cohen, S., Loth, E., Charman, T., Bourgeron, T., Murphy, D., Ecker, C., Buitelaar, J. K., Beckmann, C. F., & Marquand, A. (2020). Fractionating autism based on neuroanatomical normative modeling. Translational Psychiatry. https://doi.org/10.1038/s41398-020-01057-0

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, J. K., Beckmann, C. F., & Marquand, A. F. (2019). Dissecting the Heterogeneous Cortical Anatomy of Autism Spectrum Disorder Using Normative Models. Biological Psychiatry: Cognitive Neuroscience and Neuroimaging. https://doi.org/10.1016/j.bpsc.2018.11.013

Kia, S. M., & Marquand, A. (2018). Normative Modeling of Neuroimaging Data using Scalable Multi-Task Gaussian Processes. ArXiv. https://arxiv.org/abs/1806.01047

Kia, S. M., Beckmann, C. F., & Marquand, A. F. (2018). Scalable Multi-Task Gaussian Process Tensor Regression for Normative Modeling of Structured Variation in Neuroimaging Data. ArXiv. https://arxiv.org/abs/1808.00036

Kia, S. M., Huijsdens, H., Dinga, R., Wolfers, T., Mennes, M., Andreassen, O. A., Westlye, L. T., Beckmann, C. F., & Marquand, A. F. (2020). Hierarchical Bayesian Regression for Multi-site Normative Modeling of Neuroimaging Data.

In A. L. Martel, P. Abolmaesumi, D. Stoyanov, D. Mateus, M. A. Zuluaga, S. K. Zhou, D. Racoceanu, & L. Joskowicz (Eds.), Medical Image Computing and Computer Assisted Intervention – MICCAI 2020. Springer International Publishing. https://doi.org/10.1007/978-3-030-59728-3_68

Huertas, I., Oldehinkel, M., van Oort, E. S. B., Garcia-Solis, D., Mir, P., Beckmann, C. F., & Marquand, A. F. (2017). A Bayesian spatial model for neuroimaging data based on biologically informed basis functions. NeuroImage. https://doi.org/10.1016/j.neuroimage.2017.08.009

Fraza, C. J., Dinga, R., Beckmann, C. F., & Marquand, A. F. (2021). Warped Bayesian Linear Regression for Normative Modelling of Big Data. NeuroImage, https://doi.org/10.1016/j.neuroimage.2021.118715.

Rutherford, S., Kia, S. M., Wolfers, T., … Beckmann, C. F., & Marquand, A. F. (2022). The Normative Modeling Framework for Computational Psychiatry. Nature Protocols. https://www.nature.com/articles/s41596-022-00696-5.

Rutherford, S., Fraza, C., … Beckmann, C. F., & Marquand, A. F. (2022). Charting Brain Growth and Aging at High Spatial Precision. eLife. https://elifesciences.org/articles/72904

de Boer, A., Kia, S., … & Marquand A. F. (2022). Non-Gaussian Normative Modelling With Hierarchical Bayesian Regression. bioRxiv. https://www.biorxiv.org/content/10.1101/2022.10.05.510988v1

Fraza, C., Zabihi, M., Beckmann, C. F., & Marquand, A. F. (2022). The Extremes of Normative Modelling. bioRxiv. https://www.biorxiv.org/content/10.1101/2022.08.23.505049v1

CHAPTER

FOURTEEN

ACKNOWLEDGEMENTS

We gratefully acknowledge funding from the Dutch Organisation for Scientific Research (NWO), via a Vernieuwingsimpuls VIDI fellowship, from the UK Wellcome Trust via a Digital Innovator grant and from the UK Medical Research Council via an Experimental Medicine Challenge Grant.

Core developers of the toolbox are:

- Andre Marquand
- Seyed Mostafa Kia
- Thomas Wolfers
- Saige Rutherford
- Richard Dinga
- Mariam Zabihi
- Charlotte Fraza
- Augustijn de Boer
- Pieter Barkema

PYTHON MODULE INDEX

INDEX

N

P

Q

R

S

T

U

V

W