HiDeF

Release 1

Fan Zheng, She Zhang

Oct 02, 2021

CONTENTS

HiDeF[1] aims to reimagine hierarchical data clustering. The name HiDeF stands for "Hierarchical community Decoding Framework". HiDeF integrates graph-based community detection and the idea of "persistent homology" in order to determine robust clustering patterns in complex data at multiple scales. Given the inputs in data points in graph or matrix formats, HiDeF returns a list of multiscale clusters with measurement of their robustness, as well as a directed acyclic graph (DAG) to represent the organization of these clusters.

[1] Zheng, F, Zhang, S, et al. HiDeF: identifying persistent structures in multiscale 'omics data. *Genome Biology*, 22, 21 (2021).

CHAPTER

ONE

INSTALLATION

1.1 Local installation of python package

From source:

```
python setup.py install
```

Installation via `pip`

```
pip install hidef
```

1.2 Cytoscape

HiDeF is separately distributed via the CDAPS framework[2] in Cytoscape.

Note: We try to maintain timely synchronization of the HiDeF versions across the Python package and Cytoscape. However, it may be possible to have small difference in results across the platforms due to the Cytoscape version is behind the latest version of the Python package.

1.2.1 What's new

Version 1.1.3:

- Community detection with multiple resolutions now run in parallel with python multiprocessing module
- The default algorithm changed to Leiden as it is faster than Louvain
- Now support multiplex community detection

[2] Singhal, A. Cao, S. Churas, C. et al. Multiscale community detection in Cytoscape. *PLoS Comput. Biol.* 16, e1008239 (2020).

1.2.2 Tutorials

Tutorials

Organize a list of clusters into a DAG

HiDeF is flexible and the construction of DAG is independent of the upstream clustering algorithm. The `weaver` module can take an arbitrary list of clusters and infer the relationships among them. We defined containment index

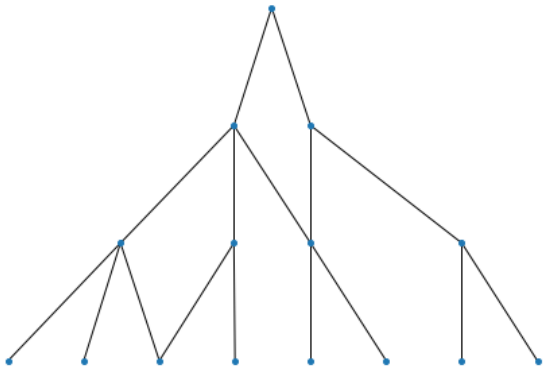
$$CI(v, w) = \frac{|s(v) \cap s(w)|}{|s(w)|}$$

as the fraction of data points in cluster *w* that is also in cluster *v*. In the following example, all data points of cluster at line 2 is also members of the cluster at line 1, so we use a directed edge to represent this relationship.

```
from hidef import *
wv = weaver.Weaver()

P = ['11111111',
     '11111100',
     '00001111',
     '11100000',
     '00110000',
     '00001100',
     '00000011']
wv = weaver.Weaver()
H = wv.weave(P, cutoff=1.0)

wv.show() # need pydot and graphviz
```



Note: The last command `wv.show` is a convenient function to quickly browse the model, but needs additional dependencies pydot and graphviz. See more discussion here: *Render the model*.

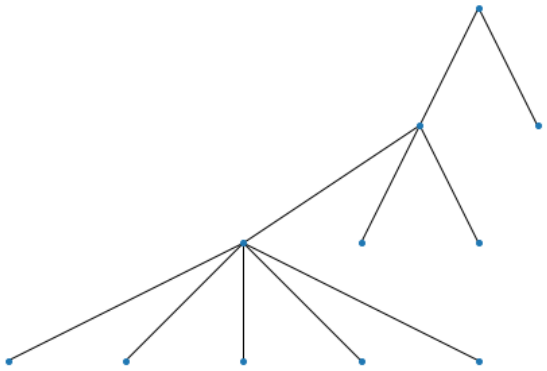
In many cases, it could be informative to relax the requirement of $CI = 1$ to reveal the relationships between clusters. Thus we provide a parameter τ ("cutoff" in the above code block) to approximate the containment relationships, as can be seen in the following example.

```
P = ['11111111',
     '11111100',
```

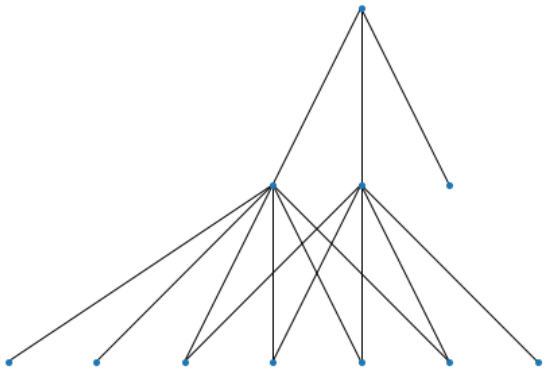
(continues on next page)

(continued from previous page)

```
     '00111110'] # cluster 3 almost contained in cluster 2
H = wv.weave(P, cutoff=0.75) # containment threshold relaxed here
wv.show()
```

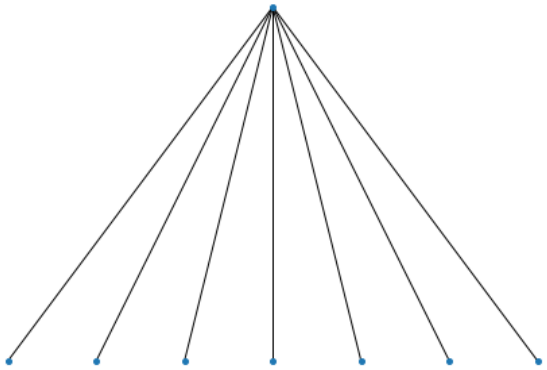


```
P = ['11111111',
     '11111100',
     '00111110'] # cluster 3 almost contained in cluster 2
H = wv.weave(P, cutoff=1.0) # strict containment
wv.show()
```

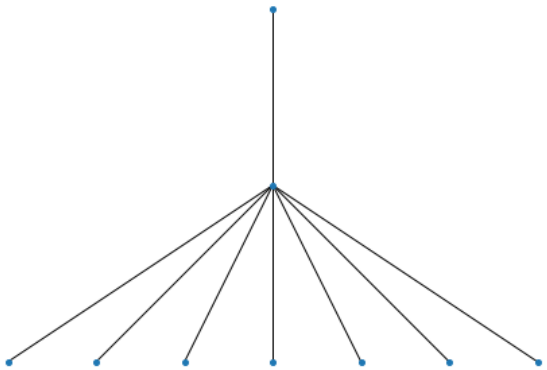


Note: Since we do not restrict the input, it is possible to have mutually contained (even identical) clusters. In such cases, we decide the two (or more) mutually contained clusters as "similar", and there is an option to merge them.

```
P = ['11111110',
     '11111110']
H = wv.weave(P, merge=True)
wv.show()
```

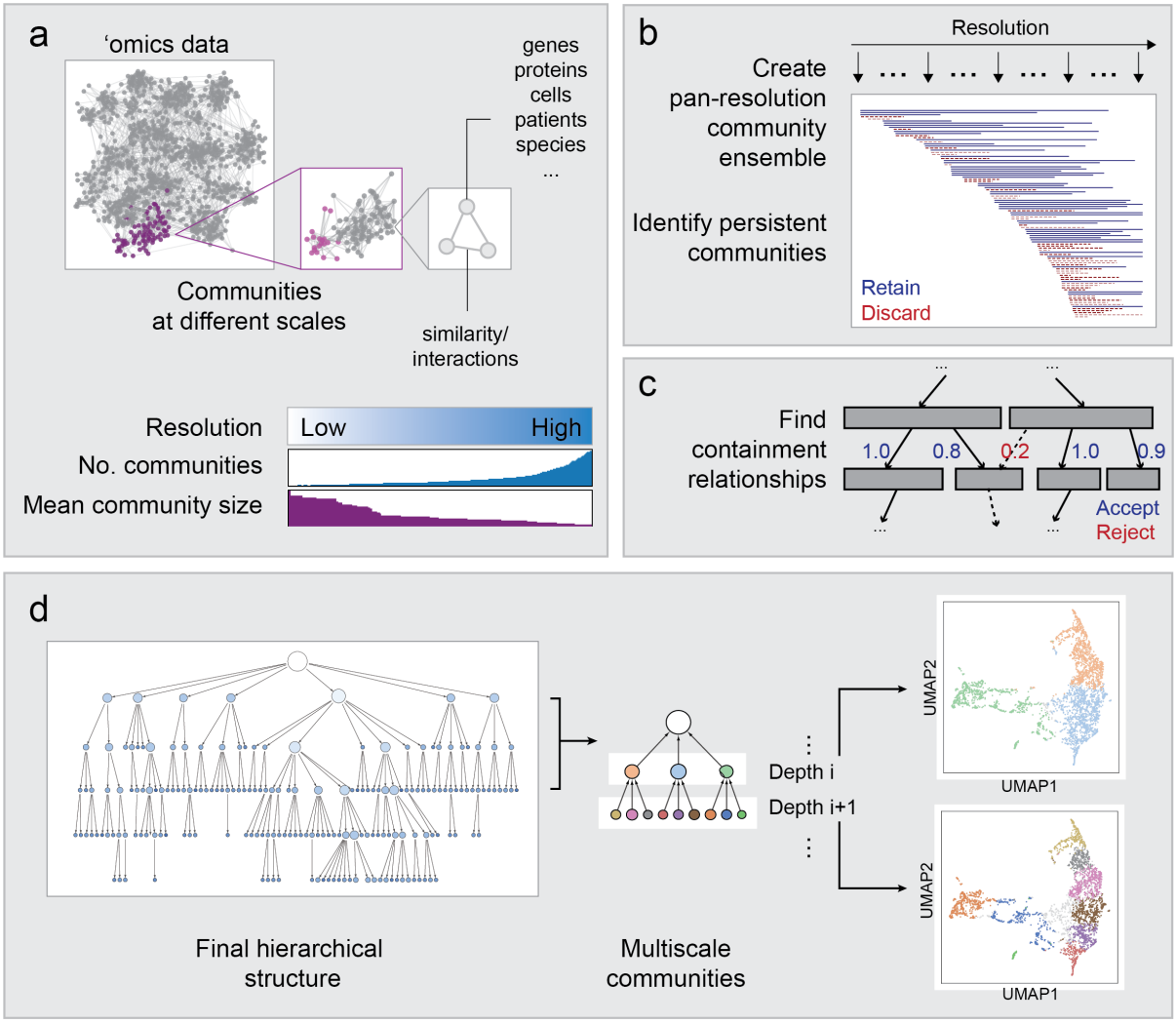


```
P = ['11111110',
     '11111110']
H = wv.weave(P, merge=False) # without merge, the DAG contains duplicated cluster
wv.show()
```



Although an arbitrarily defined set of clusters can be used, a routine workflow is to use network community detection to generate a set of clusters (see the next topic).

Generate persistent clusters



HiDeF ultilizes graph-based community detection (CD) with resolution parameters (such as Louvain[1] or Leiden[2] algorithms) to detect clusters. The resolution parameter was sampled at many values, and a set of communities was generated at each resolution. The solutions at each resolution were compared to each other. We define "persistent" communities as those repetitively discovered in many different resolutions.

As graph-based CD methods are in general heuristic, it is not common to find identical clusters repetitively. However, we can measure similarity between clusters. Imagining a graph in which each node represent a community and each edge connects two similar communities, a component in this graph a group of similar communities, and the size of these components can be seen as a metric of community persistence.

The following example takes the BioPlex[3] protein-protein interaction network, scans the resolution parameter from 0.001 to 25, and returns the communities with persistence larger than 5 (default parameters). Source files can be found in the `examples` directory.

[1] Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. J. Stat. Mech. 2008, P10008 (2008)

[2] Traag, V. A., Waltman, L. & van Eck, N. J. From Louvain to Leiden: guaranteeing well-connected communities. Sci. Rep. 9, 5233 (2019)

[3] Huttlin, E. L. et al. Architecture of the human interactome defines protein communities and disease networks. Nature 545, 505–509 (2017)

```
python hidef_finder.py --g BioPlex.tsv  --o BioPlex [--options]
```

Output formats

*.nodes: a .tsv file in which each row represents a community. The first 3 columns being the name, the size, and the elements of the community. See hidef_finder.output_nodes.

*.edges a .tsv file in which each row represents a directed edge. The first and second column indicate the target (parent) and source (child community, respectively. See hidef_finder.output_edges.

*.gml a file in the GML (Graph Modelling Language) format to represent the DAG. See hidef_finder.output_gml.

*.gmt GMT format as described in GSEA. (Coming soon)

Use the feature matrix of data points as input

The default input of HiDeF is a format representing a graph. Still, if the input data is in the format of feature vectors of data points, a k-nearest neighbor graph (kNN) or a shared k-nearest neighbor graph (sNN) can be generated by other methods, such as umap.umap_.nearest_neighbors or sklearn.neighbors.NearestNeighbors. HiDeF function jaccard_matrix can help convert kNN to sNN:

```
from hidef import utils
mat_knn = scipy.sparse.csr_matrix(mat_knn)
indices = utils.jaccard_matrix(mat_knn, mat_knn, threshold=0.1)
with open(outfile, 'w') as fh:
    for i in range(len(indices[0])):
        fh.write('{}\t{}\n'.format(indices[0][i], indices[1][i]))
```

Note: Building kNN or sNN graph followed by graph-based community detection is a common workflow of single-cell analysis in prominent software packages such as Seurat or Scanpy. However, the resolution parameter needs to be specified in those packages. HiDeF scans resolution parameters and makes it easy to identify structures at more diverse scales.

Browse and interact with the hierarchical models

Coming soon

Render the model

Simplify model by removing communities with low persistence

Find all communities containing certain element

Synchronized visualization with 2D data projection

Compare two hierarchical models

1.2.3 API

API

hidef_finder module

class hidef.hidef_finder.**Cluster**(*binary*, *gamma*)
The base class representing a cluster in hidef.

Parameters

- **binary** (*np.array*) – a binary vector indicating which objects belong to this cluster
- **gamma** (*float*) – the resolution parameter which generated this cluster

calculate_similarity(*cluster2*)
calculate the Jaccard similarity between two clusters

Parameters **cluster2** (hidef_finder.Cluster) –

Returns

Return type float

class hidef.hidef_finder.**ClusterGraph**(*incoming_graph_data=None*, ***attr*)
Extending nx.Graph class, each node is a hidef_finder.Cluster object

add_clusters(*resolution_graph*, *new_resolution*)
Add new clusters to cluster graph once a new resolution is finished by the CD algorithm

Parameters

- **resolution_graph** (*nx.Graph*) – a graph in which nodes represent resolutions in the scan, and edges connecting the resolutions that are considered 'neighbors'
- **new_resolution** (*float*) – the resolution just visited by the CD algorithm

hidef.hidef_finder.**collapse_cluster_graph**(*cluG*, *components*, *p=100*)
take the cluster graph and collapse each component based on some consensus metric

Parameters

- **cluG** (hidef_finder.ClusterGraph) –
- **components** (*list of list of int*) – elements of the inner list are numbers to query a cluster in the cluster graph
- **p** (*int or float*) – remove nodes if they did not appear in more than t percent of clusters in one component (the p parameter in the paper)

Returns **collapsed_clusters** – binary numpy arrays indicating the members of each cluster after filtering by consensus

Return type a list of np.array

See also:

consensus

hidef.hidef_finder.**consensus**(*cluG*, *k=5*, *f=1.0*, *p=100*)
create a more parsimonious results from the cluster graph

Parameters

- **cluG** (hidef_finder.ClusterGraph) –
- **k** (*int*) – delete components with lower size than this threshold. The 'chi' parameter in the paper
- **f** (*float*) – take this fraction of clusters (ordered by degree in cluster graph)
- **p** (*int*) – nodes that do not participate in the majority of clusters in a component will be removed

Returns **cluG_collapsed_w_len** – the array indicates the member of clusters; the integer represents the persistence of each

Return type a list of tuple (np.array, int)

See also:

collapse_cluster_graph

hidef.hidef_finder.**output_edges**(*wv*, *names*, *out*, *leaf=False*, *original_cluster_names=None*)
Output hierarchy in the DDOT format; wst column is parents and 2nd column is children Note this output is the 'forward' state.

Parameters

- **wv** (weaver.Weaver) –
- **names** (*list of string*) – a list of ordered gene names
- **out** (*string*) – prefix of the output file
- **leaf** (*bool*) – if True, then write genes into the result
- **original_cluster_names** (*list*) – if not None, do not use the weaver renames, but use a list of specified names; should be equal to the number of clusters in "wv"

hidef.hidef_finder.**output_gml**(*out*)
write a GML file for the hierarchy.

Parameters **out** (*string*) – prefix of the output file

hidef.hidef_finder.**output_nodes**(*wv*, *names*, *out*, *extra_data=None*, *original_cluster_names=None*)
Write a .nodes file according to the weaver result. Four columns are: community names, sizes, member genes, and persistence

Parameters

- **wv** (weaver.Weaver) –
- **names** (*list of string*) – a list of ordered gene names
- **out** (*string*) – prefix of the output file
- **extra_data** (*list of int*) – a list of numbers to show on the last column of the output file
- **original_cluster_names** (*list*) – if not None, do not use the weaver renames, but use a list of specified names; should be equal to the number of clusters in "wv"

hidef.hidef_finder.**partition_to_membership_matrix**(*partition*, *minsize=4*)

Parameters

- **partition** (*class partition in the louvain-igraph package*) –

- **minsize** (*int*) – minimum size of clusters; smaller clusters will be deleted afterwards

Returns **C** – a matrix recording the membership of each cluster

Return type scipy.sparse.csr_matrix

hidef.hidef_finder.**run**(*Gs*, *jaccard=0.75*, *sample=0.9*, *minres=0.01*, *maxres=10*, *alg='leiden'*, *maxn=None*, *density=0.1*, *neighbors=10*, *numthreads=2*, *layer_weights=None*)

Main function to run the Finder program

Parameters

- **Gs** (*a list of igraph.Graph*) – input network(s)
- **jaccard** (*float*) – use (0.5-1.0); a cutoff to call two clusters similar; the 'tau' parameter in paper
- **sample** (*float*) – parameter to perturb input network in each run by deleting edges; lower values delete more
- **minres** (*float*) – minimum resolution parameter
- **maxres** (*float*) – maximum resolution parameter
- **maxn** (*float*) – will explore resolution parameter until cluster number is similar to this number; will override 'maxres'
- **alg** (*str*) – can choose between 'louvain' or 'leiden'
- **density** (*float*) – inversed density of sampling resolution parameter. Use a smaller value to increase sample density (will increase running time)
- **neighbors** (*int*) – also affect sampling density; a larger value may have additional benefits of stabilizing clustering results
- **bisect** (*deprecated*) –
- **numthreads** (*int*) – Number of threads to run in parallel. Default is set to number of cores.

hidef.hidef_finder.**run_alg**(*Gs*, *alg*, *gamma=1.0*, *sample=1.0*, *layer_weights=None*)

Run community detection algorithm with a resolution parameter. Right now only use RB in Louvain/Leiden

Parameters

- **Gs** (*a list of igraph.Graph*) –
- **alg** (*str*) – choose between 'louvain' and 'leiden'
- **gamma** (*float*) – resolution parameter
- **sample** (*if smaller than 1, randomly delete a fraction of edges each time*) –
- **layer_weights** (*a list of float*) – specifying layer weights in the multilayer setting

Returns **C** – a matrix recording the membership of each cluster

Return type scipy.sparse.csr_matrix

hidef.hidef_finder.**update_resolution_graph**(*G*, *new_resolution*, *neighborhood_size*, *neighbor_density_threshold*)

Update the "resolution graph", which connect resolutions that are close enough

Parameters

- **G** (*nx.Graph*) – the "resolution graph"
- **new_resolution** (*float*) – the resolution just visited by the CD algorithm

- **neighborhood_size** (*float*) – if two resolutions (log-scale) differs smaller than this value, they are called 'neighbors'
- **neighbor_density_threshold** (*int*) – if a resolution has neighbors more than this number, it is called "padded". No more sampling will happen between two padded resolutions

weaver module

class hidef.weaver.**Weaver**

Class for constructing a hierarchical representation of a graph based on (a list of) input partitions.

property assignment

assignment matrix

delete_nodes(*nodes*, *relabel=False*)

Delete some nodes from the hierarchy. This approach can be used to delete those with low persistence and rebuild a simpler hierarchy.

Parameters

- **nodes** (*a list of string*) – names of the cluster to delete.
- **relabel** (*bool (default = False)*) – if True, rename nodes. Setting to False may be easier to track the cluster identities.

depth_cluster(*depth*, *flat=True*)

Recovers the partition at specified depth.

Returns **H**

Return type a Numpy array of labels for all the terminal nodes.

level_cluster(*level*, *flat=True*)

Recovers the partition that specified by the index based on the hierarchy.

Returns **H**

Return type a Numpy array of labels for all the terminal nodes.

node_cluster(*node*, *out=None*)

Recovers the cluster represented by a node in the hierarchy.

Returns **H**

Return type a Numpy array of labels for all the terminal nodes.

pick(*top*)

Picks top x percent edges. Alternative edges are ranked based on the number of overlap terminal nodes between the child and the parent. This is the second step of weave(). Subclasses can override this function to achieve different results.

Parameters **top** (*int or float (0 ~ 100, default=100)*) – top x percent (alternative) edges to be kept in the hierarchy. This parameter controls the number of parents each node has based on a global ranking of the edges. Note that if top=0 then each node will only have exactly one parent (except for the root which has none).

Returns

Return type networkx.DiGraph

show(***kwargs*)

Visualize the hierarchy using networkx/graphviz hierarchical layouts.

See also:

show_hierarchy

property terminals
terminals nodes

weave(*partitions*, *terminals=None*, *boolean=True*, *levels=False*, ***kwargs*)
Finds a directed acyclic graph that represents a hierarchy recovered from partitions.

Parameters

- **partitions** (*positional argument*) – a list of different partitions of the graph. Each item in the list should be an array (Numpy array or list) of partition labels for the nodes. A root partition (where all nodes belong to one cluster) and a terminal partition (where all nodes belong to their own cluster) will automatically added later.
- **terminals** (*keyword argument, optional (default=None)*) – a list of names for the graph nodes. If none is provided, an integer will be assigned to each node.
- **levels** (*keyword argument, optional (default=False)*) – whether assume the partitions are provided in some order. If set to True, the algorithm will only find the parents for a node from upper levels. The levels are assumed to be arranged in an ascending order, e.g. partitions[0] is the highest level and partitions[-1] the lowest.
- **boolean** (*keyword argument, optional (default=False)*) – whether the partition labels should be considered as boolean. If set to True, only the clusters labelled as True will be considered as a parent in the hierarchy.
- **merge** (*keyword argument, optional (default=False)*) – whether merge similar clusters. if one cluster is contained in another cluster (determined by "cutoff" oarameter) and vice versa, these two clusters deemed to be very similar. if set to true, such clusters groups will be merged into one (take union)
- **top** (*keyword argument (0 ~ 100, default=100)*) – top x percent (alternative) edges to be kept in the hierarchy. This parameter controls the number of parents each node has based on a global ranking of the edges. Note that if top=0 then each node will only have exactly one parent (except for the root which has none).
- **cutoff** (*keyword argument (0.5 ~ 1.0, default=0.75)*) – containment index cutoff for claiming parenthood.

Returns **T**

Return type networkx.DiGraph

write(*filename*, *format='ddot'*)
Writes the hierarchy to a text file.

Parameters

- **filename** (*str*) – the path and name of the output file.
- **format** (*str*) – output format. Available options are "ddot".

hidef.weaver.**prune**(*T*)
Removes the nodes with only one child and the nodes that have no terminal nodes (e.g. genes) as descendants. (This basically removes identical clusters)

Parameters **T** (*a weaver object*) –

hidef.weaver.**show_hierarchy**(*T*, ***kwargs*)
Visualizes the hierarchy in notebook

utils module

hidef.utils.**containment_indices_boolean**(*A*, *B*)
Calculate a matrix of containment index for two lists of clusters

Parameters

- **matA** (*2D np.array*) – axis 0 for clusters, axis 1 for nodes in network
- **matB** (*2D np.array*) –

Returns **CI**

Return type 2D np.array

hidef.utils.**data2graph**(*datafile*, *outfile=None*, *k=15*, *snn=- 1*, *mydist='cosine'*)
take a dataframe [n_samples x n_features] as input, and output knn or snn graph

Parameters

- **datafile** (*str*) – input tsv file
- **outfile** (*str*) – file name of output edge list
- **k** (*int*) – number of neighbors for each sample
- **snn** (*float*) – a threshold of Jaccard index if going to calculate shared nearest neighbor graph
- **mydist** (*string or callable*) – distance metric to calculate neighbors

Returns **idx** – the indices of entries of the output matrix

Return type tuple of two numpy.array

hidef.utils.**jaccard_matrix**(*matA*, *matB*, *threshold=0.75*, *return_mat=False*)
Calculate jaccard matrix between all pairs between two sets of clusters.

Parameters

- **matA** (*2D array or scipy.sparse.csr_matrix*) – axis 0 for clusters, axis 1 for nodes in network
- **matB** (*2D array or scipy.sparse.csr_matrix*) –
- **threshold** – a similarity cutoff for Jaccard index
- **return_mat** (*bool*) – set to true will also return the full pairwise Jaccard matrix

Returns

- **index** (*(np.array, np.array)*) – two sets of indices; the cluster pairs implied by those indices satisfied threshold
- **jac** (*np.array*) – a full matrix of pairwise Jaccard indices

hidef.utils.**network_perturb**(*G*, *sample=0.8*)
perturb the network by randomly deleting some edges

Parameters

- **G** (*input network*) –
- **sample** (*the fraction of edges to retain*) –

Returns

Return type return: the perturbed graph

hidef.utils.**node2mat**(*f*, *g2ind*, *format='node'*, *has_persistence=False*)
convert a text file to binary matrix (input of weaver)

Parameters

- **f** (*str*) – the input TSV file
- **g2ind** (*dict*) – a dictionary to index genes
- **format** (*str*) – accepted values are 'node' or 'clixo'
- **has_persistence** – if True, the input file has an extra column, here usually the persistence

Returns **data**

Return type a dictionary, contain the field 'cluster', 'name', and could contain 'extra.data'

CHAPTER

TWO

REFERENCES

PYTHON MODULE INDEX

h

INDEX