Bigraph

Release rc1

Soran Ghadri

May 22, 2022

CONTENTS:

CHAPTER

ONE

INTRODUCTION

`bigraph` is a high-level OO Python package which aims to provide an easy and intuitive way of interacting with bipartite networks. In essence, this package is an extension of the `NetworkX` package (see here)

The aim here was to define an easy-to-use package which would allow developers to perform `link pridiction`, `evaluation`, and `visualization` of the results.

The current implementation has been developed and tested in Python 3.6+, but should work with Python 2.7+ (maybe with minor modifications in terms of printing and error handling) and most Debian based OSs.

1.1 Motivation

As a developer in the field of machine learning, I found it pretty hard to identify a Python package which would enable one to perform link pridiction algorithms over bipartite networks.

This package is intended to provide a quick, as well as (hopefully) easy to undestand, way of dealing-with-bipartite-networks algorithm up and running, for all those out there, who, like myself, are hands-on learners and are eager to get their hands dirty from early on.

1.2 Limitations

- Algorithms are quite a few, and thus have to be piled with new ones.
- Evaluations used here are techniques used for unsupervised methods, however they may have to be modified.
- There are more limitations for sure if you happened to encounter any issue, you are welcomed to make contributions or open and issue on github.

CHAPTER

TWO

BIGRAPH PACKAGE

2.1 Subpackages

2.1.1 bigraph.evaluation package

2.1.1.1 Submodules

2.1.1.2 bigraph.evaluation.evaluation module

`bigraph.evaluation.evaluation.evaluate`(*graph: networkx.classes.graph.Graph*, *k: int = 2*, *method: str = 'all'*)

Evaluation interface for evaluating algorithms

Parameters

- **graph** – Networkx bipartite graph
- **k** – Number of folds (used in KFold)
- **method** – Algorithm name

Returns Calculated metrics: overall_precision, overall_auc, fpr_algo, tpr_algo

2.1.1.3 Module contents

2.1.2 bigraph.preprocessing package

2.1.2.1 Submodules

2.1.2.2 bigraph.preprocessing.get_adjacents module

class `bigraph.preprocessing.get_adjacents.GetAdjacents`

Bases: `object`

2.1.2.3 bigraph.preprocessing.import_files module

class bigraph.preprocessing.import_files.**ImportFiles**

Bases: object

import_files(*edge_csv: str = './inputs/neighbour_matrix.csv', label_id: str = './inputs/id_labels.csv', sep: str = ',', *args, **kwargs*) → dict

Import links and labels from csv files

Parameters

- **edge_csv** – A CSV file containing edge data on each line: 0,4,2 that are u,v,w which stand for node 1, node 2, weight respectively
- **label_id** – A CSV file containing labels for nodes: 10,node_name which are ID,label for each node
- **sep** – A separator that is the boundary for distinct entities

Returns links and label dataframes

2.1.2.4 bigraph.preprocessing.make_graph module

class bigraph.preprocessing.make_graph.**MakeGraph**

Bases: object

make_graph(*dataframe: dict, left_bipartite: str = 'left_side', right_bipartite: str = 'right_side'*)

Make a graph based on the input dataframe

Parameters

- **left_bipartite** – Left part of the graph
- **right_bipartite** – Right part of the graph
- **dataframe** – Bipartite graph dataframe

Returns Generated networkx graph

2.1.2.5 bigraph.preprocessing.pd_to_list module

2.1.2.6 Module contents

class bigraph.preprocessing.**GetAdjacents**

Bases: object

class bigraph.preprocessing.**ImportFiles**

Bases: object

import_files(*edge_csv: str = './inputs/neighbour_matrix.csv', label_id: str = './inputs/id_labels.csv', sep: str = ',', *args, **kwargs*) → dict

Import links and labels from csv files

Parameters

- **edge_csv** – A CSV file containing edge data on each line: 0,4,2 that are u,v,w which stand for node 1, node 2, weight respectively

- **label_id** – A CSV file containing labels for nodes: 10,node_name which are ID,label for each node
- **sep** – A separator that is the boundary for distinct entities

Returns links and label dataframes

class bigraph.preprocessing.**MakeGraph**

Bases: object

make_graph(*dataframe: dict*, *left_bipartite: str = 'left_side'*, *right_bipartite: str = 'right_side'*)

Make a graph based on the input dataframe

Parameters

- **left_bipartite** – Left part of the graph
- **right_bipartite** – Right part of the graph
- **dataframe** – Bipartite graph dataframe

Returns Generated networkx graph

2.2 Submodules

2.3 bigraph.algorithms module

class bigraph.algorithms.**Algorithms**

Bases: object

static adamic_adar(*set_one: list*, *set_two: list*, *graph*) → float

Calculate Adamic Adar score for input lists

Parameters

- **set_one** – A list of graph nodes -> part one
- **set_two** – A list of graph nodes -> part two
- **graph** – NetworkX bipartite graph

Returns Adamic Adar score

static common_neighbors(*set_one: list*, *set_two: list*) → int

Calculate Common neighbors score for input lists

Parameters

- **set_one** – A list of graph nodes -> part one
- **set_two** – A list of graph nodes -> part two

Returns Common neighbours score

static jaccard(*set_one: list*, *set_two: list*) → float

Calculate Jaccard score for input lists

Parameters

- **set_one** – A list of graph nodes -> part one
- **set_two** – A list of graph nodes -> part two

Returns Jaccard score

static katz_similarity(*node_i: int*, *node_j: int*, *graph*) → float

Calculate Katz score for input nodes

Parameters

- **node_i** – Starting node
- **node_j** – Destination node
- **graph** – NetworkX bipartite graph

Returns Katz similarity score

static preferential_attachment(*set_one: list*, *set_two: list*) → int

Calculate Preferential attachment score for input lists

Parameters

- **set_one** – A list of graph nodes -> part one
- **set_two** – A list of graph nodes -> part two

Returns Preferential attachment score

2.4 bigraph.bigraph module

class bigraph.bigraph.**BiGraph**

Bases: *bigraph.algorithms.Algorithms*, *bigraph.preprocessing.import_files.ImportFiles*, *bigraph.preprocessing.make_graph.MakeGraph*, *bigraph.preprocessing.get_adjacents.GetAdjacents*

aa_predict() → dict

Compute the Jaccard-Needham dissimilarity between two 1-D arrays.

Returns A dictionary containing the Adamic-adar score for *left_element* and *right_element*.

cn_predict() → dict

Return the common neighbors of two nodes in a graph.

Returns A dictionary containing the Common neighbours score for *left_element* and *right_element*.

jc_predict() → dict

Compute the Jaccard-Needham dissimilarity between two 1-D arrays.

Returns A dictionary containing the Jaccard distance between vectors *left_element* and *right_element*.

katz_predict(*df_nodes: dict*) → dict

Compute the Katz similarity score of all node pairs.

Parameters **df_nodes** – Graph nodes

Returns A dictionary containing the Preferential attachment score for *left_element* and *right_element*.

pa_predict() → dict

Compute the preferential attachment score of all node pairs.

Returns A dictionary containing the Preferential attachment score for *left_element* and *right_element*.

2.5 Module contents

A package for link prediction in bipartite networks.

class bigraph.**BiGraph**

Bases: *bigraph.algorithms.Algorithms*, *bigraph.preprocessing.import_files.ImportFiles*, *bigraph.preprocessing.make_graph.MakeGraph*, *bigraph.preprocessing.get_adjacents.GetAdjacents*

aa_predict() → dict

Compute the Jaccard-Needham dissimilarity between two 1-D arrays.

Returns A dictionary containing the Adamic-adar score for *left_element* and *right_element*.

cn_predict() → dict

Return the common neighbors of two nodes in a graph.

Returns A dictionary containing the Common neighbours score for *left_element* and *right_element*.

jc_predict() → dict

Compute the Jaccard-Needham dissimilarity between two 1-D arrays.

Returns A dictionary containing the Jaccard distance between vectors *left_element* and *right_element*.

katz_predict(*df_nodes: dict*) → dict

Compute the Katz similarity score of all node pairs.

Parameters **df_nodes** – Graph nodes

Returns A dictionary containing the Preferential attachment score for *left_element* and *right_element*.

pa_predict() → dict

Compute the preferential attachment score of all node pairs.

Returns A dictionary containing the Preferential attachment score for *left_element* and *right_element*.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

P