DiffuPy Documentation

Release 0.0.5-dev

Josep Marín-Llaó, Sergi Picart Armada, Daniel Domingo-Fernánde

May 13, 2020

CONTENTS:

DiffuPy is a generalizable Python implementation of the numerous label propagation algorithms inspired by the diffuStats R package[1]. DiffuPy supports generic graph formats such as JSON, CSV, GraphML, or GML.

Installation is as easy as getting the code from PyPI with `python3 -m pip install diffupy`. See the *installation* documentation.

See also:

- Documented on Read the Docs
- Versioned on GitHub
- Tested on Travis CI
- Distributed by PyPI

[1] Picart-Armada, S., *et al.* (2017). Null diffusion-based enrichment for metabolomics data. *PloS one* 12.12.

CHAPTER

ONE

INSTALLATION

The latest stable code can be installed from PyPI with:

```
$ python3 -m pip install diffupy
```

The most recent code can be installed from the source on GitHub with:

```
$ python3 -m pip install git+https://github.com/multipaths/DiffuPy.git
```

For developers, the repository can be cloned from GitHub and installed in editable mode with:

```
$ git clone https://github.com/multipaths/DiffuPy.git
$ cd diffupy
$ python3 -m pip install -e .
```

CHAPTER

TWO

FIRST STEPS

The first step before running diffusion algorithms on your network using DiffuPy is to learn about the graph and data formats are supported. Next, you can find samples of input datasets and networks to run diffusion methods over.

2.1 Input Data

You can submit your dataset in any of the following formats:

- CSV (.csv)
- TSV (.tsv)

Please ensure that the dataset minimally has a column 'Node' containing node IDs. You can also optionally add the following columns to your dataset:

- NodeType
- LogFC*[0]
- p-value

2.1.1 Input dataset examples

DiffuPath accepts several input formats which can be codified in different ways. See the diffusion scores summary for more details.

1. You can provide a dataset with a column 'Node' containing node IDs.

Node
A
B
C
D

2. You can also provide a dataset with a column 'Node' containing node IDs as well as a column 'NodeType', indicating the entity type of the node to run diffusion by entity type.

[0] Log_2 fold change

Node	NodeType
A	Gene
B	Gene
C	Metabolite
D	Gene

3. You can also choose to provide a dataset with a column 'Node' containing node IDs as well as a column 'logFC' with their logFC. You may also add a 'NodeType' column to run diffusion by entity type.

Node	LogFC
A	4
B	-1
C	1.5
D	3

4. Finally, you can provide a dataset with a column 'Node' containing node IDs, a column 'logFC' with their logFC and a column 'p-value' with adjusted p-values. You may also add a 'NodeType' column to run diffusion by entity type.

Node	LogFC	p-value
A	4	0.03
B	-1	0.05
C	1.5	0.001
D	3	0.07

See the sample datasets directory for example files.

2.2 Networks

If you would like to submit your own networks, please ensure they are in one of the following formats:

- BEL (.bel)
- CSV (.csv)
- Edge list (.lst)
- GML (.gml or .xml)
- GraphML (.graphml or .xml)
- Pickle (.pickle). BELGraph object from PyBEL 0.13.2
- TSV (.tsv)
- TXT (.txt)

Minimally, please ensure each of the following columns are included in the network file you submit:

- Source
- Target

Optionally, you can choose to add a third column, "Relation" in your network (as in the example below). If the relation between the **Source** and **Target** nodes is omitted, and/or if the directionality is ambiguous, either node can be assigned as the **Source** or **Target**.

2.2.1 Custom-network example

Source	Target	Relation
A	B	Increase
B	C	Association
A	D	Association

You can also take a look at our sample networks folder for some examples.

CHAPTER

THREE

COMMAND LINE INTERFACE

DiffuPy Command Line Interface

3.1 diffupy

DiffuPy

```
diffupy [OPTIONS] COMMAND [ARGS]...
```

3.1.1 diffuse

Run a diffusion method over a network or pre-generated kernel.

```
diffupy diffuse [OPTIONS]
```

Options

-i, --input <input>
Input data [required]

-n, --network <network>
Path to the network graph or kernel [required]

-o, --output <output>
Output file

-m, --method <method>
Diffusion method

Options z|ber_s|raw|gm|mc|ber_p|ml

-b, --binarize <binarize>
If logFC provided in dataset, convert logFC to binary (e.g., up-regulated entities to 1, down-regulated to -1). For scoring methods that accept quantitative values (i.e., raw & z), node labels can also be codified with LogFC (in this case, set binarize==False).

-t, --threshold <threshold>
Codify node labels by applying a threshold to logFC in input.

-a, --absolute_value <absolute_value>
Codify node labels by applying threshold to | logFC | in input. If absolute_value is set to False,node labels will be signed.

-p, --p_value <p_value>
Statistical significance (p-value). [default: 0.05]

-f, --output_format <output_format>
Statistical significance (p-value). [default: csv]

3.1.2 kernel

Generate a kernel for a given network.

```
diffupy kernel [OPTIONS]
```

Options

-n, --network <network>
Input network [required]

-o, --output <output>
Output path to store the generated kernel pickle [default: /home/docs/.diffupy/output]

-l, --log
Activate debug mode

CHAPTER

FOUR

KERNEL

Compute graph kernels.

diffupy.kernels.**compute_time_kernel**(*graph: networkx.classes.graph.Graph*, *normalized: bool = False*) → *diffupy.matrix.Matrix*

Compute the commute-time kernel, which is the expected time of going back and forth between a couple of nodes.

If the network is connected, then the commuted time kernel will be totally dense, therefore reflecting global properties of the network. For further details, see [Yen, 2007]. This kernel can be computed using both the unnormalised and normalised graph Laplacian.

Parameters

- **graph** – A graph
- **normalized** – Indicates if Laplacian transformation is normalized or not.

Returns Laplacian representation of the graph.

diffupy.kernels.**diffusion_kernel**(*graph: networkx.classes.graph.Graph*, *sigma2: float = 1*, *normalized: bool = True*) → *diffupy.matrix.Matrix*

Compute the classical diffusion kernel that involves matrix exponentiation.

It has a "bandwidth" parameter sigma^2 that controls the extent of the spreading. Quoting [Smola, 2003]: k(x1,x2) can be visualized as the quantity of some substance that would accumulate at vertex x2 after a given amount of time if we injected the substance at vertex x1 and let it diffuse through the graph along the edges.

This kernel can be computed using both the unnormalised and normalised graph Laplacian. :param graph: A graph :param sigma2: Controls the extent of the spreading. :param normalized: Indicates if Laplacian transformation is normalized or not. :return: Laplacian representation of the graph

diffupy.kernels.**inverse_cosine_kernel**(*graph: networkx.classes.graph.Graph*) → *diffupy.matrix.Matrix*

Compute the inverse cosine kernel, which is based on a cosine transform on the spectrum of the normalized LM.

Quoting [Smola, 2003]: the inverse cosine kernel treats lower complexity functions almost equally, with a significant reduction in the upper end of the spectrum.

This kernel is computed using the normalised graph Laplacian.

Parameters **graph** – A graph

Returns Laplacian representation of the graph

diffupy.kernels.**p_step_kernel**(*graph: networkx.classes.graph.Graph*, *a: int = 2*, *p: int = 5*) → *diffupy.matrix.Matrix*

Compute the inverse cosine kernel, which is based on a cosine transform on the spectrum of the normalized LM.

This kernel is more focused on local properties of the nodes, because random walks are limited in terms of length. Therefore, if p is small, only a fraction of the values k(x1,x2) will be non-null if the network is sparse

[Smola, 2003]. The parameter a is a regularising term that is summed to the spectrum of the normalised Laplacian matrix, and has to be 2 or greater. The p-step kernels can be cheaper to compute and have been successful in biological tasks, see the benchmark in [Valentini, 2014].

Parameters

- **graph** – A graph
- **a** – regularising summed to the spectrum. Spectrum of the normalised Laplacian matrix.
- **p** – p-step kernels can be cheaper to compute and have been successful in biological tasks.

Returns Laplacian representation of the graph.

diffupy.kernels.**regularised_laplacian_kernel**(*graph: networkx.classes.graph.Graph*, *sigma2: float = 1*, *add_diag: int = 1*, *normalized: bool = False*) → *diffupy.matrix.Matrix*

Compute the regularised Laplacian kernel, which is a standard in biological networks.

The regularised Laplacian kernel arises in numerous situations, such as the finite difference formulation of the diffusion equation and in Gaussian process estimation. Sticking to the heat diffusion model, this function allows to control the constant terms summed to the diagonal through add_diag, i.e. the strength of the leaking in each node. If a node has diagonal term of 0, it is not allowed to disperse heat. The larger the diagonal term of a node, the stronger the first order heat dispersion in it, provided that it is positive. Every connected component in the graph should be able to disperse heat, i.e. have at least a node i with add_diag[i] > 0. If this is not the case, the result diverges. More details on the parameters can be found in [Smola, 2003]. This kernel can be computed using both the unnormalised and normalised graph Laplacian.

Parameters

- **graph** – A graph
- **a** – regularising summed to the spectrum. Spectrum of the normalised Laplacian matrix.
- **p** – p-step kernels can be cheaper to compute and have been successful in biological tasks.

Returns Laplacian representation of the graph.

CHAPTER

FIVE

DIFFUSION

The methods in this modules manage the treatment of the different score diffusion methods applied to/from a path set of labels/scores of/on a certain network (as a graph format or a graph kernel matrix stemming from a graph).

Diffusion methods procedures provided in this package differ on: (a) How to distinguish positives, negatives and unlabelled examples. (b) Their statistical normalisation.

Input scores can be specified in three formats: 1. A named numeric vector, whereas if several of these vectors that share the node names need to be smoothed. 2. A column-wise matrix. However, if the unlabelled entities are not the same from one case to another. 2. A named list of such score matrices can be passed to this function. The path format will be kept in the output.

If the path labels are not quantitative, i.e. positive(1), negative(0) and possibly unlabelled, all the scores raw, gm, ml, z, mc, ber_s, ber_p can be used.

5.1 Methods

5.1.1 Methods without statistical normalisation

- **raw**: positive nodes introduce unitary flow {y_raw[i] = 1} to the network, whereas either negative and unlabelled nodes introduce null diffusion {y_raw[j] = 0}.[1]. They are computed as: f_{raw} = K · y_{raw}. Where K is a graph kernel, see *kernels*. These scores treat negative and unlabelled nodes equivalently.
- **ml**: Same as raw, but negative nodes introduce a negative unit of flow. Therefore not equivalent to unlabelled nodes[2].
- **gl**: Same as ml, but the unlabelled nodes are assigned a (generally non-null) bias term based on the total number of positives, negatives and unlabelled nodes[3].
- **ber_s**: A quantification of the relative change in the node score before and after the network smoothing. The score for a particular node i can be written as f_{ber_s}[i] = f_{raw}[i] / (y_{raw}[i] + eps). Where eps is a parameter controlling the importance of the relative change.

[1] Vandin, F., *et al.* (2010). Algorithms for detecting significantly mutated pathways in cancer. Lecture Notes in Computer Science. 6044, 506–521.

[2] Zoidi, O., *et al.* (2015). Graph-based label propagation in digital media: A review. ACM Computing Surveys (CSUR), 47(3), 1-35.

[3] Mostafavi, S., *et al.* (2008). Genemania: a real-time multiple association network integration algorithm for predicting gene function.Genome Biology. (9), S4.

5.1.2 Methods with statistical normalisation

- **z**: a parametric alternative to the raw score of node is subtracted its mean value and divided by its standard deviation. Differential trait of this package. The statistical moments have a closed analytical form and are inspired in[4].
- **mc**: the score of node code {i} is based on its empirical p-value, computed by permuting the path {n.perm} times. It is roughly the proportion of path permutations that led to a diffusion score as high or higher than the original diffusion score.
- **ber_p**: used in[5], this score combines raw and mc, in order to take into account both the magnitude of the {raw} scores and the effect of the network topology: this is a quantification of the relative change in the node score before and after the network smoothing.

5.2 Summary tables

5.2.1 Methods without statistical normalization

Scores	y+	y-	yn	Normalized	Stochastic	Quantitative	Reference
raw	1	0	0	No	No.	Yes	[1]
ml	1	-1	0	No	No	No	[6]
gm	1	-1	k	No	No	No	[3]
ber_s	1	0	0	No	No	Yes	[5]

5.2.2 Methods with statistical normalization

Scores	y+	y-	yn	Normalized	Stochastic	Quantitative	Reference
ber_p	1	0	0*	Yes	Yes	Yes	[5]
mc	1	0	0*	Yes	Yes	Yes	[5]
z	1	0	0*	Yes	No	Yes	[4]

This module provides a generalized function as an interface to interact with the different diffusion methods.

`diffupy.diffuse.`**`diffuse`**(*input_scores*, *method: str = 'raw'*, *graph: networkx.classes.graph.Graph = None*, ***kwargs*) → *diffupy.matrix.Matrix*

Run diffusion on a network given an input and a diffusion method.

Parameters

- **`input_scores`** – score collection, supplied as n-dimensional array. Could be 1-dimensional (Vector) or n-dimensional (Matrix).
- **`method`** – Elected method ["raw", "ml", "gm", "ber_s", "ber_p", "mc", "z"]
- **`graph`** – A network as a graph. It could be optional if a Kernel is provided
- **`kwargs`** – Optional arguments: - k: a kernel [matrix] stemming from a graph, thus sparing the graph transformation process - Other arguments which would differ depending on the chosen method

[4] Harchaoui, Z., *et al.* (2013). Kernel-based methods for hypothesis testing: a unified view. IEEE Signal Processing Magazine. (30), 87–97.
[5] Bersanelli, M. *et al.* (2016). Network diffusion-based analysis of high-throughput data for the detection of differentially enriched modules. Scientific Reports. (6), 34841.
[6] Tsuda, K., *et al.* (2005). Fast protein classification with multiple networks. Bioinformatics, (21), 59–65

Returns The diffused scores within the matrix transformation of the network, with the diffusion operation [k x input_vector] performed

Diffuse scores on a network.

diffupy.diffuse_raw.**diffuse_raw**(*graph: networkx.classes.graph.Graph*, *scores: diffupy.matrix.Matrix*, *z: bool = False*, *k: diffupy.matrix.Matrix = None*) → *diffupy.matrix.Matrix*

Compute the score diffusion procedure, given an initial state as a set of scores and a network to diffuse over.

Parameters

- **graph** – background network
- **scores** – array of score matrices. For a single path with a single background, supply a list with a vector col
- **z** – bool to indicate if z-scores be computed instead of raw scores
- **k** – optional precomputed diffusion kernel matrix

Returns A list of scores, with the same length and dimensions as scores

5.3 References

CHAPTER

SIX

CONSTANTS

Constants of diffupy.

diffupy.constants.**DEFAULT_DIFFUPY_DIR = '/home/docs/.diffupy'**
Default DiffuPy directory

diffupy.constants.**OUTPUT = '/home/docs/.diffupy/output'**
Default DiffuPy output directory

diffupy.constants.**ensure_output_dirs**()
Ensure that the output directories exists.

diffupy.constants.**EMOJI = ''**
Available diffusion methods

diffupy.constants.**RAW = 'raw'**
raw

diffupy.constants.**ML = 'ml'**
ml

diffupy.constants.**GM = 'gm'**
gm

diffupy.constants.**MC = 'mc'**
mc

diffupy.constants.**Z = 'z'**
z

diffupy.constants.**BER_S = 'ber_s'**
ber_s

diffupy.constants.**BER_P = 'ber_p'**
ber p

diffupy.constants.**METHODS = {'ber_p', 'ber_s', 'gm', 'mc', 'ml', 'raw', 'z'}**
Available formats

diffupy.constants.**CSV = 'csv'**
csv

diffupy.constants.**XLS = 'xls'**
xml

diffupy.constants.**XLSX = 'xlsx'**
xmls

diffupy.constants.**TSV = 'tsv'**
tsv

diffupy.constants.**GRAPHML = 'graphml'**
graphML

diffupy.constants.**BEL = 'bel'**
bel

diffupy.constants.**JSON = 'json'**
node link json

diffupy.constants.**PICKLE = 'pickle'**
pickle

diffupy.constants.**GML = 'gml'**
gml

diffupy.constants.**EDGE_LIST = '.lst'**
edge list

diffupy.constants.**GRAPH_FORMATS = ('csv', 'tsv', 'graphml', 'bel', 'json', 'pickle')**
Available graph formats

diffupy.constants.**KERNEL_FORMATS = ('csv', 'tsv', 'json', 'pickle')**
Available kernel formats

diffupy.constants.**FORMAT_SEPARATOR_MAPPING = {'csv': ',', 'tsv': '\t'}**
Optional parameters

diffupy.constants.**THRESHOLD = 'threshold'**
Expression value threshold

diffupy.constants.**ABSOLUTE_VALUE_EXP = 'absolute_value'**
Acceptable column names of user submitted network

diffupy.constants.**SOURCE = 'Source'**
Column name for source node

diffupy.constants.**TARGET = 'Target'**
Column name for target node

diffupy.constants.**RELATION = 'Relation '**
Dataset column names

diffupy.constants.**NODE = 'Node'**
Node name

diffupy.constants.**NODE_TYPE = 'NodeType'**
Node type

diffupy.constants.**SCORE = 'Score'**
Unspecified score type

diffupy.constants.**LOG_FC = 'LogFC'**
Log2 fold change (logFC)

diffupy.constants.**P_VALUE = 'p-value'**
Statistical significance (p-value)

CHAPTER

SEVEN

MATRIX

Matrix class

Main Matrix Class.

class diffupy.matrix.**Matrix**(*mat=None*, *rows_labels=None*, *cols_labels=None*, *graph=None*, *quadratic=False*, *name=''*, *init_value=None*)

Matrix class.

Initialize matrix.

Parameters

- **mat** – matrix initialization
- **rows_labels** –
- **cols_labels** – column labels
- **graph** – graph
- **quadratic** – quadratic
- **name** – name
- **init_value** – value to be initialized (int) or list of values from labels

validate_labels()

Sanity function to check the dimensionality of the Matrix.

update_ix_mappings()

Update the index-label mapping.

validate_labels_and_update_ix_mappings()

Update function, called when the Matrix mutates, combining the two previous functionalities.

property cols_labels

Return a copy of Matrix Object.

property rows_labels_ix_mapping

Set row labels to ix.

property cols_labels_ix_mapping

Set column labels to ix.

property rows_idx_scores_mapping

Set mapping indexes to scores.

property cols_idx_scores_mapping

Set mapping indexes to scores.

get_row_from_label(*label*)
Get row from labels.

set_row_from_label(*label*, *x*)
Set row from label.

delete_row_from_label(*label*)
Set row from label.

get_col_from_label(*label*)
Get col from labels.

delete_col_from_label(*label*)
Set col from label.

set_cell_from_labels(*row_label*, *col_label*, *x*)
Set cell from labels.

get_cell_from_labels(*row_label*, *col_label*)
Get cell from labels.

row_bind(*rows=None*, *rows_labels=None*, *matrix=None*)
Return a copy of Matrix Object.

col_bind(*cols=None*, *cols_labels=None*, *matrix=None*)
Return a copy of Matrix Object.

match_rows(*reference_matrix*)
Match method to set rows labels as reference matrix.

match_cols(*reference_matrix*)
Match method to set cols labels as reference matrix.

match_mat(*reference_matrix*, *match_quadratic=None*)
Match method to set axis labels as reference matrix.

match_missing_rows(*reference_labels*, *missing_fill=0*)
Match method to set missing rows labels from reference labels with the missing_fill value.

match_delete_rows(*reference_labels*)
Match method to set missing rows labels from reference labels with the missing_fill value.

match_missing_cols(*reference_labels*, *missing_fill*)
Match method to set missing cols labels from reference labels with the missing_fill value.

order_rows(*reverse=True*, *col_ref_idx=None*)
Order matrix rows by cell values.

to_dict(*ordered=True*)
Export/convert matrix as a dictionary data structure.

to_df(*ordered=True*)
Export matrix as a data frame using the headers (row_labels, cols_labels) of the Matrix class.

to_csv(*path*, *file_name='_export.csv'*, *index=False*, *ordered=True*)
Export matrix to csv file using the headers (row_labels, cols_labels) of the Matrix class.

to_nx_graph()
Export matrix as a Graph using the headers (row_labels, cols_labels) of the Matrix class.

class diffupy.matrix.**LaplacianMatrix**(*graph*, *normalized=False*, *node_argument='name'*, *name=''*)
Laplacian matrix class.

Initialize laplacian.

CHAPTER

EIGHT

REFERENCES

CHAPTER

NINE

DISCLAIMER

DiffuPy is a scientific software that has been developed in an academic capacity, and thus comes with no warranty or guarantee of maintenance, support, or back-up of data.

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

C

N

O

P

R

S

T

U

V

X

Z