DCDF

Feb 24, 2022

Contents:

CHAPTER 1

User Guide

1.1 Installation and Requirements

This package can be installed through the Python Package Index (PyPI) using the command:

```
python3 -m pip install dcdf
```

CHAPTER 2

Code Documentation

2.1 CLI Module

Contains functionality for commandline interface as well as argument validation.

dcdf.cli.**_build_parser**() → argparse.ArgumentParser
Build the parser for commandline arguments.

Returns returns an argument parser for the CLI

dcdf.cli.**_check_nifti**(*file_list: List[str], from_file: Optional[bool] = False*) → bool
Check whether each of the nifti files can be found.

Parameters

- **file_list** – Should be one of: args.build, args.evaluate, arg.reference_masks, args.evaluation_masks, args.group_mask. Where args are the parsed arguments.
- **from_file** – Whether the arguments have been supplied as text files.

Returns True if each of the nifti images can be found on disk.

dcdf.cli.**_get_bounds_filter**(*args*)
If lower/upper bounds have been specified by the arguments, then provide a filter to be applied to the data.

Parameters **args** – The parsed arguments to this program.

dcdf.cli.**_get_list**(*arg: List[str], from_file: bool*) → List[str]
Helper function to handle the from_file = True/False usecases.

Parameters

- **arg** – either a list of nifti filenames, or a list with a single entry to a textfile (of filenames)
- **from_file** – if False, arg is a list of filenames, if true, it is a list with a single entry to a textfile

Returns List of filenames

dcdf.cli.**_lc**(*filename: str*) → int
Helper function to count the number of lines in a file.

Parameters **filename** – name of the file to be read – assumed to be plain text.

Returns count of the number of lines in file

dcdf.cli.**_validate_args**(*parser: argparse.ArgumentParser*) → bool
Sanity checking on the inputs. Returns False if any checks fail.

Parameters **parser** – the parser returned from *_build_parser*

Returns False if any of the arguments fail the sanity checks, else True.

dcdf.cli.**main**()
This function is called on startup, and consists of the following steps: 1. Construct our parser, validate commandline arguments, and retrieve arguments. 2. Construct a filter based on any specified bounds. 3. Build reference as specified. 4. Write reference (if requested). 5. Evaluate samples (if requested) – run in parallel if requested. 6. Print results.

2.2 Data Module

Contains functions used to load and create the datastructures used in this project.

class dcdf.data.**ModifiedECDF**(*ecdf: scipy.stats.stats.CumfreqResult*)
Bases: object

__dict__ = mappingproxy({'__module__': 'dcdf.data', '__init__': <function ModifiedECD

__init__(*ecdf: scipy.stats.stats.CumfreqResult*)
Slightly hack-ish class that was added to add support for inverse functions without changing too much of the code.

Parameters **ecdf** – the output of *scipy.stats.cumfreq* which is being modified.

__module__ = 'dcdf.data'

__weakref__
list of weak references to the object (if defined)

dcdf.data.**get_datapoints**(*input_filename: str, mask_filename: Optional[str] = None, mask_indices: Optional[numpy.ndarray] = None, ignore_zeros: Optional[bool] = True, filter: Optional[Callable[[numpy.ndarray], numpy.ndarray]] = None*) → numpy.ndarray
This function reads a nifti file and returns a flat (1D) array of the image. Various options can be used to filter the array

Parameters

- **input_filename** – filename of the nifti file to be loaded
- **mask_filename** – Optional: filename of mask to be applied to data
- **mask_indices** – Optional and ignored if mask_filename is set. Indices to extract from data array
- **filter** – Optional: function which takes in an np.ndarray and returns an np.ndarrray. Can be used to apply a filter to the data (e.g thresholding)

Returns A 1-D number array containing the filtered datapoints

dcdf.data.**get_null_reference_cdf**(*lowerlimit: numpy.float32*, *upperlimit: numpy.float32*, *numbins: int = 1000*) → dcdf.data.ModifiedECDF

This function will return a CDF to be used as a null reference.

Parameters

- **lowerlimit** – lower bound for the CDF
- **upperlimit** – upperbound for the CDF
- **numbins** – How many bins should be used for the reference

Returns ModifiedECDF of all zeros for the specified range

dcdf.data.**get_percentiles**(*data: numpy.ndarray*, *nsamples: int*) → numpy.ndarray

Sample the data at various percentiles.

Parameters

- **data** – the full data to be considered
- **nsamples** – the number of percentiles to be evaluated (e.g nsamples=100 will calulate the 1st, 2nd, . . . , 99th, 100th percentile)

Returns a numpy array holding the data values which correspond

the index's percentile

dcdf.data.**get_reference_cdf**(*reference_list: List[str], numbins: Optional[int] = 1000, indv_mask_list: Optional[List[str]] = None, group_mask_filename: Optional[str] = None, filter: Optional[Callable[[numpy.ndarray], numpy.ndarray]] = None, lowerlimit: Optional[numpy.float32] = None, upperlimit: Optional[numpy.float32] = None, _piecewise: Optional[bool] = True*) → dcdf.data.ModifiedECDF

This function will return a CDF to be used as a reference based on the provided images.

Parameters

- **reference_list** – List of nifti files to be used for reference.
- **numbins** – How many bins should be used for the reference
- **indv_mask_list** – A list with the same length as *reference_list* of masks to be used for each subject.
- **group_mask_filename** – If not None, this should be a path to a nifti file which should be used as a mask for each of the reference images. If set, *indv_mask_list* will be ignored.
- **filter** – A filtering function to be applied to the flattened array of nifti data.
- **lowerlimit** – lower bound for the CDF
- **upperlimit** – upperbound for the CDF
- **_piecewise** – memory efficient loading – requires lowerlimit and upper limit to be set behaviour is undefined otherwise.

Returns A data structure containing information about the cumulative distribution of the data

dcdf.data.**get_subject_cdf**(*subject_array: numpy.ndarray*, *reference_cdf: dcdf.data.ModifiedECDF*) → dcdf.data.ModifiedECDF

Calculate the individual subject's cdf with respect to the reference CDF.

Parameters

- **subject_array** – numpy array of datapoints from *get_datapoints*

- **reference_cdf** – reference_cdf that was built using *get_reference_cdf*.

Returns ECDF information for the requested subject.

dcdf.data.**get_subject_cdf2**(*subject_array: numpy.ndarray*, *numbins: int*, *lowerlimit: numpy.float32*, *binsize: int*) → dcdf.data.ModifiedECDF

Calculate the individual subject's cdf with respect to the reference CDF.

Parameters

- **subject_array** – numpy array of datapoints from *get_datapoints*
- **numbins** – len(Cumfreqresult.cumcount)
- **lowerlimit** – CumfreqResult.lowerlimit
- **binsize** – Cumfreqresult.binsize

Returns ECDF information for the requested subject.

dcdf.data.**load_reference**(*filename*) → dcdf.data.ModifiedECDF

Load and retrun a pickled reference. Note: this function will assume that the pickled object is in fact of type ModifiedECDF. No checks will be performed ...

Parameters **filename** – path to the pickled ModifiedECDF which should be loaded.

Returns the pickled reference

dcdf.data.**save_reference**(*reference: dcdf.data.ModifiedECDF*, *filename: str*)

Save the reference using pickle. If available, protocol 4 will be used.

Parameters

- **reference** – ModifiedECDF to be saved.
- **filename** – path to save the reference to.

2.3 Measure Module

Contains functions used to perform single-threaded evaluation of subjects.

dcdf.measure.**get_func_dict**(*func_file: str*) → dict

Reads a textfile which should be in the format: [function name] : [function code] where [function code] will later be executed using *eval*

Parameters **func_file** – name of the textfile containing the equations

Returns a dictionary of [function name] and [function code] pairs

dcdf.measure.**measure_single_subject**(*subject: str, reference: scipy.stats.stats.CumfreqResult, func_dict: Dict[str, Callable[[numpy.ndarray, numpy.ndarray, numpy.float32], numpy.float32]], indv_mask: Optional[str] = None, group_mask_indices: Optional[numpy.ndarray] = None, filter: Optional[Callable[[numpy.ndarray], numpy.ndarray]] = None, _print_inverse: Optional[bool] = False*) → Tuple[str, Dict[str, numpy.float32]]

Function to apply provided measures to a single subject

Parameters

- **subjects** – Nifti file paths
- **reference** – CumfreqResult from *data.get_reference_cdf*

- **func_dict** – Output of *measure.get_func_dict*. A dictionary of functions to be calculated over CDF differences. Keys will be used as column names in the return of this function
- **indv_mask** – Mask to be applied to subject image
- **group_mask_filename** – If not None, this should be a path to a nifti file which will be used as a mask for eac of the individual images. If set, *indv_mask_list* will be ignored.
- **filter** – Optional: function which takes in an np.ndarray and returns an np.ndarray. Can be used to apply a filter to the data (e.g thresholding)

Returns a Tuple with first element being the nifti file path, and the second element being a dictionary of results with keys being function names

dcdf.measure.**measure_subjects**(*subjects_list: List[str], reference: scipy.stats.stats.CumfreqResult, func_dict: Dict[str, Callable[[numpy.ndarray, numpy.ndarray, numpy.float32], numpy.float32]], indv_mask_list: Optional[List[str]] = None, group_mask_filename: Optional[str] = None, filter: Optional[Callable[[numpy.ndarray], numpy.ndarray]] = None*) → pandas.core.frame.DataFrame

Wrapper around measure_single_subject to apply to each subject

Parameters

- **subjects_list** – List of nifti file paths
- **reference** – CumfreqResult from *data.get_reference_cdf*
- **func_dict** – Output of *measure.get_func_dict*. A dictionary of functions to be calculated over CDF differences. Keys will be used as column names in the return of this function
- **indv_mask_list** – A list with the same length as *subjects_list* to be used for each subject.
- **group_mask_filename** – If not None, this should be a path to anifti file which will be used as a mask for eac of the individual images. If set, *indv_mask_list* will be ignored.
- **filter** – Optional: function which takes in an np.ndarray and returns an np.ndarray. Can be used to apply a filter to the data (e.g thresholding)

Returns pandas.DataFrame containing all of the results.

dcdf.measure.**print_measurements**(*mdf: pandas.core.frame.DataFrame*)

This function will print out the results of *measure.measure_subjects*.

Parameters **mdf** – pd.DataFrame returned from *measure.measure_subjects*

2.4 Parallel Module

As per *measure* module, but allows for parallel evaluation of subjects.

dcdf.parallel.**_mp_measure**(*subject: str, indv_mask_filename: Optional[str] = None*) → List[numpy.float32]

Function to apply provided measures to a single subject

Parameters

- **subjects** – Nifti file paths
- **indv_mask_filename** – Mask to be applied to subject image

dcdf.parallel.**_worker_init**(*binsize: numpy.float32, inverse_binsize: numpy.float32, lowerlimit: numpy.float32, func_dict: Dict[str, Callable[[numpy.ndarray, numpy.ndarray, numpy.float32], numpy.float32]], shared_mask: Tuple[str, Tuple[int], numpy.dtype], shared_ref: Tuple[str, Tuple[int], numpy.dtype], shared_ref_inverse: Tuple[str, Tuple[int], numpy.dtype], filter: Optional[Callable[[numpy.ndarray], numpy.ndarray]] = None*) → None

Initialization function for parallel workers calling *_mp_measure*. :param binsize: *CumfreqResult.binsize* :param lowerlimit: *CumfreqResult.lowerlimit*. :param func_dict: Output of *measure.get_func_dict*. A dictionary :param shm_ref_tuple: (shm.name,shape,dtype) :param shm_mask_tuple: (shm.name,shape,dtype) :param filter: Optional: function which takes in an np.ndarray and

dcdf.parallel.**parallel_measure_subjects**(*subjects_list: List[str], reference: scipy.stats.stats.CumfreqResult, func_dict: Dict[str, Callable[[numpy.ndarray, numpy.ndarray, numpy.float32], numpy.float32]], indv_mask_list: Optional[List[str]] = None, group_mask_filename: Optional[str] = None, filter: Optional[Callable[[numpy.ndarray], numpy.ndarray]] = None, n_procs: Optional[int] = None*) → pandas.core.frame.DataFrame

Parameters

- **subjects_list** – List of nifti file paths
- **reference** – CumfreqResult from *data.get_reference_cdf*
- **func_dict** – Output of *measure.get_func_dict*. A dictionary of functions to be calculated over CDF differences. Keys will be used as column names in the return of this function
- **indv_mask_list** – A list with the same length as *subjects_list* to be used for each subject.
- **group_mask_filename** – If not None, this should be a path to anifti file which will be used as a mask for eac of the individual images. If set, *indv_mask_list* will be ignored.
- **filter** – Optional: function which takes in an np.ndarray and returns an np.ndarray. Can be used to apply a filter to the data (e.g thresholding)
- **n_procs** – Number of processes to be started. If none, then the number returned by *os.cpu_count()* is used

CHAPTER 3

General Framework

In neuroimaging, summary statistics are frequently used in an attempt to describe various aspects of a patient's neuroanatomy. For example, one might measure the mean FA intensity across some white matter region of interest in order to quantify structural integrity. Similarly, PSMD, defined as the width between the 5th and 95th percentile in a skeletonized mean diffusivity map, has recently been used to quantify vascular burden in an SVD cohort. Such statistics are generally intended to maximally describe the underlying data generating process. Unless a statistic is sufficient, however, there is no guarantee that it will be able to capture information which correlates well with independent variables of interest. Furthermore, non-trivial sufficient statistics require the underlying distribution to be known in parametric form, and, even then, the theoretical derivations can be extremely complicated.

To date, no group has parameterized the distribution of DTI metrics (AD, FA, MD, & RD) within white matter, and so no non-trivial sufficient statistic is known for these distributions. To bridge this gap, we have devised a general framework which allows differences between a target and a reference distribution to be weighted according to a user supplied function. This allows custom statistics to be developed which may carry more information about independent variables of interest than traditional summary statistics are able to convey.

We begin by considering two cumulative distribution functions (CDF). Let F_R denote the CDF of some reference distribution. This could be the distribution of a group of healthy controls or an entire population. Let F_S denote the CDF of a single sample or subject. Our framework proposes statistics which take the form:

$$\int_l^u \phi\big(F_R^{-1}(x) - F_S^{-1}(x)\big)dx$$

Where ϕ is a weighting function applied to the differences of the inverse CDFs. Using the inverse CDF is preferred here, as the differences refer to the measure difference between corresponding quantiles. That is, $F_R^{-1}(0.5) - F_S^{-1}(0.5)$ can be interpreted as the measured difference between the medians of the two distributions. Furthermore, the use of inverse CDFs allows the integral to be defined cleanly in terms of quantiles as opposed to measure specific values.

It is worth noting the case when ϕ is the identity function, that is $\phi(x) = x$, we have

$$\begin{aligned}\int_l^u \big(F_R^{-1}(x) - F_S^{-1}(x)\big)dx &= \int_l^u F_R^{-1}(x) - \int_l^u F_S^{-1}(x) \\ &= \mathbb{E}_R\big(X|F_R^{-1}(l) < X < F_R^{-1}(u)\big) - \mathbb{E}_S\big(X|F_S^{-1}(l) < X < F_S^{-1}(u)\big)\end{aligned}$$

Noting that $\mathbb{E}_S(X|F_S^{-1}(l) < X < F_S^{-1}(u))$ is the conditional expectation of a random variable X with respect to distribution F. This is simply the difference between the truncated means of the two distributions. When compar-

ing subjects evaluated against the same reference, the truncated mean of the reference distribution can be treated as constant and thus ignored.

CHAPTER 4

Implementation

This was developed to perform numerical integration using Reimann sums over user specified functions (ϕ). These functions can be defined using Python 3 syntax and any functions available through the Numpy library. The reference distribution is first estimated by calculating the running average of cumulative bin frequencies over a user supplied reference list of images. Subjects are then evaluated in parallel against the generated reference.

—

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

Symbols

D

G

L

M

P

S