netcoloc Documentation

Release 0.1.7

Brin Rosenthal, Sophie Liu

Aug 11, 2023

CONTENTS

CHAPTER

ONE

DESCRIPTION

Here we introduce NetColoc, a tool which evaluates the extent to which two gene sets are related in network space, i.e. the extent to which they are colocalized in a molecular interaction network, and interrogates the underlying biological pathways and processes using multiscale community detection.

This framework may be applied to any number of scenarios in which gene sets have been associated with a phenotype or condition, including rare and common variants within the same disease, genes associated with two comorbid diseases, genetically correlated GWAS phenotypes, GWAS across two different species, or gene expression changes after treatment with two different drugs, to name a few.

NetColoc relies on a dual network propagation approach to identify the region of network space which is significantly proximal to both input gene sets, and as such is highly effective for small to medium input gene sets.

CHAPTER

TWO

DOCUMENTATION

For a quick-start on NetColoc's functionality, please see the example notebooks.

Usage Note: Please follow steps in example notebooks for correct usage of NetColoc. At this time, individual functionalities have not been tested for independent use.

CHAPTER

THREE

DEPENDENCIES

NetColoc requires the following python packages:

Note: All of the following packages minus DDOT and cdapsutil will be automatically installed via `pip install netcoloc`

- click
- matplotlib
- ndex2
- networkx
- numpy
- seaborn
- tqdm
- mygene >= 3.2.2
- scipy >= 1.5.3
- statsmodels
- gprofiler-official >= 1.0.0
- ipywidgets
- ipycytoscape
- python3 branch of DDOT

  DDOT can be installed one of two ways:

  1. To install DDOT by downloading the zip file of the source tree:

     ```
     wget https://github.com/idekerlab/ddot/archive/refs/heads/python3.zip
     unzip python3.zip
     cd ddot-python3
     python setup.py bdist_wheel
     pip install dist/ddot*py3*whl
     ```

  2. To install DDOT by cloning the repo:

```
git clone --branch python3 https://github.com/idekerlab/ddot.git
cd ddot
python setup.py bdist_wheel
pip install dist/ddot*py3*whl
```

Note: Due to dependency issue DDOT, will not work with Python 3.10 or later

Additional requirements for full functionality of example notebook:

- cdapsutil >= 0.2.0a1

CHAPTER

FOUR

INSTALLATION

NetColoc is available on PyPI

```
pip install netcoloc
```

CHAPTER

FIVE

LICENSE

- Free software: MIT license

CHAPTER

SIX

CITING NETCOLOC

Rosenthal, Sara Brin, Sarah N. Wright, Sophie Liu, Christopher Churas, Daisy Chilin-Fuentes, Chi-Hua Chen, Kathleen M. Fisch, Dexter Pratt, Jason F. Kreisberg, and Trey Ideker. "Mapping the common gene networks that underlie related diseases." Nature protocols (2023): 1-15.

https://www.nature.com/articles/s41596-022-00797-1

CHAPTER

SEVEN

CREDITS

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

Contents:

7.1 NetColoc

7.1.1 Description

Here we introduce NetColoc, a tool which evaluates the extent to which two gene sets are related in network space, i.e. the extent to which they are colocalized in a molecular interaction network, and interrogates the underlying biological pathways and processes using multiscale community detection.

This framework may be applied to any number of scenarios in which gene sets have been associated with a phenotype or condition, including rare and common variants within the same disease, genes associated with two comorbid diseases, genetically correlated GWAS phenotypes, GWAS across two different species, or gene expression changes after treatment with two different drugs, to name a few.

NetColoc relies on a dual network propagation approach to identify the region of network space which is significantly proximal to both input gene sets, and as such is highly effective for small to medium input gene sets.

7.1.2 Documentation

For a quick-start on NetColoc's functionality, please see the example notebooks.

Usage Note: Please follow steps in example notebooks for correct usage of NetColoc. At this time, individual functionalities have not been tested for independent use.

7.1.3 Dependencies

NetColoc requires the following python packages:

Note: All of the following packages minus DDOT and cdapsutil will be automatically installed via `pip install netcoloc`

- click
- matplotlib
- ndex2
- networkx
- numpy
- seaborn
- tqdm
- mygene >= 3.2.2
- scipy >= 1.5.3
- statsmodels
- gprofiler-official >= 1.0.0
- ipywidgets
- ipycytoscape
- python3 branch of DDOT

  DDOT can be installed one of two ways:

  1. To install DDOT by downloading the zip file of the source tree:

     ```
     wget https://github.com/idekerlab/ddot/archive/refs/heads/python3.zip
     unzip python3.zip
     cd ddot-python3
     python setup.py bdist_wheel
     pip install dist/ddot*py3*whl
     ```

  2. To install DDOT by cloning the repo:

     ```
     git clone --branch python3 https://github.com/idekerlab/ddot.git
     cd ddot
     python setup.py bdist_wheel
     pip install dist/ddot*py3*whl
     ```

  Note: Due to dependency issue DDOT, will not work with Python 3.10 or later

Additional requirements for full functionality of example notebook:

- cdapsutil >= 0.2.0a1

7.1.4 Installation

NetColoc is available on PyPI

```
pip install netcoloc
```

7.1.5 License

- Free software: MIT license

7.1.6 Citing NetColoc

Rosenthal, Sara Brin, Sarah N. Wright, Sophie Liu, Christopher Churas, Daisy Chilin-Fuentes, Chi-Hua Chen, Kathleen M. Fisch, Dexter Pratt, Jason F. Kreisberg, and Trey Ideker. "Mapping the common gene networks that underlie related diseases." Nature protocols (2023): 1-15.

https://www.nature.com/articles/s41596-022-00797-1

7.1.7 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

7.2 Installation

netcoloc depends on DDOT which has to be manually installed.

DDOT can be installed one of two ways:

1. To install DDOT by downloading the zip file of the source tree:

```
wget https://github.com/idekerlab/ddot/archive/refs/heads/python3.zip
unzip python3.zip
cd ddot-python3
python setup.py bdist_wheel
pip install dist/ddot*py3*whl
```

2. To install DDOT by cloning the repo:

```
git clone --branch python3 https://github.com/idekerlab/ddot.git
cd ddot
python setup.py bdist_wheel
pip install dist/ddot*py3*whl
```

7.2.1 Stable release

To install netcoloc, run this command in your terminal:

```
$ pip install netcoloc
```

If you don't have pip installed, this Python installation guide can guide you through the process.

7.2.2 From sources

The sources for NetColoc can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/ucsd-ccbb/NetColoc
```

Or download the tarball:

```
$ curl  -OL https://github.com/ucsd-ccbb/NetColoc/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

7.3 Usage

To use netcoloc in a project:

```
import netcoloc
```

7.4 netcoloc package

7.4.1 netcoloc.netcoloc_utils module

Utility functions useful across multiple modules.

netcoloc.netcoloc_utils.**get_degree_binning**(*node_to_degree_dict*, *min_bin_size*, *lengths=None*)

Groups nodes by degree into similarly sized bins. This function comes from network_utilities.py of emreg00/toolbox

Returns a tuple with following two values:

- **list of bins** where each bin contains a list of nodes of similar degree
- **mapping of degree to index of bin** dict mapping a degree to the index of the bin in the bins list which contains nodes of that degree

Parameters

- **node_to_degree_dict** (*dict*) – Map of nodes to their degrees
- **min_bin_size** (*int*) – minimum number of nodes each bin should contain.

- **lengths** (*list*) – List of nodes to bin. If lengths is equal to None, then all nodes will be binned

Returns
(list of bins, mapping of degree to index of bin)

Return type
tuple

7.4.2 netcoloc.netprop module

Functions for performing network propagation

netcoloc.netprop.**get_individual_heats_matrix**(*nam_or_graph*, *alpha=0.5*, *conserve_heat=True*, *weighted=False*)

Returns the pre-calculated contributions of each individual gene in the interactome to the final heat of each other gene in the interactome after propagation.

Changed in version 0.1.6: In addition, to a normalized adjacency matrix, this function now also supports `networkx.Graph` network as input

If a `networkx.Graph` network is passed in as the **nam_or_graph** parameter, the function *get_normalized_adjacency_matrix()* is called to generate the normalized adjacency matrix using **conserve_heat** and **weighted** parameters

---

Note: Resulting matrix from this function can be saved to a file with `numpy.save()` and loaded later with `numpy.load()`, but resulting file can be several gigabytes and take a minute or more to save/load.

```
numpy.save('heats_matrix.npy', w_double_prime)
w_double_prime = numpy.load('heats_matrix.npy')
```

Parameters

- **nam_or_graph** (`numpy.ndarray` or `networkx.Graph`) – square normalized adjacency matrix or network
- **alpha** (*float*) – heat dissipation coefficient between 1 and 0. The contribution of the heat propagated from adjacent nodes in determining the final heat of a node, as opposed to the contribution from being a part of the gene set initially
- **conserve_heat** (*bool*) – If `True`, heat will be conserved (ie. the sum of the heat vector will be equal to 1), and the graph will be asymmetric. Otherwise, heat will not be conserved, and the graph will be symmetric. **NOTE:** Only applies if **nam_or_graph** is `networkx.Graph`
- **weighted** (*bool*) – If `True`, then the graph's edge weights will be taken into account. Otherwise, all edge weights will be set to 1. **NOTE:** Only applies if **nam_or_graph** is `networkx.Graph`

Returns
square individual heats matrix

Return type
`numpy.ndarray`

netcoloc.netprop.**get_normalized_adjacency_matrix**(*graph*, *conserve_heat=True*, *weighted=False*)

Returns normalized adjacency matrix (W'), as detailed in:

Vanunu, Oron, et al. 'Associating genes and protein complexes with disease via network propagation.'

With version *0.1.6* and newer, the networkx.Graph can be directly passed into *get_individual_heats_matrix()* and this method will be invoked to create the normalized adjacency matrix

---

Note: Resulting matrix from this function can be saved to a file with numpy.save() and loaded later with numpy.load(), but resulting file can be several gigabytes and take a minute or more to save/load.

```
numpy.save('nam.npy', adjacency_matrix)
adjacency_matrix = numpy.load('nam.npy')
```

Parameters

- **graph** (networkx.Graph) – Interactome from which to calculate normalized adjacency matrix.
- **conserve_heat** (*bool*) – If True, heat will be conserved (ie. the sum of the heat vector will be equal to 1), and the graph will be asymmetric. Otherwise, heat will not be conserved, and the graph will be symmetric.
- **weighted** (*bool*) – If True, then the graph's edge weights will be taken into account. Otherwise, all edge weights will be set to 1.

Returns
Square normalized adjacency matrix

Return type
numpy.ndarray

netcoloc.netprop.**network_propagation**(*individual_heats_matrix*, *nodes*, *seed_genes*)

Implements network propagation, as detailed in:

Vanunu, Oron, et al. 'Associating genes and protein complexes with disease via network propagation.'

Using this function, the final heat of the network is calculated directly, instead of iteratively. This method is faster when many different propagations need to be performed on the same network (with different seed gene sets). It is slower than iterative_network_propagation() for a single propagation.

Parameters

- **individual_heats_matrix** (numpy.ndarray) – Square matrix that is the output of *get_individual_heats_matrix()*
- **nodes** (*list*) – List of nodes in the network represented by the individual_heats_matrix, in the same order in which they were supplied to *get_individual_heats_matrix()*
- **seed_genes** (*list*) – Input list of genes/nodes for intializing the heat in network propagation. Any items in *seed genes* that are not present in *nodes* will be ignored.

Returns
Final heat of each node after propagation, with the name of the nodes as the index

Return type
pandas.Series

7.4.3 netcoloc.netprop_zscore module

Functions for getting z-scores from network propagation.

netcoloc.netprop_zscore.**calculate_heat_zscores**(*individual_heats_matrix*, *nodes*, *degrees*, *seed_genes*, *num_reps=10*, *alpha=0.5*, *minimum_bin_size=10*, *random_seed=1*)

Helper function to perform network heat propagation using the given individual heats matrix with the given seed genes and return the z-scores of the final heat values of each node.

The z-scores are calculated based on a null model, which is built by running the network propagation multiple times using randomly selected seed genes with similar degree distributions to the original seed gene set.

The returned tuple contains the following:

- pandas.Series containing z-scores for each gene. Gene names comprise the index column
- pandas.Series containing the final heat scores for each gene. Gene names comprise the index column,
- numpy.ndarray containing square matrix in which each row contains the final heat scores for each gene from a network propagation from random seed genes)

Parameters

- **individual_heats_matrix** (numpy.ndarray) – output of the netprop.get_individual_heats_matrix. A square matrix containing the final heat contributions of each gene
- **nodes** (*list*) – nodes, in the order in which they were supplied to the *get_normalized_adjacency_matrix()* method which returns the precursor to the individual_heats_matrix
- **degrees** (*dict*) – Mapping of node names to node degrees
- **seed_genes** (*list*) – list of genes to use for network propagation. The results of this network propagation will be compared to a set of random results in order to obtain z-scores
- **num_reps** (*int*) – Number of times the network propagation algorithm should be run using random seed genes in order to build the null model
- **alpha** (*float*) – Number between 0 and 1. Denotes the importance of the propagation step in the network propagation, as opposed to the step where heat is added to seed genes only. Recommended to be 0.5 or greater
- **minimum_bin_size** (*int*) – minimum number of genes that should be in each degree matching bin
- **random_seed** –

Returns
(pandas.Series, pandas.Series, numpy.ndarray)

Return type
tuple

netcoloc.netprop_zscore.**netprop_zscore**(*seed_gene_file*, *seed_gene_file_delimiter=None*, *num_reps=10*, *alpha=0.5*, *minimum_bin_size=10*, *interactome_file=None*, *interactome_uuid='f93f402c-86d4-11e7-a10d-0ac135e8bacf'*, *ndex_server='public.ndexbio.org'*, *ndex_user=None*, *ndex_password=None*, *out_name='out'*, *save_z_scores=False*, *save_final_heat=False*, *save_random_final_heats=False*, *verbose=True*)

Performs network heat propagation on the given interactome with the given seed genes, then returns the z-scores of the final heat values of each node in the interactome.

The z-scores are calculated based on a null model, which is built by running the network propagation multiple times using randomly selected seed genes with similar degree distributions to the original seed gene set.

This method returns a tuple containing the following:

- `pandas.Series` containing z-scores for each gene. Gene names comprise the index column
- `numpy.ndarray` containing square matrix where each row contains the final heat scores for each gene from a network propagation from random seed genes

Parameters

- **seed_gene_file** (*str*) – Location of file containing a delimited list of seed genes
- **seed_gene_file_delimiter** (*str*) – Delimiter used to separate genes in seed gene file. Default any whitespace
- **num_reps** (*int*) – Number of times the network propagation algorithm should be run using random seed genes in order to build the null model
- **alpha** (*float*) – Number between 0 and 1. Denotes the importance of the propagation step in the network propagation, as opposed to the step where heat is added to seed genes only. Recommended to be 0.5 or greater
- **minimum_bin_size** (*int*) – minimum number of genes that should be in each degree matching bin.
- **interactome_file** (*str*) – Location of file containing the interactome in NetworkX gpickle format. Either the interactome_file argument or the interactome_uuid argument must be defined.
- **interactome_uuid** (*str*) – UUID of the interactome on NDEx. Either the interactome_file argument or the interactome_uuid argument must be defined. (Default: The UUID of PCNet, the Parsimonious Composite Network: f93f402c-86d4-11e7-a10d-0ac135e8bacf)
- **ndex_server** (*str*) – NDEx server on which the interactome is stored. Only needs to be defined if interactome_uuid is defined
- **ndex_user** (*str*) – NDEx user that the interactome belongs to. Only needs to be defined if interactome_uuid is defined, and the interactome is private
- **ndex_password** (*str*) – password of the NDEx user's account. Only needs to be defined if interactome_uuid is defined, and the interactome is private
- **out_name** (*str*) – Prefix for saving output files
- **save_z_scores** –
- **save_final_heat** (*bool*) – If `True`, then the raw network propagation heat scores for the original seed gene set will be saved in the form of a tsv file in the current directory
- **save_random_final_heats** (*bool*) – If `True`, then the raw network propagation heat scores for every repetition of the algorithm using random seed genes will be saved in the form of a tsv file in the current directory. (Beware: This can be a large file if num_reps is large.)
- **verbose** – If `True`, then progress information will be logged. Otherwise, nothing will be printed

Returns
(pandas.Series, numpy.ndarray)

Return type
tuple

Raises
TypeError – If neither interactome_file or interactome_uuid is provided or if **num_reps** is not an int

7.4.4 netcoloc.network_colocalization module

Functions for performing network colocalization

netcoloc.network_colocalization.**calculate_expected_overlap**(*z_scores_1*, *z_scores_2*, *z_score_threshold=3*, *z1_threshold=1.5*, *z2_threshold=1.5*, *num_reps=1000*, *plot=False*)

Determines size of expected network overlap by randomly shuffling gene names

Parameters

- **z_scores_1** (pandas.Series) – Result from *netprop_zscore()* or *calculate_heat_zscores()* containing the z-scores of each gene following network propagation. The index consists of gene names
- **z_scores_2** (pandas.Series) – Similar to **z_scores_1**. This and **z_scores_1** must contain the same genes (ie. come from the same interactome network)
- **z_score_threshold** (*float*) – threshold to determine whether a gene is a part of the network overlap or not. Genes with combined z-scores below this threshold will be discarded
- **z1_threshold** (*float*) – individual z1-score threshold to determine whether a gene is a part of the network overlap or not. Genes with z1-scores below this threshold will be discarded
- **z2_threshold** (*float*) – individual z2-score threshold to determine whether a gene is a part of the network overlap or not. Genes with z2-scores below this threshold will be discarded
- **num_reps** (*int*) – Number of repitions of randomly shuffling input z_score vectors to generate null distribution
- **plot** (*bool*) – If True, distribution will be plotted

Returns
Observed overlap size, and vector of randomized overlap sizes from permuted z_scores

Return type
float, np.array of floats

netcoloc.network_colocalization.**calculate_network_enrichment**(*z_D1*, *z_D2*, *zthresh_list=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]*, *z12thresh_list=[1, 1.5, 2]*, *verbose=True*)

Evaluate NetColoc enrichment for a range of thresholds on network proximity z-scores.

Parameters

- **z_D1** (pandas.DataFrame) – DataFrame containing gene names and network proximity z-scores for the first gene set

- **z_D2** (pandas.DataFrame) – DataFrame containing gene names and network proximity z-scores for the second gene set
- **zthresh_list** (*:list*) – list of combined z-score thresholds to iterate over
- **z12thresh_list** (*:list*) – list of individual z-score thresholds to iterate over
- **verbose** (*Boolean*) – if True, print out some diagnostics

Returns
netcoloc_enrichment_df: DataFrame containing NetColoc enrichment results

Return type
pandas.DataFrame

netcoloc.network_colocalization.**calculate_network_overlap**(*z_scores_1*, *z_scores_2*, *z_score_threshold=3*, *z1_threshold=1.5*, *z2_threshold=1.5*)

Function to determine which genes overlap. Returns a list of the overlapping genes

Parameters

- **z_scores_1** (pandas.Series) – Result from *netprop_zscore()* or *calculate_heat_zscores()* containing the z-scores of each gene following network propagation. The index consists of gene names
- **z_scores_2** (pandas.Series) – Similar to **z_scores_1**. This and **z_scores_1** must contain the same genes (ie. come from the same interactome network)
- **z_score_threshold** (*float*) – threshold to determine whether a gene is a part of the network overlap or not. Genes with combined z-scores below this threshold will be discarded
- **z1_threshold** (*float*) – individual z1-score threshold to determine whether a gene is a part of the network overlap or not. Genes with z1-scores below this threshold will be discarded
- **z2_threshold** (*float*) – individual z2-score threshold to determine whether a gene is a part of the network overlap or not. Genes with z2-scores below this threshold will be discarded

Returns
genes in the network overlap (genes with high combined z-scores)

Return type
list

netcoloc.network_colocalization.**calculate_network_overlap_subgraph**(*interactome*, *z_scores_1*, *z_scores_2*, *z_score_threshold=3*, *z1_threshold=1.5*, *z2_threshold=1.5*)

Function to return subgraph of network intersection.

Code to create subgraph is from NetworkX subgraph documentation

Parameters

- **interactome** (networkx.Graph) – network whose subgraph will be returned
- **z_scores_1** (pandas.Series) – Result from *netprop_zscore()* or *calculate_heat_zscores()* containing the z-scores of each gene following network propagation. The index consists of gene names
- **z_scores_2** (pandas.Series) – Similar to **z_scores_1**. This and **z_scores_1** must contain the same genes (ie. come from the same interactome network)

- **z_score_threshold** (*float*) – threshold to determine whether a gene is a part of the network overlap or not. Genes with combined z-scores below this threshold will be discarded
- **z1_threshold** (*float*) – individual z1-score threshold to determine whether a gene is a part of the network overlap or not. Genes with z1-scores below this threshold will be discarded
- **z2_threshold** (*float*) – individual z2-score threshold to determine whether a gene is a part of the network overlap or not. Genes with z2-scores below this threshold will be discarded

Returns
Subgraph of the interactome containing only genes that are in the network intersection (genes with high combined z-scores)

Return type
networkx.Graph

netcoloc.network_colocalization.**sweep_input_pvals**(*D1_df*, *D2_df*, *individual_heats_matrix*, *nodes*, *degrees*, *cutoff_list=[0.01, 0.02, 0.03, 0.04, 0.05, 0.1]*, *gene_column='gene'*, *score_column='pval'*, *cutoff_max=True*, *num_reps=100*, *verbose=True*, *z_score_threshold=3*, *z12_threshold=1.5*)

Evaluate NetColoc enrichment for a range of thresholds on input gene lists.

Parameters

- **D1_df** (pandas.DataFrame) – DataFrame containing gene names and scores for the first gene set
- **D2_df** (pandas.DataFrame) – DataFrame containing gene names and scores for the second gene set
- **individual_heats_matrix** (numpy.ndarray) – output of the netprop.get_individual_heats_matrix. A square matrix containing the final heat contributions of each gene
- **nodes** (*list*) – nodes, in the order in which they were supplied to the *get_normalized_adjacency_matrix()* method which returns the precursor to the individual_heats_matrix
- **degrees** (*dict*) – Mapping of node names to node degrees
- **cutoff_list** (*list*) – list of values to threshold the input gene sets by
- **gene_column** (*string*) – name of column containing genes in D1_df and D2_df
- **score_column** (*string*) – name of column containing scores (usually p-value or log fold change) in D1_df and D2_df
- **cutoff_max** (*Boolean*) – if True, genes will be selected which have scores less than the cutoff value. If false, genes will be selected which have scores greater than the cutoff value.
- **num_reps** (*int*) – Number of times the network propagation algorithm should be run using random seed genes in order to build the null model
- **verbose** (*Boolean*) – if True, print out some diagnostics
- **z_score_threshold** (*float*) – threshold to determine whether a gene is a part of the network overlap or not. Genes with combined z-scores below this threshold will be discarded
- **z12_threshold** (*float*) – individual z1/z2-score threshold to determine whether a gene is a part of the network overlap or not. Genes with z1/z2-scores below this threshold will be discarded

Returns
netcoloc_pval_df: DataFrame containing NetColoc enrichment results

Return type
pandas.DataFrame

netcoloc.network_colocalization.**transform_edges**(*G*, *method='cosine_sim'*, *edge_weight_threshold=0.95*)

Transforms binary edges using selected method (currently only cosine similarity is implemented). Cosine similarity measures the similarity between neighbors of node pairs in the input network

Parameters

- **G** (networkx.Graph) – network whose edges will be transformed
- **method** (*str*) – Method to use, only `cosine_sim` supported. Any other value will cause this method to output a warning and immediately return
- **edge_weight_threshold** (*float*) – Transformed edges will be returned which have values greater than this

Returns
Graph with nodes identical to input G, but with transformed edges (values > edge_weight_threshold)

Return type
networkx.Graph

netcoloc.network_colocalization.**view_G_hier**(*G_hier*, *layout='cose'*)

In-notebook visualization of NetColoc hierarchy, using ipycytoscape.

Parameters

- **G_hier** – network to visualize. Expects output of cdapsutil.CommunityDetection(), transformed to networkx format. 'CD_MemberList_LogSize' is a required field of the network to map to the node size.
- **layout** (*str*) – Layout method to use, any layout supported by cytoscape.js is supported. Suggest 'cose' or 'breadthfirst'.
- **edge_weight_threshold** (*float*) – Transformed edges will be returned which have values greater than this

Returns
Nothing

7.4.5 netcoloc.network_localization module

7.4.6 netcoloc.validation module

Functions for performing validation of NetColoc subgraph

netcoloc.validation.**MPO_enrichment_full**(*hier_df*, *MPO*, *mgi_df*, *MP_focal_list*, *G_int*)

Function to test for enrichment of genes resulting in selected phenotypes when knocked out in every NetColoc system (not just root)

The returned pandas.DataFrame will have these columns:

- **log(OR_p)** - -log10(Odds ratio p-vlaue)
- **log_OR** - natural log odds ratio

- **num_genes** - number of genes in MPO term overlapping with focal system
- **gene_ids** - list of overlapping genes between MPO term and focal system

Parameters

- **hier_df** (pandas.DataFrame) – NetColoc systems map (processed output from cdaps_util)
- **MPO** (ddot.Ontology) – DDOT ontology containing the parsed mammalian phenotype ontology
- **mgi_df** (pandas.DataFrame) – parsed MGI knockout dataframe
- **MP_focal_list** (*list*) – List of MPO phenotypes to check for enrichment against
- **G_int** (networkx.Graph) – Background interactome

Returns
Dataframe containing enrichment results

Return type
pandas.DataFrame

netcoloc.validation.**MPO_enrichment_root**(*hier_df*, *MPO*, *mgi_df*, *MP_focal_list*, *G_int*, *verbose=True*)

Function to test for enrichment of genes resulting in selected phenotypes when knocked out in root node of NetColoc hierarchy.

The returned pandas.DataFrame will have the following columns:

- **OR_p** - Odds ratio p-vlaue
- **log_OR** - natural log odds ratio
- **log_OR_CI_lower** - lower 95% confidence interval on log_OR
- **log_OR_CI_upper** - upper 95% confidence interval on log_OR
- **num_genes_in_term** - number of genes in MPO term
- **MP_description** - description of MPO phenotype

Parameters

- **hier_df** (pandas.DataFrame) – NetColoc systems map (processed output from cdaps_util)
- **MPO** (ddot.Ontology) – DDOT ontology containing the parsed mammalian phenotype ontology
- **mgi_df** (pandas.DataFrame) – parsed MGI knockout dataframe
- **MP_focal_list** (*list*) – List of MPO phenotypes to check for enrichment against
- **G_int** (networkx.Graph) – Background interactome
- **verbose** (*bool*) – If true, print out some progress

Returns
Dataframe containing enrichment results

Return type
pandas.DataFrame

netcoloc.validation.**load_MGI_mouseKO_data**(*url='http://www.informatics.jax.org/downloads/reports/MGI_PhenoGenoMP.rpt*

Function to parse and load mouse knockout data from MGI.

Parameters
url (*str*) – location of MGI knockout data

Returns
parsed MGI knockout dataframe, including column for human orthologs

Return type
pandas.DataFrame

netcoloc.validation.**load_MPO**(*url='http://www.informatics.jax.org/downloads/reports/MPheno_OBO.ontology'*)

Function to parse and load mouse phenotype ontology, using DDOT's ontology module

Parameters
url (*str*) – URL containing MPO ontology file

Returns
MPO parsed using DDOT

Return type
ddot.Ontology

Raises
ImportError – If DDOT package is not found

7.4.7 Module contents

7.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/ucsd-ccbb/NetColoc/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

NetColoc could always use more documentation, whether as part of the official NetColoc docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ucsd-ccbb/NetColoc/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.5.2 Get Started!

Ready to contribute? Here's how to set up *NetColoc* for local development.

1. Fork the *NetColoc* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/NetColoc.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv NetColoc
$ cd NetColoc/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 NetColoc tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/ucsd-ccbb/NetColoc/pull_requests and make sure that the tests pass for all supported Python versions.

7.5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_netcoloc
```

7.6 Credits

7.6.1 Development Lead

- Brin Rosenthal <sbrosenthal@ucsd.edu>

7.6.2 Contributors

- Sophie Liu <sol015@ucsd.edu>

7.7 History

7.7.1 0.1.7 (2022-06-28)

- Removed unused *network_localization.py* module

7.7.2 0.1.6 (2022-06-16)

- ipycytoscape added as a dependency
- ipywidgets added as a dependency
- Added *network_colocalization.sweep_input_pvals()* to sweep p-values and scores
- Added *network_colocalization.calculate_network_enrichment()* to sweep over z-score thresholds
- *netprop.get_individual_heats_matrix()* can take a networkx *Graph* object and internally call *netprop.get_normalized_adjancency_matrix()*. Documentation updated in both methods to note that the resulting matrices can be saved via *numpy.save()* and retrieved via *numpy.load()*
- *example_notebooks/ASD_CHD_NetColoc_analysis.ipynb* now visualizes hierarchy using ipycytoscape
- *example_notebooks/ASD_CHD_NetColoc_analysis.ipynb* updated with a note about using *numpy.save()* and *numpy.load()* to save and retrieve result from *netprop.get_individual_heats_matrix()*

7.7.3 0.1.5 (2022-03-09)

- Fixed divide by zero error seen when calculating cosine distance by updating *netprop.get_normalized_adjancency_matrix()* to properly normalize an adjacency matrix that is asymetric (UD-1863)

7.7.4 0.1.4 (2021-08-31)

- If import of DDOT package fails, only a warning message will be displayed unless user invokes `netcoloc.validation.load_MPO()` in which case an `ImportError` is raised
- Fixed bug where `z1_threshold` parameter was being passed to `z2_threshold` parameter in `netcoloc.network_cololcalization.calcualte_network_overlap` method called by `netcoloc.network_colocalization.calculate_network_overlap_subgraph` method

7.7.5 0.1.3 (2021-08-18)

- Added dependency gprofiler-official to `setup.py` and `requirements.txt` because this is used by `network_colocalization.py`

7.7.6 0.1.2 (2021-08-17)

- Added new *validation.py* module containing mouse knockout database functionality

7.7.7 0.1.1 (2021-08-06)

- Fixed netcoloc imports in *netprop_zcore.py*

7.7.8 0.1.0 (2021-03-10)

- First release on PyPI.

7.8 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX