dynamicgem Documentation

Release 0.0.3

Palash Goyal, Sujit Rokka Chhetri and Arquimedes Canedo

Aug 17, 2019

QUICK START TUTORIAL - EXAMPLES

dynamicgem is an open-source Python library for learning node representations of dynamic graphs. It consists of state-of-the-art algorithms for defining embeddings of nodes whose connections evolve over time. We have implemented various metrics to evaluate the state-of-the-art methods, and examples of evolving networks from various domains. We have easy-to-use functions to call and evaluate the methods and have extensive usage documentation.

Checkout the Github Repository.

CHAPTER

ONE

DEPENDENCIES

You will need following dependencies to be installed for the dynamicgem library:

- sphinx>=2.1.2
- networkx>=2.2
- setuptools>=40.8.0
- matplotlib
- numpy>=1.16.2
- seaborn>=0.9.0
- scikit_learn>=0.20.3
- numpydoc>=0.9.1
- sphinx-gallery>=0.3.1
- sphinx-rtd-theme>=0.4.3
- pytest>=3.6
- tensorflow==1.14.0
- h5py>=2.8.0
- joblib>=0.12.5
- Keras>=2.2.4
- pandas>=0.23.4
- six>=1.11.0

CHAPTER

TWO

INSTALLATION

dynamicegem is available in the PyPi's repository.

Please install Tensorflow **gpu version before installing dynamicgem!** for best performance.

Prepare your environment:

```
$ sudo apt update
$ sudo apt install python3-dev python3-pip
$ sudo pip3 install -U virtualenv
```

Create a virtual environment

If you have tensorflow installed in the root env, do the following:

```
$ virtualenv --system-site-packages -p python3 ./venv
```

If you you want to install tensorflow later, do the following:

```
$ virtualenv -p python3 ./venv
```

Activate the virtual environment using a shell-specific command:

```
$ source ./venv/bin/activate
```

Upgrade pip:

```
$ pip install --upgrade pip
```

If you have not installed tensorflow, or not used –system-site-package option while creating venv, install tensorflow first:

```
(venv) $ pip install tensorflow
```

Install dynamicgem using 'pip':

```
(venv) $ pip install dynamicgem
```

Install stable version directly from github repo:

```
(venv) $ git clone https://github.com/Sujit-O/dynamicgem.git
(venv) $ cd dynamicgem
(venv) $ python setup.py install
```

Install development version directly from github repo:

```
(venv) $ git clone https://github.com/Sujit-O/dynamicgem.git
(venv) $ cd dynamicgem
(venv) $ git checkout development
(venv) $ python setup.py install
```

CHAPTER

THREE

TESTING

After installation, you can use *pytest* to run the test suite from dynamicgem's root directory:

```
pytest
```

CHAPTER

FOUR

GENERAL EXAMPLES

General-purpose and introductory examples for the *dynamicgem* library.

Note: Click *here* to download the full example code

4.1 Example Code for Dynamic Graph Factorization

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

disp_avlbl = True
import os
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib

    matplotlib.use('Agg')
import matplotlib.pyplot as plt

from dynamicgem.embedding.graphFac_dynamic import GraphFactorization
from dynamicgem.visualization import plot_dynamic_sbm_embedding
from dynamicgem.graph_generation import dynamic_SBM_graph


if __name__ == '__main__':
    node_num = 100
    community_num = 2
    node_change_num = 2
    length = 5
    dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_num,␣
↪community_num, length,1,
                                                                            node_change_
↪num)

    dynamic_embeddings = GraphFactorization(16, 10, 10, 5 * 10 ** -2, 1.0, 1.0)
    # pdb.set_trace()
    dynamic_embeddings.learn_embeddings([g[0] for g in dynamic_sbm_series])
```

(continues on next page)

(continued from previous page)

```
    plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding(dynamic_embeddings.get_
↪embeddings(), list(dynamic_sbm_series))
    plt.show()
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.2 Example Code for DynGEM

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

disp_avlbl = True
import os
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
    matplotlib.use('Agg')
import matplotlib.pyplot as plt

from dynamicgem.embedding.dynGEM import DynGEM
from dynamicgem.graph_generation import SBM_graph
from dynamicgem.utils import graph_util, plot_util
from dynamicgem.graph_generation import SBM_graph
from dynamicgem.evaluation import evaluate_graph_reconstruction as gr
from time import time

if __name__ == '__main__':
    my_graph = SBM_graph.SBMGraph(100, 2)
    my_graph.sample_graph()
    node_colors = plot_util.get_node_color(my_graph._node_community)
    t1 = time()
    embedding = DynGEM(d=8, beta=5, alpha=0, nu1=1e-6, nu2=1e-6, K=3,
                       n_units=[64, 16], n_iter=2, xeta=0.01,
                       n_batch=50,
                       modelfile=['./intermediate/enc_model.json',
                                  './intermediate/dec_model.json'],
                       weightfile=['./intermediate/enc_weights.hdf5',
                                   './intermediate/dec_weights.hdf5'])
    embedding.learn_embedding(graph=my_graph._graph, edge_f=None,
                              is_weighted=True, no_python=True)
    print('SDNE:\n\tTraining time: %f' % (time() - t1))
    MAP, prec_curv, err, err_baseline = \
        gr.evaluateStaticGraphReconstruction(
            my_graph._graph,
            embedding,
            embedding.get_embedding(),
            None
```

(continues on next page)

(continued from previous page)

```
        )
    print(MAP)
    print(prec_curv[:10])
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.3 Example Code for Structural Deep Network Embedding

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

disp_avlbl = True
import os
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
    matplotlib.use('Agg')
import matplotlib.pyplot as plt

import numpy as np
import scipy.io as sio
from argparse import ArgumentParser
import time

from dynamicgem.embedding.sdne_dynamic import SDNE
from dynamicgem.utils import graph_util
from dynamicgem.visualization import plot_dynamic_sbm_embedding
from dynamicgem.graph_generation import dynamic_SBM_graph
from dynamicgem.evaluation import evaluate_link_prediction as lp
from dynamicgem.evaluation import visualize_embedding as viz

if __name__ == '__main__':
    parser = ArgumentParser(description='Learns node embeddings for a sequence of␣
↪graph snapshots')
    parser.add_argument('-t', '--testDataType', default='sbm_cd', type=str,help='Type␣
↪of data to test the code')
    args = parser.parse_args()

    if args.testDataType == 'sbm_rp':
        node_num = 10000
        community_num = 500
        node_change_num = 100
        length = 2
        dynamic_sbm_series = dynamic_SBM_graph.get_random_perturbation_series(node_
↪num, community_num, length, node_change_num)
        dynamic_embedding = SDNE(d=100, beta=5, alpha=1e-5, nu1=1e-6, nu2=1e-6, K=3,␣
↪n_units=[500, 300,], rho=0.3, n_iter=30, n_iter_subs=5, xeta=0.01, n_batch=500,␣
↪modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json'],␣
↪weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_weights.hdf5'],␣
↪node_frac=1, n_walks_per_node=10, len_rw=2)
```

(continues on next page)

(continued from previous page)

```
        dynamic_embedding.learn_embeddings([g[0] for g in dynamic_sbm_series], False,␣
↪subsample=False)
        plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding(dynamic_embedding.get_
↪embeddings(), dynamic_sbm_series)
        plt.savefig('result/visualization_sdne_rp.png')
        plt.show()
    elif args.testDataType == 'sbm_cd':
        node_num = 100
        community_num = 2
        node_change_num = 2
        length = 5
        dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num, community_num, length, 1, node_change_num)
        dynamic_embedding = SDNE(d=16, beta=5, alpha=1e-5, nu1=1e-6, nu2=1e-6, K=3, n_
↪units=[500, 300,], rho=0.3, n_iter=2, n_iter_subs=5, xeta=0.01, n_batch=50,␣
↪modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json'],␣
↪weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_weights.hdf5'],␣
↪node_frac=1, n_walks_per_node=10, len_rw=2)
        embs= []
        graphs = [g[0] for g in dynamic_sbm_series]
        for temp_var in range(length):
                emb= dynamic_embedding.learn_embedding(graphs[temp_var])
                embs.append(emb)
        viz.plot_static_sbm_embedding(embs[-4:], list(dynamic_sbm_series)[-4:])
    else:
        dynamic_graph_series = graph_util.loadRealGraphSeries('data/real/hep-th/month_
↪', 1, 5)
        dynamic_embedding = SDNE(d=100, beta=2, alpha=1e-6, nu1=1e-5, nu2=1e-5, K=3,␣
↪n_units=[400, 250,], rho=0.3, n_iter=100, n_iter_subs=30, xeta=0.001, n_batch=500,␣
↪modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json'],␣
↪weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_weights.hdf5'])
        dynamic_embedding.learn_embeddings(dynamic_graph_series, False)
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.4 Example Code for Static AE

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import networkx as nx
disp_avlbl = True
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
    matplotlib.use('Agg')
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt

from argparse import ArgumentParser
import sys
import pdb
from joblib import Parallel, delayed
import operator
from time import time

from dynamicgem.embedding.ae_static import AE
from dynamicgem.utils import graph_util, dataprep_util
from dynamicgem.evaluation import visualize_embedding as viz
from dynamicgem.utils.sdne_utils import *
from dynamicgem.evaluation import evaluate_link_prediction as lp
from dynamicgem.graph_generation import dynamic_SBM_graph



if __name__ == '__main__':

    parser = ArgumentParser(description='Learns static node embeddings')
    parser.add_argument('-t', '--testDataType',
                        default='sbm_cd',
                        type=str,
                        help='Type of data to test the code')
    parser.add_argument('-l', '--timelength',
                        default=5,
                        type=int,
                        help='Number of time series graph to generate')
    parser.add_argument('-nm', '--nodemigration',
                        default=5,
                        type=int,
                        help='number of nodes to migrate')
    parser.add_argument('-iter', '--epochs',
                        default=2,
                        type=int,
                        help='number of epochs')
    parser.add_argument('-emb', '--embeddimension',
                        default=16,
                        type=int,
                        help='embedding dimension')
    parser.add_argument('-sm', '--samples',
                        default=10,
                        type=int,
                        help='samples for test data')
    parser.add_argument('-exp', '--exp',
                        default='lp',
                        type=str,
                        help='experiments (lp, emb)')
    parser.add_argument('-rd', '--resultdir',
                        type=str,
                        default='./results_link_all',
                        help="result directory name")

    args = parser.parse_args()
    epochs = args.epochs
    dim_emb = args.embeddimension
```

(continues on next page)

(continued from previous page)

```
    length = args.timelength

    if not os.path.exists('./intermediate'):
          os.mkdir('./intermediate')

    if args.testDataType == 'sbm_cd':
        node_num = 100
        community_num = 2
        node_change_num = args.nodemigration
        dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                                ␣
↪community_num,
                                                                                ␣
↪length,
                                                                                 1,
                                                                                 node_
↪change_num)

        embedding = AE(d=dim_emb,
                       beta=5,
                       nu1=1e-6,
                       nu2=1e-6,
                       K=3,
                       n_units=[500, 300, ],
                       n_iter=epochs,
                       xeta=1e-4,
                       n_batch=100,
                       modelfile=['./intermediate/AE_enc_modelsbm.json',
                                  './intermediate/AE_dec_modelsbm.json'],
                       weightfile=['./intermediate/AE_enc_weightssbm.hdf5',
                                   './intermediate/AE_dec_weightssbm.hdf5'])

        graphs = [g[0] for g in dynamic_sbm_series]
        embs = []

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/staticAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            for temp_var in range(length):
                emb, _ = embedding.learn_embeddings(graphs[temp_var])
                embs.append(emb)

            result = Parallel(n_jobs=4)(
                delayed(embedding.learn_embeddings)(graphs[temp_var]) for temp_var in␣
↪range(length))
            for i in range(len(result)):
                embs.append(np.asarray(result[i][0]))
```

(continues on next page)

(continued from previous page)

```
            plt.figure()
            plt.clf()
            viz.plot_static_sbm_embedding(embs[-4:], dynamic_sbm_series[-4:])

            plt.savefig('./' + outdir + '/V_AE_nm' + str(args.nodemigration) + '_l' +␣
↪str(length) + '_epoch' + str(
                epochs) + '_emb' + str(dim_emb) + '.pdf', bbox_inches='tight',␣
↪dpi=600)
            plt.show()
            plt.close()

        if args.exp == 'lp':
            lp.expstaticLP(dynamic_sbm_series,
                           graphs,
                           embedding,
                           1,
                           outdir + '/',
                           'nm' + str(args.nodemigration) + '_l' + str(length) + '_emb
↪' + str(dim_emb),
                           )

    elif args.testDataType == 'academic':
        print("datatype:", args.testDataType)

        embedding = AE(d=dim_emb,
                       beta=5,
                       nu1=1e-6,
                       nu2=1e-6,
                       K=3,
                       n_units=[500, 300, ],
                       n_iter=epochs,
                       xeta=1e-4,
                       n_batch=1000,
                       modelfile=['./intermediate/enc_modelacdm.json',
                                  './intermediate/dec_modelacdm.json'],
                       weightfile=['./intermediate/enc_weightsacdm.hdf5',
                                   './intermediate/dec_weightsacdm.hdf5'])

        sample = args.samples
        if not os.path.exists('./test_data/academic/pickle'):
            os.mkdir('./test_data/academic/pickle')
            graphs, length = dataprep_util.get_graph_academic('./test_data/academic/
↪adjlist')
            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/academic/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/academic/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/academic/pickle/' +␣
↪str(i)))

        G_cen = nx.degree_centrality(graphs[29])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
```

(continues on next page)

(continued from previous page)

```
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1

        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/staticAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            lp.expstaticLP(None,
                           graphs[-args.timelength:],
                           embedding,
                           1,
                           outdir + '/',
                           'l' + str(args.timelength) + '_emb' + str(dim_emb) + '_
↪samples' + str(sample),
                           n_sample_nodes=graphs[i].number_of_nodes()
                           )

    elif args.testDataType == 'hep':
        print("datatype:", args.testDataType)
        embedding = AE(d=dim_emb,
                       beta=5,
                       nu1=1e-6,
                       nu2=1e-6,
                       K=3,
                       n_units=[500, 300, ],
                       n_iter=epochs,
                       xeta=1e-4,
                       n_batch=1000,
                       modelfile=['./intermediate/enc_modelhep.json',
                                  './intermediate/dec_modelhep.json'],
                       weightfile=['./intermediate/enc_weightshep.hdf5',
                                   './intermediate/dec_weightshep.hdf5'])

        if not os.path.exists('./test_data/hep/pickle'):
            os.mkdir('./test_data/hep/pickle')
            files = [file for file in os.listdir('./test_data/hep/hep-th') if '.
↪gpickle' in file]
            length = len(files)
            graphs = []
            for i in range(length):
                G = nx.read_gpickle('./test_data/hep/hep-th/month_' + str(i + 1) + '_
↪graph.gpickle')
```

(continues on next page)

(continued from previous page)

```
                graphs.append(G)
            total_nodes = graphs[-1].number_of_nodes()

            for i in range(length):
                for j in range(total_nodes):
                    if j not in graphs[i].nodes():
                        graphs[i].add_node(j)

            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/hep/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/hep/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/hep/pickle/' + str(i)))

        # pdb.set_trace()
        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/staticAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            lp.expstaticLP(None,
                           graphs[-args.timelength:],
                           embedding,
                           1,
                           outdir + '/',
                           'l' + str(args.timelength) + '_emb' + str(dim_emb) + '_
↪samples' + str(sample),
                           n_sample_nodes=graphs[i].number_of_nodes()
                           )

    elif args.testDataType == 'AS':
        print("datatype:", args.testDataType)
```

(continues on next page)

(continued from previous page)

```
        embedding = AE(d=dim_emb,
                       beta=5,
                       nu1=1e-6,
                       nu2=1e-6,
                       K=3,
                       n_units=[500, 300, ],
                       n_iter=epochs,
                       xeta=1e-4,
                       n_batch=1000,
                       modelfile=['./intermediate/enc_modelAS.json',
                                  './intermediate/dec_modelAS.json'],
                       weightfile=['./intermediate/enc_weightsAS.hdf5',
                                   './intermediate/dec_weightsAS.hdf5'])

        files = [file for file in os.listdir('./test_data/AS/as-733') if '.gpickle'␣
↪in file]
        length = len(files)
        graphs = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/AS/as-733/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphs.append(G)

        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/staticAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            lp.expstaticLP(None,
                           graphs[-args.timelength:],
                           embedding,
                           1,
                           outdir + '/',
                           'l' + str(args.timelength) + '_emb' + str(dim_emb) + '_
↪samples' + str(sample),
```

(continues on next page)

(continued from previous page)

```
                            n_sample_nodes=graphs[i].number_of_nodes()
                            )

    elif args.testDataType == 'enron':
        print("datatype:", args.testDataType)
        embedding = AE(d=dim_emb,
                       beta=5,
                       nu1=1e-6,
                       nu2=1e-6,
                       K=3,
                       n_units=[500, 300, ],
                       n_iter=epochs,
                       xeta=1e-8,
                       n_batch=20,
                       modelfile=['./intermediate/enc_modelAS.json',
                                  './intermediate/dec_modelAS.json'],
                       weightfile=['./intermediate/enc_weightsAS.hdf5',
                                   './intermediate/dec_weightsAS.hdf5'])

        files = [file for file in os.listdir('./test_data/enron') if '.gpickle' in␣
↪file if 'month' in file]
        length = len(files)
        graphsall = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/enron/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphsall.append(G)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/staticAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            sample = graphsall[0].number_of_nodes()
            graphs = graphsall[-args.timelength:]
            lp.expstaticLP(None,
                           graphs,
                           embedding,
                           1,
                           outdir + '/',
                           'l' + str(args.timelength) + '_emb' + str(dim_emb) + '_
↪samples' + str(sample),
                           n_sample_nodes=sample
                           )
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.5 Example Code for DynamicRNN

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

disp_avlbl = True
import os
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
    matplotlib.use('Agg')
import matplotlib.pyplot as plt
from joblib import Parallel, delayed
import operator
from argparse import ArgumentParser
from time import time

from dynamicgem.embedding.dynRNN import DynRNN
from dynamicgem.utils import plot_util, graph_util, dataprep_util
from dynamicgem.visualization import plot_dynamic_sbm_embedding
from dynamicgem.graph_generation import dynamic_SBM_graph
from dynamicgem.evaluation import evaluate_link_prediction
from dynamicgem.utils.dnn_utils import *


if __name__ == '__main__':
    parser = ArgumentParser(description='Learns node embeddings for a sequence of␣
↪graph snapshots')
    parser.add_argument('-t', '--testDataType',
                        default='sbm_cd',
                        type=str,
                        help='Type of data to test the code')
    parser.add_argument('-c', '--criteria',
                        default='degree',
                        type=str,
                        help='Node Migration criteria')
    parser.add_argument('-rc', '--criteria_r',
                        default=True,
                        type=bool,
                        help='Take highest centrality measure to perform node␣
↪migration')
    parser.add_argument('-l', '--timelength',
                        default=7,
                        type=int,
                        help='Number of time series graph to generate')
    parser.add_argument('-lb', '--lookback',
                        default=2,
                        type=int,
```

(continues on next page)

(continued from previous page)

```
                        help='number of lookbacks')
    parser.add_argument('-nm', '--nodemigration',
                        default=2,
                        type=int,
                        help='number of nodes to migrate')
    parser.add_argument('-iter', '--epochs',
                        default=2,
                        type=int,
                        help='number of epochs')
    parser.add_argument('-emb', '--embeddimension',
                        default=16,
                        type=int,
                        help='embedding dimension')
    parser.add_argument('-rd', '--resultdir',
                        type=str,
                        default='./results_link_all',
                        help="result directory name")
    parser.add_argument('-sm', '--samples',
                        default=5,
                        type=int,
                        help='samples for test data')
    parser.add_argument('-eta', '--learningrate',
                        default=1e-3,
                        type=float,
                        help='learning rate')
    parser.add_argument('-bs', '--batch',
                        default=10,
                        type=int,
                        help='batch size')
    parser.add_argument('-ht', '--hypertest',
                        default=0,
                        type=int,
                        help='hyper test')
    parser.add_argument('-exp', '--exp',
                        default='lp',
                        type=str,
                        help='experiments (lp, emb)')

    args = parser.parse_args()
    epochs = args.epochs
    dim_emb = args.embeddimension
    lookback = args.lookback
    length = args.timelength

    if not os.path.exists('./intermediate'):
          os.mkdir('./intermediate')

    if length < lookback + 5:
        length = lookback + 5

    if args.testDataType == 'sbm_rp':
        node_num = 100
        community_num = 50
        node_change_num = 10
        dynamic_sbm_series = dynamic_SBM_graph.get_random_perturbation_series(node_
↪num, community_num, length,
                                                                              node_
↪change_num)
```

(continues on next page)

(continued from previous page)

```
        dynamic_embedding = DynRNN(
            d=100,
            beta=100,
            n_prev_graphs=5,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[50, 30, ],
            rho=0.3,
            n_iter=30,
            xeta=0.005,
            n_batch=50,
            modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json
↪'],
            weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_
↪weights.hdf5'],
        )
        dynamic_embedding.learn_embeddings([g[0] for g in dynamic_sbm_series])
        plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding(dynamic_embedding.get_
↪embeddings(), dynamic_sbm_series)
        plt.savefig('result/visualization_DynRNN_rp.png')
        plt.show()
    elif args.testDataType == 'sbm_cd':
        node_num = 100
        community_num = 2
        node_change_num = args.nodemigration
        dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                                 ␣
↪community_num, length, 1,
                                                                                  node_
↪change_num)
        dynamic_embedding = DynRNN(
            d=dim_emb,  # 128,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_enc_units=[500, 300],
            n_dec_units=[500, 300],
            rho=0.3,
            n_iter=epochs,
            xeta=args.learningrate,
            n_batch=args.batch,
            modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json
↪'],
            weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_
↪weights.hdf5'],
            savefilesuffix="testing"
        )
        graphs = [g[0] for g in dynamic_sbm_series]

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)
```

(continues on next page)

(continued from previous page)

```
        outdir = outdir + '/dynRNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            embs = []
            result = Parallel(n_jobs=4)(delayed(dynamic_embedding.learn_
↪embeddings)(graphs[:temp_var]) for temp_var in
                                        range(lookback + 1, length + 1))
            for i in range(len(result)):
                embs.append(np.asarray(result[i][0]))

            for temp_var in range(lookback + 1, length + 1):
                emb, _ = dynamic_embedding.learn_embeddings(graphs[:temp_var])
                embs.append(emb)
            plt.figure()
            plt.clf()
            plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding_v2(embs[-5:-1],␣
↪dynamic_sbm_series[-5:])
            plt.savefig('./' + outdir + '/V_DynRNN_nm' + str(args.nodemigration) + '_l
↪' + str(length) + '_epoch' + str(
                epochs) + '_emb' + str(dim_emb) + '.pdf', bbox_inches='tight',␣
↪dpi=600)
            plt.show()

        if args.hypertest == 1:
            fname = 'epoch' + str(args.epochs) + '_bs' + str(args.batch) + '_lb' +␣
↪str(args.lookback) + '_eta' + str(
                args.learningrate) + '_emb' + str(args.embeddimension)
        else:
            fname = 'nm' + str(args.nodemigration) + '_l' + str(length) + '_emb' +␣
↪str(dim_emb)

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(
                graphs,
                dynamic_embedding,
                1,
                outdir + '/',
                fname,
            )

    elif args.testDataType == 'academic':
        print("datatype:", args.testDataType)

        dynamic_embedding = DynRNN(
            d=dim_emb,  # 128,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_enc_units=[500, 300],
            n_dec_units=[500, 300],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-3,
```

(continues on next page)

(continued from previous page)

```
            n_batch=int(args.samples / 10),
            modelfile=['./intermediate/enc_modelRNN.json', './intermediate/dec_
↪modelRNN.json'],
            weightfile=['./intermediate/enc_weightsRNN.hdf5', './intermediate/dec_
↪weightsRNN.hdf5'],
            savefilesuffix="testing"
        )

        sample = args.samples
        if not os.path.exists('./test_data/academic/pickle'):
            os.mkdir('./test_data/academic/pickle')
            graphs, length = dataprep_util.get_graph_academic('./test_data/academic/
↪adjlist')
            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/academic/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/academic/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/academic/pickle/' +␣
↪str(i)))

        G_cen = nx.degree_centrality(graphs[29])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynRNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=graphs[i].number_of_nodes()
                                           )
```

(continues on next page)

(continued from previous page)

```
    elif args.testDataType == 'hep':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynRNN(
            d=dim_emb,  # 128,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_enc_units=[500, 300],
            n_dec_units=[500, 300],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-3,
            n_batch=int(args.samples / 10),
            modelfile=['./intermediate/enc_modelRNN.json', './intermediate/dec_
↪modelRNN.json'],
            weightfile=['./intermediate/enc_weightsRNN.hdf5', './intermediate/dec_
↪weightsRNN.hdf5'],
            savefilesuffix="testing"
        )

        if not os.path.exists('./test_data/hep/pickle'):
            os.mkdir('./test_data/hep/pickle')
            files = [file for file in os.listdir('./test_data/hep/hep-th') if '.
↪gpickle' in file]
            length = len(files)
            graphs = []
            for i in range(length):
                G = nx.read_gpickle('./test_data/hep/hep-th/month_' + str(i + 1) + '_
↪graph.gpickle')

                graphs.append(G)
            total_nodes = graphs[-1].number_of_nodes()

            for i in range(length):
                for j in range(total_nodes):
                    if j not in graphs[i].nodes():
                        graphs[i].add_node(j)

            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/hep/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/hep/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/hep/pickle/' + str(i)))

        # pdb.set_trace()
        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
```

(continues on next page)

(continued from previous page)

```
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynRNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=graphs[i].number_of_nodes()
                                           )


    elif args.testDataType == 'AS':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynRNN(
            d=dim_emb,  # 128,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_enc_units=[500, 300],
            n_dec_units=[500, 300],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-3,
            n_batch=int(args.samples / 10),
            modelfile=['./intermediate/enc_modelRNN.json', './intermediate/dec_
↪modelRNN.json'],
            weightfile=['./intermediate/enc_weightsRNN.hdf5', './intermediate/dec_
↪weightsRNN.hdf5'],
            savefilesuffix="testing"
        )

        files = [file for file in os.listdir('./test_data/AS/as-733') if '.gpickle'␣
↪in file]
        length = len(files)
        graphs = []

        for i in range(length):
```

(continues on next page)

(continued from previous page)

```
            G = nx.read_gpickle('./test_data/AS/as-733/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphs.append(G)

        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynRNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=graphs[i].number_of_nodes()
                                           )

    elif args.testDataType == 'enron':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynRNN(
            d=dim_emb,  # 128,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-4,
            nu2=1e-4,
            n_enc_units=[100, 80],
            n_dec_units=[100, 80],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-7,
            n_batch=2000,
            modelfile=['./intermediate/enc_modelRNN.json', './intermediate/dec_
↪modelRNN.json'],
```

(continues on next page)

(continued from previous page)

```
            weightfile=['./intermediate/enc_weightsRNN.hdf5', './intermediate/dec_
↪weightsRNN.hdf5'],
            savefilesuffix="testing"
        )

        files = [file for file in os.listdir('./test_data/enron') if 'week' in file]
        length = len(files)
        graphs = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/enron/week_' + str(i) + '_graph.gpickle')
            graphs.append(G)

        sample = graphs[0].number_of_nodes()

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynRNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=sample
                                           )
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.6 Example Code for Dynamic AE and RNN

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
```

(continues on next page)

(continued from previous page)

```
import os
disp_avlbl = True
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
    matplotlib.use('Agg')
import matplotlib.pyplot as plt

import operator
from argparse import ArgumentParser
from time import time
from joblib import Parallel, delayed

from dynamicgem.embedding.dynAERNN import DynAERNN
from dynamicgem.utils import plot_util, graph_util, dataprep_util
from dynamicgem.visualization import plot_dynamic_sbm_embedding
from dynamicgem.graph_generation import dynamic_SBM_graph
from dynamicgem.utils.dnn_utils import *
from dynamicgem.evaluation import evaluate_link_prediction


if __name__ == '__main__':
    parser = ArgumentParser(description='Learns node embeddings for a sequence of␣
↪graph snapshots')
    parser.add_argument('-t', '--testDataType',
                        default='sbm_cd',
                        type=str,
                        help='Type of data to test the code')
    parser.add_argument('-c', '--criteria',
                        default='degree',
                        type=str,
                        help='Node Migration criteria')
    parser.add_argument('-rc', '--criteria_r',
                        default=True,
                        type=bool,
                        help='Take highest centrality measure to perform node␣
↪migration')
    parser.add_argument('-l', '--timelength',
                        default=4,
                        type=int,
                        help='Number of time series graph to generate')
    parser.add_argument('-lb', '--lookback',
                        default=2,
                        type=int,
                        help='number of lookbacks')
    parser.add_argument('-nm', '--nodemigration',
                        default=2,
                        type=int,
                        help='number of nodes to migrate')
    parser.add_argument('-iter', '--epochs',
                        default=2,
                        type=int,
                        help='number of epochs')
    parser.add_argument('-emb', '--embeddimension',
                        default=16,
                        type=int,
                        help='embedding dimension')
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('-rd', '--resultdir',
                        type=str,
                        default='./results_link_all',
                        help="result directory name")
    parser.add_argument('-sm', '--samples',
                        default=5,
                        type=int,
                        help='samples for test data')
    parser.add_argument('-eta', '--learningrate',
                        default=1e-3,
                        type=float,
                        help='learning rate')
    parser.add_argument('-bs', '--batch',
                        default=10,
                        type=int,
                        help='batch size')
    parser.add_argument('-ht', '--hypertest',
                        default=0,
                        type=int,
                        help='hyper test')
    parser.add_argument('-fs', '--show',
                        default=0,
                        type=int,
                        help='show figure ')
    parser.add_argument('-exp', '--exp',
                        default='lp',
                        type=str,
                        help='experiments (lp, emb)')

    args = parser.parse_args()
    epochs = args.epochs
    dim_emb = args.embeddimension
    lookback = args.lookback
    length = args.timelength

    if not os.path.exists('./intermediate'):
          os.mkdir('./intermediate')

    if length < 7:
        length = 7
    lookback = args.lookback

    if args.testDataType == 'sbm_rp':
        node_num = 1000
        community_num = 50
        node_change_num = 10
        dynamic_sbm_series = dynamic_SBM_graph.get_random_perturbation_series(node_
↪num, community_num, length,
                                                                              node_
↪change_num)
        dynamic_embedding = DynAERNN(
            d=100,
            beta=100,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[50, 30, ],
```

(continues on next page)

(continued from previous page)

```
            rho=0.3,
            n_iter=30,
            xeta=0.005,
            n_batch=50,
            modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json
↪'],
            weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_
↪weights.hdf5'],
        )
        dynamic_embedding.learn_embeddings([g[0] for g in dynamic_sbm_series])
        plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding(dynamic_embedding.get_
↪embeddings(), dynamic_sbm_series)
        plt.savefig('result/visualization_DynRNN_rp.png')
        plt.show()
    elif args.testDataType == 'sbm_cd':
        node_num = 100
        community_num = 2
        node_change_num = args.nodemigration
        dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                                ␣
↪community_num, length, 1,
                                                                                 node_
↪change_num)
        dynamic_embedding = DynAERNN(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_aeunits=[500, 300],
            n_lstmunits=[500, dim_emb],
            rho=0.3,
            n_iter=epochs,
            xeta=args.learningrate,
            n_batch=args.batch,
            modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json
↪'],
            weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_
↪weights.hdf5'],
            savefilesuffix="testing"
        )
        graphs = [g[0] for g in dynamic_sbm_series]

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAERNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            embs = []
```

(continues on next page)

(continued from previous page)

```
            result = Parallel(n_jobs=4)(delayed(dynamic_embedding.learn_
↪embeddings)(graphs[:temp_var]) for temp_var in
                                        range(lookback + 1, length + 1))
            for i in range(len(result)):
                embs.append(np.asarray(result[i][0]))
            plt.figure()
            plt.clf()
            plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding_v2(embs[-5:-1],␣
↪dynamic_sbm_series[-5:])
            plt.savefig(
                './' + outdir + '/V_DynAERNN_nm' + str(args.nodemigration) + '_l' +␣
↪str(length) + '_epoch' + str(
                    epochs) + '_emb' + str(dim_emb) + '.pdf', bbox_inches='tight',␣
↪dpi=600)
            plt.show()

        if args.hypertest == 1:
            fname = 'epoch' + str(args.epochs) + '_bs' + str(args.batch) + '_lb' +␣
↪str(args.lookback) + '_eta' + str(
                args.learningrate) + '_emb' + str(args.embeddimension)
        else:
            fname = 'nm' + str(args.nodemigration) + '_l' + str(length) + '_emb' +␣
↪str(dim_emb)

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(
                graphs,
                dynamic_embedding,
                1,
                outdir + '/',
                fname,
            )

    elif args.testDataType == 'academic':
        print("datatype:", args.testDataType)

        dynamic_embedding = DynAERNN(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_aeunits=[500, 300],
            n_lstmunits=[500, dim_emb],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-3,
            n_batch=100,
            modelfile=['./intermediate/enc_modelAERNN.json', './intermediate/dec_
↪modelAERNN.json'],
            weightfile=['./intermediate/enc_weightsAERNN.hdf5', './intermediate/dec_
↪weightsAERNN.hdf5'],
            savefilesuffix="testing"
        )

        sample = args.samples
        if not os.path.exists('./test_data/academic/pickle'):
```

(continues on next page)

(continued from previous page)

```
            os.mkdir('./test_data/academic/pickle')
            graphs, length = dataprep_util.get_graph_academic('./test_data/academic/
↪adjlist')
            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/academic/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/academic/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/academic/pickle/' +␣
↪str(i)))

        G_cen = nx.degree_centrality(graphs[29])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        # pdb.set_trace()
        # node_l = np.random.choice(range(graphs[29].number_of_nodes()), 5000,␣
↪replace=False)
        # print(node_l)
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)
        # pdb.set_trace()

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAERNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=graphs[i].number_of_nodes()
                                           )

    elif args.testDataType == 'hep':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynAERNN(
            d=dim_emb,
```

(continues on next page)

(continued from previous page)

```
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_aeunits=[500, 300],
            n_lstmunits=[500, dim_emb],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-3,
            n_batch=100,
            modelfile=['./intermediate/enc_modelAERNN.json', './intermediate/dec_
↪modelAERNN.json'],
            weightfile=['./intermediate/enc_weightsAERNN.hdf5', './intermediate/dec_
↪weightsAERNN.hdf5'],
            savefilesuffix="testing"
        )

        if not os.path.exists('./test_data/hep/pickle'):
            os.mkdir('./test_data/hep/pickle')
            files = [file for file in os.listdir('./test_data/hep/hep-th') if '.
↪gpickle' in file]
            length = len(files)
            graphs = []
            for i in range(length):
                G = nx.read_gpickle('./test_data/hep/hep-th/month_' + str(i + 1) + '_
↪graph.gpickle')

                graphs.append(G)
            total_nodes = graphs[-1].number_of_nodes()

            for i in range(length):
                for j in range(total_nodes):
                    if j not in graphs[i].nodes():
                        graphs[i].add_node(j)

            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/hep/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/hep/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/hep/pickle/' + str(i)))

        # pdb.set_trace()
        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
```

(continues on next page)

(continued from previous page)

```
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAERNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=graphs[i].number_of_nodes()
                                           )

    elif args.testDataType == 'AS':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynAERNN(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_aeunits=[500, 300],
            n_lstmunits=[500, dim_emb],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-3,
            n_batch=100,
            modelfile=['./intermediate/enc_modelAERNN.json', './intermediate/dec_
↪modelAERNN.json'],
            weightfile=['./intermediate/enc_weightsAERNN.hdf5', './intermediate/dec_
↪weightsAERNN.hdf5'],
            savefilesuffix="testing"
        )

        files = [file for file in os.listdir('./test_data/AS/as-733') if '.gpickle'␣
↪in file]
        length = len(files)
        graphs = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/AS/as-733/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphs.append(G)

        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
```

(continues on next page)

(continued from previous page)

```
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAERNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=graphs[i].number_of_nodes()
                                           )

    elif args.testDataType == 'enron':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynAERNN(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-4,
            nu2=1e-4,
            n_aeunits=[100, 80],
            n_lstmunits=[100, 20],
            rho=0.3,
            n_iter=2000,
            xeta=1e-7,
            n_batch=100,
            modelfile=['./intermediate/enc_modelAERNN.json', './intermediate/dec_
↪modelAERNN.json'],
            weightfile=['./intermediate/enc_weightsAERNN.hdf5', './intermediate/dec_
↪weightsAERNN.hdf5'],
            savefilesuffix="testing"
        )

        files = [file for file in os.listdir('./test_data/enron') if 'week' in file]
        length = len(files)
```

(continues on next page)

(continued from previous page)

```
        graphsall = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/enron/week_' + str(i) + '_graph.gpickle')
            graphsall.append(G)

        sample = graphsall[0].number_of_nodes()

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAERNN'
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        graphs = graphsall[-args.timelength:]

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs,
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=sample
                                           )
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.7 Example Code for Dynamic AE

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

disp_avlbl = True
import os
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
```

(continues on next page)

(continued from previous page)

```
    matplotlib.use('Agg')
import matplotlib.pyplot as plt
import sys
from joblib import Parallel, delayed

import keras.regularizers as Reg
from argparse import ArgumentParser
from time import time
import operator

from dynamicgem.embedding.dynAE import DynAE
from dynamicgem.utils import plot_util, graph_util, dataprep_util
from dynamicgem.visualization import plot_dynamic_sbm_embedding
from dynamicgem.graph_generation import dynamic_SBM_graph
from dynamicgem.evaluation import evaluate_link_prediction, evaluate_graph_
↪reconstruction
from dynamicgem.utils.dnn_utils import *


if __name__ == '__main__':
    parser = ArgumentParser(description='Learns node embeddings for a sequence of␣
↪graph snapshots')
    parser.add_argument('-t', '--testDataType',
                        default='sbm_cd',
                        type=str,
                        help='Type of data to test the code')
    parser.add_argument('-c', '--criteria',
                        default='degree',
                        type=str,
                        help='Node Migration criteria')
    parser.add_argument('-rc', '--criteria_r',
                        default=1,
                        type=int,
                        help='Take highest centrality measure to perform node␣
↪migration')
    parser.add_argument('-l', '--timelength',
                        default=5,
                        type=int,
                        help='Number of time series graph to generate')
    parser.add_argument('-lb', '--lookback',
                        default=2,
                        type=int,
                        help='number of lookbacks')
    parser.add_argument('-eta', '--learningrate',
                        default=1e-4,
                        type=float,
                        help='learning rate')
    parser.add_argument('-bs', '--batch',
                        default=100,
                        type=int,
                        help='batch size')
    parser.add_argument('-nm', '--nodemigration',
                        default=2,
                        type=int,
                        help='number of nodes to migrate')
    parser.add_argument('-iter', '--epochs',
                        default=2,
```

(continues on next page)

(continued from previous page)

```
                        type=int,
                        help='number of epochs')
    parser.add_argument('-emb', '--embeddimension',
                        default=16,
                        type=int,
                        help='embedding dimension')
    parser.add_argument('-rd', '--resultdir',
                        type=str,
                        default='./results_link_all',
                        help="result directory name")
    parser.add_argument('-sm', '--samples',
                        default=10,
                        type=int,
                        help='samples for test data')
    parser.add_argument('-ht', '--hypertest',
                        default=0,
                        type=int,
                        help='hyper test')
    parser.add_argument('-exp', '--exp',
                        default='lp',
                        type=str,
                        help='experiments (lp, emb)')

    args = parser.parse_args()
    epochs = args.epochs
    dim_emb = args.embeddimension
    lookback = args.lookback
    length = args.timelength

    if not os.path.exists('./intermediate'):
          os.mkdir('./intermediate')

    if length < lookback + 5:
        length = lookback + 5
    if args.testDataType == 'sbm_rp':
        node_num = 10000
        community_num = 500
        node_change_num = 100
        dynamic_sbm_series = dynamic_SBM_graph.get_random_perturbation_series(node_
↪num,
                                                                              ␣
↪community_num,
                                                                               length,
                                                                               node_
↪change_num)
        dynamic_embedding = DynAE(
            d=100,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[500, 300, ],
            rho=0.3,
            n_iter=1000,
            xeta=0.005,
            n_batch=500,
            modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json
↪'],
```

(continues on next page)

(continued from previous page)

```
            weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_
↪weights.hdf5'],
        )
        dynamic_embedding.learn_embeddings([g[0] for g in dynamic_sbm_series])
        plt.clf()
        plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding(dynamic_embedding.get_
↪embeddings(),
                                                              dynamic_sbm_series)
        plt.savefig('result/visualization_DynRNN_rp.png')
        plt.show()

    elif args.testDataType == 'sbm_cd':
        node_num = 100
        community_num = 2
        node_change_num = args.nodemigration
        dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                                ␣
↪community_num,
                                                                                ␣
↪length,
                                                                                 1,  #␣
↪communitiy to dimisnish
                                                                                 node_
↪change_num
                                                                                 )
        dynamic_embedding = DynAE(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[500, 300, ],
            rho=0.3,
            n_iter=epochs,
            xeta=args.learningrate,
            n_batch=args.batch,
            modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json
↪'],
            weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_
↪weights.hdf5'],
            savefilesuffix="testing"
        )
        graphs = [g[0] for g in dynamic_sbm_series]
        embs = []

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)
```

(continues on next page)

(continued from previous page)

```
        if args.exp == 'emb':
            result = Parallel(n_jobs=4)(delayed(dynamic_embedding.learn_
↪embeddings)(graphs[:temp_var]) for temp_var in
                                        range(lookback + 1, length + 1))
            for i in range(len(result)):
                embs.append(np.asarray(result[i][0]))

            for temp_var in range(lookback + 1, length + 1):
                emb, _ = dynamic_embedding.learn_embeddings(graphs[:temp_var])
                embs.append(emb)

            plt.figure()
            plt.clf()
            plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding_v2(embs[-5:-1],␣
↪dynamic_sbm_series[-5:])
            plt.savefig('./' + outdir + '/V_DynAE_nm' + str(args.nodemigration) + '_l
↪' + str(length) + '_epoch' + str(
                epochs) + '_emb' + str(dim_emb) + '.pdf', bbox_inches='tight',␣
↪dpi=600)
            plt.show()

        if args.hypertest == 1:
            fname = 'epoch' + str(args.epochs) + '_bs' + str(args.batch) + '_lb' +␣
↪str(args.lookback) + '_eta' + str(
                args.learningrate) + '_emb' + str(args.embeddimension)
        else:
            fname = 'nm' + str(args.nodemigration) + '_l' + str(length) + '_emb' +␣
↪str(dim_emb)

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(
                graphs,
                dynamic_embedding,
                1,
                outdir + '/',
                fname,
            )

    elif args.testDataType == 'academic':
        print("datatype:", args.testDataType)

        dynamic_embedding = DynAE(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[500, 300, ],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-5,
            n_batch=100,
            modelfile=['./intermediate/enc_modelacdm.json', './intermediate/dec_
↪modelacdm.json'],
            weightfile=['./intermediate/enc_weightsacdm.hdf5', './intermediate/dec_
↪weightsacdm.hdf5'],
            savefilesuffix="testingacdm"
```

(continues on next page)

(continued from previous page)

```
        )

        sample = args.samples
        if not os.path.exists('./test_data/academic/pickle'):
            os.mkdir('./test_data/academic/pickle')
            graphs, length = dataprep_util.get_graph_academic('./test_data/academic/
↪adjlist')
            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/academic/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/academic/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/academic/pickle/' +␣
↪str(i)))

        G_cen = nx.degree_centrality(graphs[29])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb_' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=sample
                                           )

    elif args.testDataType == 'hep':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynAE(
            d=dim_emb,
            beta=5,
```

(continues on next page)

(continued from previous page)

```
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[500, 300, ],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-8,
            n_batch=int(args.samples / 10),
            modelfile=['./intermediate/enc_modelhep.json', './intermediate/dec_
↪modelhep.json'],
            weightfile=['./intermediate/enc_weightshep.hdf5', './intermediate/dec_
↪weightshep.hdf5'],
            savefilesuffix="testinghep"
        )

        if not os.path.exists('./test_data/hep/pickle'):
            os.mkdir('./test_data/hep/pickle')
            files = [file for file in os.listdir('./test_data/hep/hep-th') if '.
↪gpickle' in file]
            length = len(files)
            graphs = []
            for i in range(length):
                G = nx.read_gpickle('./test_data/hep/hep-th/month_' + str(i + 1) + '_
↪graph.gpickle')

                graphs.append(G)
            total_nodes = graphs[-1].number_of_nodes()

            for i in range(length):
                for j in range(total_nodes):
                    if j not in graphs[i].nodes():
                        graphs[i].add_node(j)

            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/hep/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/hep/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/hep/pickle/' + str(i)))

        # pdb.set_trace()
        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
```

(continues on next page)

(continued from previous page)

```
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb_' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=sample
                                           )

    elif args.testDataType == 'AS':
        print("datatype:", args.testDataType)
        dynamic_embedding = DynAE(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[500, 300, ],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-5,
            n_batch=int(args.samples / 10),
            modelfile=['./intermediate/enc_modelAS.json', './intermediate/dec_modelAS.
↪json'],
            weightfile=['./intermediate/enc_weightsAS.hdf5', './intermediate/dec_
↪weightsAS.hdf5'],
            savefilesuffix="testingAS"
        )

        files = [file for file in os.listdir('./test_data/AS/as-733') if '.gpickle'␣
↪in file]
        length = len(files)
        graphs = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/AS/as-733/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphs.append(G)

        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
```

(continues on next page)

(continued from previous page)

```
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb_' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=sample
                                           )

    elif args.testDataType == 'enron':
        print("datatype:", args.testDataType)

        dynamic_embedding = DynAE(
            d=dim_emb,
            beta=5,
            n_prev_graphs=lookback,
            nu1=1e-6,
            nu2=1e-6,
            n_units=[500, 300, ],
            rho=0.3,
            n_iter=epochs,
            xeta=1e-8,
            n_batch=20,
            modelfile=['./intermediate/enc_modelenron.json', './intermediate/dec_
↪modelenron.json'],
            weightfile=['./intermediate/enc_weightsenron.hdf5', './intermediate/dec_
↪weightsenron.hdf5'],
            savefilesuffix="testingAS"
        )

        files = [file for file in os.listdir('./test_data/enron') if 'week' in file]
        length = len(files)
        graphs = []
```

(continues on next page)

(continued from previous page)

```
        for i in range(length):
            G = nx.read_gpickle('./test_data/enron/week_' + str(i) + '_graph.gpickle')
            graphs.append(G)

        sample = graphs[0].number_of_nodes()
        print(sample)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynAE'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            evaluate_link_prediction.expLP(graphs[-args.timelength:],
                                           dynamic_embedding,
                                           1,
                                           outdir + '/',
                                           'lb_' + str(lookback) + '_l' + str(args.
↪timelength) + '_emb' + str(
                                               dim_emb) + '_samples' + str(sample),
                                           n_sample_nodes=sample
                                           )
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.8 Example Code for DynamicTRIAD

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

disp_avlbl = True
import os
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
    matplotlib.use('Agg')

import sys
```

(continues on next page)

(continued from previous page)

```
import tensorflow as tf
import argparse
import operator
import time
import os
import importlib
import pdb
import random
import networkx as nx

from dynamicgem.embedding.dynamicTriad import dynamicTriad
from dynamicgem.utils import graph_util, plot_util, dataprep_util
from dynamicgem.evaluation import visualize_embedding as viz
from dynamicgem.utils.sdne_utils import *
from dynamicgem.graph_generation import dynamic_SBM_graph
from dynamicgem.utils.dynamictriad_utils import *
import dynamicgem.utils.dynamictriad_utils.dataset.dataset_utils as du
import dynamicgem.utils.dynamictriad_utils.algorithm.embutils as eu
from dynamicgem.evaluation import evaluate_link_prediction as lp


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='Learns static node embeddings')
    parser.add_argument('-t', '--testDataType',
                        default='sbm_cd',
                        type=str,
                        help='Type of data to test the code')
    parser.add_argument('-nm', '--nodemigration',
                        default=2,
                        type=int,
                        help='number of nodes to migrate')
    parser.add_argument('-iter', '--niters',
                        type=int,
                        help="number of optimization iterations",
                        default=2)
    parser.add_argument('-m', '--starttime',
                        type=str,
                        help=argparse.SUPPRESS,
                        default=0)
    parser.add_argument('-d', '--datafile',
                        type=str,
                        help='input directory name')
    parser.add_argument('-b', '--batchsize',
                        type=int,
                        help="batchsize for training",
                        default=100)
    parser.add_argument('-n', '--nsteps',
                        type=int,
                        help="number of time steps",
                        default=4)
    parser.add_argument('-K', '--embdim',
                        type=int,
                        help="number of embedding dimensions",
                        default=32)
    parser.add_argument('-l', '--stepsize',
                        type=int,
```

(continues on next page)

(continued from previous page)

```
                        help="size of of a time steps",
                        default=1)
    parser.add_argument('-s', '--stepstride',
                        type=int,
                        help="interval between two time steps",
                        default=1)
    parser.add_argument('-o', '--outdir',
                        type=str,
                        default='./output',
                        help="output directory name")
    parser.add_argument('-rd', '--resultdir',
                        type=str,
                        default='./results_link_all',
                        help="result directory name")
    parser.add_argument('--lr',
                        type=float,
                        help="initial learning rate",
                        default=0.1)
    parser.add_argument('--beta-smooth',
                        type=float,
                        default=0.1,
                        help="coefficients for smooth component")
    parser.add_argument('--beta-triad',
                        type=float,
                        default=0.1,
                        help="coefficients for triad component")
    parser.add_argument('--negdup',
                        type=int,
                        help="neg/pos ratio during sampling",
                        default=1)
    parser.add_argument('--datasetmod',
                        type=str,
                        default='dynamicgem.utils.dynamictriad_utils.dataset.adjlist',
                        help='module name for dataset loading',
                        )
    parser.add_argument('--validation',
                        type=str,
                        default='link_reconstruction',
                        help=', '.join(list(sorted(set(du.TestSampler.tasks) & set(eu.
↪Validator.tasks)))))
    parser.add_argument('-te', '--test',
                        type=str,
                        nargs='+',
                        default='link_predict',
                        help='type of test, (node_classify, node_predict, link_
↪classify, link_predict, '
                             'changed_link_classify, changed_link_predict, all)')
    parser.add_argument('--classifier',
                        type=str,
                        default='lr',
                        help='lr, svm')
    parser.add_argument('--repeat',
                        type=int,
                        default=1,
                        help='number of times to repeat experiment')
    parser.add_argument('-sm', '--samples',
                        default=5000,
```

(continues on next page)

(continued from previous page)

```
                        type=int,
                        help='samples for test data')
    args = parser.parse_args()
    if not os.path.exists(args.outdir):
        os.mkdir(args.outdir)
    args.embdir = args.outdir + '/dynTriad/' + args.testDataType
    args.cachefn = '/tmp/' + args.testDataType
    args.beta = [args.beta_smooth, args.beta_triad]
    # some fixed arguments in published code
    args.pretrain_size = args.nsteps
    args.trainmod = 'dynamicgem.utils.dynamictriad_utils.algorithm.dynamic_triad'
    args.sampling_args = {}
    args.debug = False
    args.scale = 1

    if args.validation not in du.TestSampler.tasks:
        raise NotImplementedError("Validation task {} not supported in TestSampler".
↪format(args.validation))
    if args.validation not in eu.Validator.tasks:
        raise NotImplementedError("Validation task {} not supported in Validator".
↪format(args.validation))

    print("running with options: ", args.__dict__)

    epochs = args.niters
    length = args.nsteps

    if args.testDataType == 'sbm_cd':
        node_num = 200
        community_num = 2
        node_change_num = args.nodemigration
        dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                                ␣
↪community_num,
                                                                                ␣
↪length,
                                                                                 1,
                                                                                 node_
↪change_num)
        graphs = [g[0] for g in dynamic_sbm_series]

        datafile = dataprep_util.prep_input_dynTriad(graphs, length, args.
↪testDataType)

        embedding = dynamicTriad(niters=args.niters,
                                 starttime=args.starttime,
                                 datafile=datafile,
                                 batchsize=args.batchsize,
                                 nsteps=args.nsteps,
                                 embdim=args.embdim,
                                 stepsize=args.stepsize,
                                 stepstride=args.stepstride,
                                 outdir=args.outdir,
                                 cachefn=args.cachefn,
                                 lr=args.lr,
                                 beta=args.beta,
```

(continues on next page)

(continued from previous page)

```
                                    negdup=args.negdup,
                                    datasetmod=args.datasetmod,
                                    trainmod=args.trainmod,
                                    pretrain_size=args.pretrain_size,
                                    sampling_args=args.sampling_args,
                                    validation=args.validation,
                                    datatype=args.testDataType,
                                    scale=args.scale,
                                    classifier=args.classifier,
                                    debug=args.debug,
                                    test=args.test,
                                    repeat=args.repeat,
                                    resultdir=args.resultdir,
                                    testDataType=args.testDataType,
                                    clname='lr',
                                    node_num=node_num )

        embedding.learn_embedding()
        embedding.get_embedding()
        # embedding.plotresults(dynamic_sbm_series)

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + 'dynTRIAD'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        lp.expstaticLP_TRIAD(dynamic_sbm_series,
                             graphs,
                             embedding,
                             1,
                             outdir + '/',
                             'nm' + str(args.nodemigration) + '_l' + str(args.nsteps)␣
↪+ '_emb' + str(args.embdim),
                             )


    elif args.testDataType == 'academic':
        print("datatype:", args.testDataType)

        sample = args.samples
        if not os.path.exists('./test_data/academic/pickle'):
            os.mkdir('./test_data/academic/pickle')
            graphs, length = dataprep_util.get_graph_academic('./test_data/academic/
↪adjlist')
            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/academic/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/academic/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/academic/pickle/' +␣
↪str(i)))
```

(continues on next page)

(continued from previous page)

```
        G_cen = nx.degree_centrality(graphs[29])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        # pdb.set_trace()
        # node_l = np.random.choice(range(graphs[29].number_of_nodes()), 5000,␣
↪replace=False)
        # print(node_l)
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)
        # pdb.set_trace()
        graphs = graphs[-args.nsteps:]
        datafile = dataprep_util.prep_input_dynTriad(graphs, args.nsteps, args.
↪testDataType)

        embedding = dynamicTriad(niters=args.niters,
                                 starttime=args.starttime,
                                 datafile=datafile,
                                 batchsize=args.batchsize,
                                 nsteps=args.nsteps,
                                 embdim=args.embdim,
                                 stepsize=args.stepsize,
                                 stepstride=args.stepstride,
                                 outdir=args.outdir,
                                 cachefn=args.cachefn,
                                 lr=args.lr,
                                 beta=args.beta,
                                 negdup=args.negdup,
                                 datasetmod=args.datasetmod,
                                 trainmod=args.trainmod,
                                 pretrain_size=args.pretrain_size,
                                 sampling_args=args.sampling_args,
                                 validation=args.validation,
                                 datatype=args.testDataType,
                                 scale=args.scale,
                                 classifier=args.classifier,
                                 debug=args.debug,
                                 test=args.test,
                                 repeat=args.repeat,
                                 resultdir=args.resultdir,
                                 testDataType=args.testDataType,
                                 clname='lr',
                                 node_num=sample

                                 )
        embedding.learn_embedding()
        embedding.get_embedding()

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
```

(continues on next page)

(continued from previous page)

```
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynTriad'
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        lp.expstaticLP_TRIAD(None,
                             graphs,
                             embedding,
                             1,
                             outdir + '/',
                             'l' + str(args.nsteps) + '_emb' + str(args.embdim) + '_
↪samples' + str(sample),
                             n_sample_nodes=sample
                             )


    elif args.testDataType == 'hep':
        print("datatype:", args.testDataType)

        if not os.path.exists('./test_data/hep/pickle'):
            os.mkdir('./test_data/hep/pickle')
            files = [file for file in os.listdir('./test_data/hep/hep-th') if '.
↪gpickle' in file]
            length = len(files)
            graphs = []
            for i in range(length):
                G = nx.read_gpickle('./test_data/hep/hep-th/month_' + str(i + 1) + '_
↪graph.gpickle')

                graphs.append(G)
            total_nodes = graphs[-1].number_of_nodes()

            for i in range(length):
                for j in range(total_nodes):
                    if j not in graphs[i].nodes():
                        graphs[i].add_node(j)

            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/hep/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/hep/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/hep/pickle/' + str(i)))

        # pdb.set_trace()
        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
```

(continues on next page)

(continued from previous page)

```
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_1)

        graphs = graphs[-args.nsteps:]
        datafile = dataprep_util.prep_input_dynTriad(graphs, args.nsteps, args.
↪testDataType)

        embedding = dynamicTriad(niters=args.niters,
                                 starttime=args.starttime,
                                 datafile=datafile,
                                 batchsize=args.batchsize,
                                 nsteps=args.nsteps,
                                 embdim=args.embdim,
                                 stepsize=args.stepsize,
                                 stepstride=args.stepstride,
                                 outdir=args.outdir,
                                 cachefn=args.cachefn,
                                 lr=args.lr,
                                 beta=args.beta,
                                 negdup=args.negdup,
                                 datasetmod=args.datasetmod,
                                 trainmod=args.trainmod,
                                 pretrain_size=args.pretrain_size,
                                 sampling_args=args.sampling_args,
                                 validation=args.validation,
                                 datatype=args.testDataType,
                                 scale=args.scale,
                                 classifier=args.classifier,
                                 debug=args.debug,
                                 test=args.test,
                                 repeat=args.repeat,
                                 resultdir=args.resultdir,
                                 testDataType=args.testDataType,
                                 clname='lr',
                                 node_num=sample

                                 )
        embedding.learn_embedding()
        embedding.get_embedding()

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynTriad'
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        lp.expstaticLP_TRIAD(None,
                             graphs,
                             embedding,
                             1,
                             outdir + '/',
                             'l' + str(args.nsteps) + '_emb' + str(args.embdim) + '_
↪samples' + str(sample),
                             n_sample_nodes=sample
```

(continues on next page)

(continued from previous page)

```
                                )


    elif args.testDataType == 'AS':
        print("datatype:", args.testDataType)

        files = [file for file in os.listdir('./test_data/AS/as-733') if '.gpickle'␣
↪in file]
        length = len(files)
        graphs = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/AS/as-733/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphs.append(G)

        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        graphs = graphs[-args.nsteps:]
        datafile = dataprep_util.prep_input_dynTriad(graphs, args.nsteps, args.
↪testDataType)

        embedding = dynamicTriad(niters=args.niters,
                                 starttime=args.starttime,
                                 datafile=datafile,
                                 batchsize=args.batchsize,
                                 nsteps=args.nsteps,
                                 embdim=args.embdim,
                                 stepsize=args.stepsize,
                                 stepstride=args.stepstride,
                                 outdir=args.outdir,
                                 cachefn=args.cachefn,
                                 lr=args.lr,
                                 beta=args.beta,
                                 negdup=args.negdup,
                                 datasetmod=args.datasetmod,
                                 trainmod=args.trainmod,
                                 pretrain_size=args.pretrain_size,
                                 sampling_args=args.sampling_args,
                                 validation=args.validation,
                                 datatype=args.testDataType,
                                 scale=args.scale,
                                 classifier=args.classifier,
                                 debug=args.debug,
                                 test=args.test,
                                 repeat=args.repeat,
                                 resultdir=args.resultdir,
```

(continues on next page)

(continued from previous page)

```
                                testDataType=args.testDataType,
                                clname='lr',
                                node_num=sample

                                )

        embedding.learn_embedding()
        embedding.get_embedding()

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynTriad'
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        lp.expstaticLP_TRIAD(None,
                             graphs,
                             embedding,
                             1,
                             outdir + '/',
                             'l' + str(args.nsteps) + '_emb' + str(args.embdim) + '_
↪samples' + str(sample),
                             n_sample_nodes=sample
                             )

    elif args.testDataType == 'enron':
        print("datatype:", args.testDataType)

        files = [file for file in os.listdir('./test_data/enron') if 'month' in file]
        length = len(files)
        graphsall = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/enron/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphsall.append(G)

        sample = graphsall[0].number_of_nodes()
        graphs = graphsall[-args.nsteps:]
        datafile = dataprep_util.prep_input_dynTriad(graphs, args.nsteps, args.
↪testDataType)
        # pdb.set_trace()

        embedding = dynamicTriad(niters=args.niters,
                                 starttime=args.starttime,
                                 datafile=datafile,
                                 batchsize=100,
                                 nsteps=args.nsteps,
                                 embdim=args.embdim,
                                 stepsize=args.stepsize,
                                 stepstride=args.stepstride,
                                 outdir=args.outdir,
                                 cachefn=args.cachefn,
```

(continues on next page)

(continued from previous page)

```
                                lr=args.lr,
                                beta=args.beta,
                                negdup=args.negdup,
                                datasetmod=args.datasetmod,
                                trainmod=args.trainmod,
                                pretrain_size=args.pretrain_size,
                                sampling_args=args.sampling_args,
                                validation=args.validation,
                                datatype=args.testDataType,
                                scale=args.scale,
                                classifier=args.classifier,
                                debug=args.debug,
                                test=args.test,
                                repeat=args.repeat,
                                resultdir=args.resultdir,
                                testDataType=args.testDataType,
                                clname='lr',
                                node_num=sample

                                )

        embedding.learn_embedding()
        embedding.get_embedding()

        outdir = args.resultdir
        if not os.path.exists(outdir):
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        outdir = outdir + '/dynTriad'
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        lp.expstaticLP_TRIAD(None,
                             graphs,
                             embedding,
                             1,
                             outdir + '/',
                             'l' + str(args.nsteps) + '_emb' + str(args.embdim) + '_
↪samples' + str(sample),
                             n_sample_nodes=sample
                             )
```

Total running time of the script: (0 minutes 0.000 seconds)

Note: Click *here* to download the full example code

4.9 Example Code for TIMERS

```
#!/usr/bin/env python
```

(continues on next page)

(continued from previous page)

```
# -*- coding: utf-8 -*-
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
disp_avlbl = True
if os.name == 'posix' and 'DISPLAY' not in os.environ:
    disp_avlbl = False
    import matplotlib
    matplotlib.use('Agg')

import matplotlib.pyplot as plt
import networkx as nx
import operator
from time import time
from argparse import ArgumentParser

from dynamicgem.embedding.TIMERS import TIMERS
from dynamicgem.utils import graph_util, plot_util, dataprep_util
from dynamicgem.evaluation import visualize_embedding as viz
from dynamicgem.utils.sdne_utils import *
from dynamicgem.evaluation import evaluate_graph_reconstruction as gr
from dynamicgem.evaluation import evaluate_link_prediction as lp
from dynamicgem.graph_generation import dynamic_SBM_graph


if __name__ == '__main__':

    parser = ArgumentParser(description='Learns static node embeddings')
    parser.add_argument('-t', '--testDataType',
                        default='sbm_cd',
                        type=str,
                        help='Type of data to test the code')
    parser.add_argument('-l', '--timelength',
                        default=5,
                        type=int,
                        help='Number of time series graph to generate')
    parser.add_argument('-nm', '--nodemigration',
                        default=5,
                        type=int,
                        help='number of nodes to migrate')
    parser.add_argument('-emb', '--embeddimension',
                        default=16,
                        type=float,
                        help='embedding dimension')
    parser.add_argument('-theta', '--theta',
                        default=0.5,  # 0.17
                        type=float,
                        help='a threshold for re-run SVD')
    parser.add_argument('-rdir', '--resultdir',
                        default='./results_link_all',  # 0.17
                        type=str,
                        help='directory for storing results')
    parser.add_argument('-sm', '--samples',
                        default=10,
                        type=int,
```

(continues on next page)

(continued from previous page)

```
                        help='samples for test data')
    parser.add_argument('-exp', '--exp',
                        default='lp',
                        type=str,
                        help='experiments (lp, emb)')

    args = parser.parse_args()
    dim_emb = args.embeddimension
    length = args.timelength
    theta = args.theta
    sample = args.samples

    if args.testDataType == 'sbm_cd':
        node_num = 100
        community_num = 2
        node_change_num = args.nodemigration
        dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                                 ␣
↪community_num,
                                                                                 ␣
↪length,
                                                                                  1,
                                                                                 node_
↪change_num)
        graphs = [g[0] for g in dynamic_sbm_series]

        datafile = dataprep_util.prep_input_TIMERS(graphs, length, args.testDataType)

        embedding = TIMERS(K=dim_emb,
                           Theta=theta,
                           datafile=datafile,
                           length=length,
                           nodemigration=args.nodemigration,
                           resultdir=args.resultdir,
                           datatype=args.testDataType
                           )
        outdir_tmp = './output'
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        outdir_tmp = outdir_tmp + '/sbm_cd'
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        if not os.path.exists(outdir_tmp + '/incrementalSVD'):
            os.mkdir(outdir_tmp + '/incrementalSVD')
        if not os.path.exists(outdir_tmp + '/rerunSVD'):
            os.mkdir(outdir_tmp + '/rerunSVD')
        if not os.path.exists(outdir_tmp + '/optimalSVD'):
            os.mkdir(outdir_tmp + '/optimalSVD')

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            embedding.learn_embedding()

            outdir = args.resultdir
```

(continues on next page)

(continued from previous page)

```
            if not os.path.exists(outdir):
                os.mkdir(outdir)
            outdir = outdir + '/' + args.testDataType
            if not os.path.exists(outdir):
                os.mkdir(outdir)

            embedding.get_embedding(outdir_tmp, 'incrementalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/incrementalSVD'
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(dynamic_sbm_series,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'nm' + str(args.nodemigration) + '_l' + str(length)␣
↪+ '_emb' + str(int(dim_emb)),
                                  )

            embedding.get_embedding(outdir_tmp, 'rerunSVD')
            outdir1 = outdir + '/rerunSVD'
            # embedding.plotresults()
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(dynamic_sbm_series,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'nm' + str(args.nodemigration) + '_l' + str(length)␣
↪+ '_emb' + str(int(dim_emb)),
                                  )

            embedding.get_embedding(outdir_tmp, 'optimalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/optimalSVD'
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(dynamic_sbm_series,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'nm' + str(args.nodemigration) + '_l' + str(length)␣
↪+ '_emb' + str(int(dim_emb)),
                                  )

    elif args.testDataType == 'academic':
        print("datatype:", args.testDataType)

        sample = args.samples
        if not os.path.exists('./test_data/academic/pickle'):
            os.mkdir('./test_data/academic/pickle')
            graphs, length = dataprep_util.get_graph_academic('./test_data/academic/
↪adjlist')
            for i in range(length):
```

(continues on next page)

(continued from previous page)

```
                nx.write_gpickle(graphs[i], './test_data/academic/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/academic/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/academic/pickle/' +␣
↪str(i)))

        G_cen = nx.degree_centrality(graphs[29])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        # pdb.set_trace()
        # node_l = np.random.choice(range(graphs[29].number_of_nodes()), 5000,␣
↪replace=False)
        # print(node_l)
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)
        # pdb.set_trace()
        graphs = graphs[-args.timelength:]

        datafile = dataprep_util.prep_input_TIMERS(graphs, args.timelength, args.
↪testDataType)

        embedding = TIMERS(K=dim_emb,
                           Theta=theta,
                           datafile=datafile,
                           length=args.timelength,
                           nodemigration=args.nodemigration,
                           resultdir=args.resultdir,
                           datatype=args.testDataType
                           )
        outdir_tmp = './output'
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        outdir_tmp = outdir_tmp + '/' + args.testDataType
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        if not os.path.exists(outdir_tmp + '/incrementalSVD'):
            os.mkdir(outdir_tmp + '/incrementalSVD')
        if not os.path.exists(outdir_tmp + '/rerunSVD'):
            os.mkdir(outdir_tmp + '/rerunSVD')
        if not os.path.exists(outdir_tmp + '/optimalSVD'):
            os.mkdir(outdir_tmp + '/optimalSVD')

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            embedding.learn_embedding()

            outdir = args.resultdir
            if not os.path.exists(outdir):
```

(continues on next page)

(continued from previous page)

```
            os.mkdir(outdir)
        outdir = outdir + '/' + args.testDataType
        if not os.path.exists(outdir):
            os.mkdir(outdir)

        embedding.get_embedding(outdir_tmp, 'incrementalSVD')
        # embedding.plotresults()
        outdir1 = outdir + '/incrementalSVD'
        if not os.path.exists(outdir1):
            os.mkdir(outdir1)
        lp.expstaticLP_TIMERS(None,
                              graphs,
                              embedding,
                              1,
                              outdir1 + '/',
                              'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                              n_sample_nodes=sample
                              )

        embedding.get_embedding(outdir_tmp, 'rerunSVD')
        outdir1 = outdir + '/rerunSVD'
        # embedding.plotresults()
        if not os.path.exists(outdir1):
            os.mkdir(outdir1)
        lp.expstaticLP_TIMERS(None,
                              graphs,
                              embedding,
                              1,
                              outdir1 + '/',
                              'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                              n_sample_nodes=sample
                              )

        embedding.get_embedding(outdir_tmp, 'optimalSVD')
        # embedding.plotresults()
        outdir1 = outdir + '/optimalSVD'
        if not os.path.exists(outdir1):
            os.mkdir(outdir1)
        lp.expstaticLP_TIMERS(None,
                              graphs,
                              embedding,
                              1,
                              outdir1 + '/',
                              'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                              n_sample_nodes=sample
                              )


    elif args.testDataType == 'hep':
        print("datatype:", args.testDataType)

        if not os.path.exists('./test_data/hep/pickle'):
            os.mkdir('./test_data/hep/pickle')
            files = [file for file in os.listdir('./test_data/hep/hep-th') if '.
↪gpickle' in file]
```

(continues on next page)

(continued from previous page)

```
            length = len(files)
            graphs = []
            for i in range(length):
                G = nx.read_gpickle('./test_data/hep/hep-th/month_' + str(i + 1) + '_
↪graph.gpickle')

                graphs.append(G)
            total_nodes = graphs[-1].number_of_nodes()

            for i in range(length):
                for j in range(total_nodes):
                    if j not in graphs[i].nodes():
                        graphs[i].add_node(j)

            for i in range(length):
                nx.write_gpickle(graphs[i], './test_data/hep/pickle/' + str(i))
        else:
            length = len(os.listdir('./test_data/hep/pickle'))
            graphs = []
            for i in range(length):
                graphs.append(nx.read_gpickle('./test_data/hep/pickle/' + str(i)))

        # pdb.set_trace()
        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        graphs = graphs[-args.timelength:]

        datafile = dataprep_util.prep_input_TIMERS(graphs, args.timelength, args.
↪testDataType)

        embedding = TIMERS(K=dim_emb,
                           Theta=theta,
                           datafile=datafile,
                           length=args.timelength,
                           nodemigration=args.nodemigration,
                           resultdir=args.resultdir,
                           datatype=args.testDataType
                           )
        outdir_tmp = './output'
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        outdir_tmp = outdir_tmp + '/' + args.testDataType
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        if not os.path.exists(outdir_tmp + '/incrementalSVD'):
            os.mkdir(outdir_tmp + '/incrementalSVD')
        if not os.path.exists(outdir_tmp + '/rerunSVD'):
```

(continues on next page)

(continued from previous page)

```
            os.mkdir(outdir_tmp + '/rerunSVD')
        if not os.path.exists(outdir_tmp + '/optimalSVD'):
            os.mkdir(outdir_tmp + '/optimalSVD')

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            embedding.learn_embedding()

            outdir = args.resultdir
            if not os.path.exists(outdir):
                os.mkdir(outdir)
            outdir = outdir + '/' + args.testDataType
            if not os.path.exists(outdir):
                os.mkdir(outdir)

            embedding.get_embedding(outdir_tmp, 'incrementalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/incrementalSVD'
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )

            embedding.get_embedding(outdir_tmp, 'rerunSVD')
            outdir1 = outdir + '/rerunSVD'
            # embedding.plotresults()
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )

            embedding.get_embedding(outdir_tmp, 'optimalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/optimalSVD'
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
```

(continues on next page)

(continued from previous page)

```
                                    'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                    n_sample_nodes=sample
                                    )

    elif args.testDataType == 'AS':
        print("datatype:", args.testDataType)

        files = [file for file in os.listdir('./test_data/AS/as-733') if '.gpickle'␣
↪in file]
        length = len(files)
        graphs = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/AS/as-733/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphs.append(G)

        sample = args.samples
        G_cen = nx.degree_centrality(graphs[-1])  # graph 29 in academia has highest␣
↪number of edges
        G_cen = sorted(G_cen.items(), key=operator.itemgetter(1), reverse=True)
        node_l = []
        i = 0
        while i < sample:
            node_l.append(G_cen[i][0])
            i += 1
        for i in range(length):
            graphs[i] = graph_util.sample_graph_nodes(graphs[i], node_l)

        graphs = graphs[-args.timelength:]

        datafile = dataprep_util.prep_input_TIMERS(graphs, args.timelength, args.
↪testDataType)

        embedding = TIMERS(K=dim_emb,
                           Theta=theta,
                           datafile=datafile,
                           length=args.timelength,
                           nodemigration=args.nodemigration,
                           resultdir=args.resultdir,
                           datatype=args.testDataType
                           )
        outdir_tmp = './output'
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        outdir_tmp = outdir_tmp + '/' + args.testDataType
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        if not os.path.exists(outdir_tmp + '/incrementalSVD'):
            os.mkdir(outdir_tmp + '/incrementalSVD')
        if not os.path.exists(outdir_tmp + '/rerunSVD'):
            os.mkdir(outdir_tmp + '/rerunSVD')
        if not os.path.exists(outdir_tmp + '/optimalSVD'):
            os.mkdir(outdir_tmp + '/optimalSVD')

        if args.exp == 'emb':
```

(continues on next page)

(continued from previous page)

```
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            embedding.learn_embedding()

            outdir = args.resultdir
            if not os.path.exists(outdir):
                os.mkdir(outdir)
            outdir = outdir + '/' + args.testDataType
            if not os.path.exists(outdir):
                os.mkdir(outdir)

            embedding.get_embedding(outdir_tmp, 'incrementalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/incrementalSVD'
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )

            embedding.get_embedding(outdir_tmp, 'rerunSVD')
            outdir1 = outdir + '/rerunSVD'
            # embedding.plotresults()
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )

            embedding.get_embedding(outdir_tmp, 'optimalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/optimalSVD'
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )
```

(continues on next page)

(continued from previous page)

```
    elif args.testDataType == 'enron':
        print("datatype:", args.testDataType)

        files = [file for file in os.listdir('./test_data/enron') if 'month' in file]
        length = len(files)
        # print(length)
        graphsall = []

        for i in range(length):
            G = nx.read_gpickle('./test_data/enron/month_' + str(i + 1) + '_graph.
↪gpickle')
            graphsall.append(G)

        sample = graphsall[0].number_of_nodes()
        graphs = graphsall[-args.timelength:]
        # pdb.set_trace()
        datafile = dataprep_util.prep_input_TIMERS(graphs, args.timelength, args.
↪testDataType)

        embedding = TIMERS(K=dim_emb,
                           Theta=theta,
                           datafile=datafile,
                           length=args.timelength,
                           nodemigration=args.nodemigration,
                           resultdir=args.resultdir,
                           datatype=args.testDataType
                           )
        outdir_tmp = './output'
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        outdir_tmp = outdir_tmp + '/' + args.testDataType
        if not os.path.exists(outdir_tmp):
            os.mkdir(outdir_tmp)
        if not os.path.exists(outdir_tmp + '/incrementalSVD'):
            os.mkdir(outdir_tmp + '/incrementalSVD')
        if not os.path.exists(outdir_tmp + '/rerunSVD'):
            os.mkdir(outdir_tmp + '/rerunSVD')
        if not os.path.exists(outdir_tmp + '/optimalSVD'):
            os.mkdir(outdir_tmp + '/optimalSVD')

        if args.exp == 'emb':
            print('plotting embedding not implemented!')

        if args.exp == 'lp':
            embedding.learn_embedding()

            outdir = args.resultdir
            if not os.path.exists(outdir):
                os.mkdir(outdir)
            outdir = outdir + '/' + args.testDataType
            if not os.path.exists(outdir):
                os.mkdir(outdir)

            embedding.get_embedding(outdir_tmp, 'incrementalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/incrementalSVD'
            if not os.path.exists(outdir1):
```

(continues on next page)

(continued from previous page)

```
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )

            embedding.get_embedding(outdir_tmp, 'rerunSVD')
            outdir1 = outdir + '/rerunSVD'
            # embedding.plotresults()
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )

            embedding.get_embedding(outdir_tmp, 'optimalSVD')
            # embedding.plotresults()
            outdir1 = outdir + '/optimalSVD'
            if not os.path.exists(outdir1):
                os.mkdir(outdir1)
            lp.expstaticLP_TIMERS(None,
                                  graphs,
                                  embedding,
                                  1,
                                  outdir1 + '/',
                                  'l' + str(args.timelength) + '_emb' + str(int(dim_
↪emb)) + '_samples' + str(sample),
                                  n_sample_nodes=sample
                                  )
```

Total running time of the script: (0 minutes 0.000 seconds)

CHAPTER

FIVE

INTRODUCTION

Graph embedding methods aim to represent each node of a graph in a low-dimensional vector space while preserving certain graph's properties. Such methods have been used to tackle many real-world tasks, e.g., friend recommendation in social networks, genome classification in biology networks, and visualizing topics in research using collaboration networks.

More recently, much attention has been devoted to extending static embedding techniques to capture graph evolution. Applications include temporal link prediction, and understanding the evolution dynamics of network communities. Most methods aim to efficiently update the embedding of the graph at each time step using information from previous embedding and from changes in the graph. Some methods also capture the temporal patterns of the evolution in the learned embedding, leading to improved link prediction performance.

In this library, we present an easy-to-use toolkit of state-of-the-art dynamic graph embedding methods. **dynamicgem** implements methods which can handle the evolution of networks over time. Further, we provide a comprehensive framework to evaluate the methods by providing support for four tasks on dynamic networks: graph reconstruction, static and temporal link prediction, node classification, and temporal visualization. For each task, our framework includes multiple evaluation metrics to quantify the performance of the methods. We further share synthetic and real networks for evaluation. Thus, our library is an end-to-end framework to experiment with dynamic graph embedding.

CHAPTER

SIX

IMPLEMENTED ALGORITHMS

Dynamic graph embedding algorithms aim to capture the dynamics of the network and its evolution. These methods are useful to predict the future behavior of the network, such as future connections within a network. The problem can be defined formally as follows.

Consider a weighted graph $G(V, E)$, with V and E as the set of vertices and edges respectively. Given an evolution of graph $\mathcal{G} = \{G_1, .., G_T\}$, where G_t represents the state of graph at time t, a dynamic graph embedding method aims to represent each node v in a series of low-dimensional vector space $y_{v_1}, \dots y_{v_t}$ by learning mappings $f_t : \{V_1, \dots, V_t, E_1, \dots E_t\} \rightarrow \mathbb{R}^d$ and $y_{v_i} = f_i(v_1, \dots, v_i, E_1, \dots E_i)$. The methods differ in the definition of f_t and the properties of the network preserved by f_t.

There are various existing state of the art methods trying to solve this problem that we have incorporated and included them in this python package including:

- Optimal SVD: This method decomposes adjacency matrix of the graph at each time step using Singular Value Decomposition (SVD) to represent each node using the d largest singular values.
- Incremental SVD: This method utilizes a perturbation matrix capturing the dynamics of the graphs along with performing additive modification on the SVD.
- Rerun SVD: This method uses incremental SVD to create the dynamic graph embedding. In addition to that, it uses a tolerance threshold to restart the optimal SVD calculations and avoid deviation in incremental graph embedding.
- Dynamic TRIAD: This method utilizes the triadic closure process to generate a graph embedding that preserves structural and evolution patterns of the graph.
- AEalign: This method uses deep auto-encoder to embed each node in the graph and aligns the embeddings at different time steps using a rotation matrix.
- dynGEM: This method utilizes deep auto-encoders to incrementally generate embedding of a dynamic graph at snapshot t.
- dyngraph2vecAE: This method models the interconnection of nodes within and across time using multiple fully connected layers.
- dyngraph2vecRNN: This method uses sparsely connected Long Short Term Memory (LSTM) networks to learn the embedding.
- dyngraph2vecAERNN: This method uses a fully connected encoder to initially acquire low dimensional hidden representation and feeds this representation into LSTMs to capture network dynamics.

CHAPTER

SEVEN

SOFTWARE ARCHITECTURE

dynamicegem package contains README files in dynamicegem and its sub directories including dynamicgem/dynamictriad, and dynamicgem/graph-generation directories containing explanation about the repository, its structure, setup, implemented methods, usage, dependencies, and other useful information for user guidance. The repository is organized in an easy to navigate manner. The subdirectories are organized based on the functionality which they serve as follows:

- dynamicegem/embedding: It contains implementation of the algorithms listed in section. In addition to dynamic graph embedding algorithms, this sub directory contains implementation for some static graph embedding methods on which the dynamic methods are built.
- dynamicegem/evaluation: It contains implementations of graph reconstruction, static and temporal link prediction and visualization for evaluation purposes.
- dynamicegem/utils: It contains implementation of utility functions for data preparation, plotting, embedding formatting, evaluation, and a variety of other functions that are building blocks of other functions.
- dynamicegem/graph-generation: It consists of functions to generate dynamic Stochastic Block Models (SBM) with diminishing community.
- dynamicegem/visualization: It contains functions for visualizing dynamic and static graph embeddings.
- dynamicegem/experiments: It contains useful hyper-parameter tuning function implementations.
- dynamicegem/test: It contains testing function used for coverage analysis, unit testing and functional testing.

CHAPTER

EIGHT

USING DYNAMICGEM

Please checkout the examples Examples

If more details are needed, have a look at the API Documentation.

CHAPTER

NINE

GRAPH EMBEDDING ALGORITHMS

9.1 AE Static

class dynamicgem.embedding.ae_static.**AE**(*d*, **hyper_dict*, ***kwargs*)

Auto-Encoder based static graph embedding.

AE is a static graph embedding method which can be used as a baseline for comparing the dynamic graph embedding methods. It uses the fully connected Nueral network as its encoder and decoder.

Parameters

- **d** (*int*) – dimension of the embedding
- **beta** (*float*) – penalty parameter in matrix B of 2nd order objective
- **nu1** (*float*) – L1-reg hyperparameter
- **nu2** (*float*) – L2-reg hyperparameter
- **K** (*float*) – number of hidden layers in encoder/decoder
- **n_units** (*list*) – vector of length K-1 containing #units in hidden layers of encoder/decoder, not including the units in the embedding layer
- **n_iter** (*int*) – number of sgd iterations for first embedding (const)
- **xeta** (*float*) – sgd step size parameter
- **n_batch** (*int*) – minibatch size for SGD
- **modelfile** (*str*) – Files containing previous encoder and decoder models
- **weightfile** (*str*) – Files containing previous encoder and decoder weights

Examples

```
>>> from dynamicgem.embedding.ae_static import AE
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 1000
>>> community_num = 2
>>> node_change_num = 10
>>> length =5
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
```

(continues on next page)

(continued from previous page)

```
                                                                        1,
                                                                        node_
↪change_num)
>>> embedding = AE(d=dim_emb,
                   beta=5,
                   nu1=1e-6,
                   nu2=1e-6,
                   K=3,
                   n_units=[500, 300, ],
                   n_iter=epochs,
                   xeta=1e-4,
                   n_batch=100,
                   modelfile=['./intermediate/enc_modelsbm.json',
                              './intermediate/dec_modelsbm.json'],
                   weightfile=['./intermediate/enc_weightssbm.hdf5',
                               './intermediate/dec_weightssbm.hdf5'])
```

```
>>> graphs = [g[0] for g in dynamic_sbm_series]
>>> embs = []
```

```
>>> for temp_var in range(length):
>>>         emb, _ = embedding.learn_embeddings(graphs[temp_var])
>>>         embs.append(emb)
```

get_edge_weight(*self*, *i*, *j*, *embed=None*, *filesuffix=None*)

Function to get edge weight.

i

source node for the edge.

Type int

j

target node for the edge.

Type int

embed

Embedding values of all the nodes.

Type Matrix

filesuffix

File suffix to be used to load the embedding.

Type str

Returns Weight of the given edge.

Return type Float

get_embedding(*self*, *filesuffix=None*)

Function to load the embedding values.

filesuffix

File suffix to be used to load the embedding.

Type str

Returns Numpy vector of embedding values

Return type Vector

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconst_from_embed(*self*, *embed*, *node_l=None*, *filesuffix=None*)
Function to reconstruct the graph from the embedding.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns REconstructed graph for the given nodes.

Return type List

get_reconstructed_adj(*self*, *embed=None*, *node_l=None*, *filesuffix=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embeddings(*self*, *graph=None*, *edge_f=None*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
Type Object

edge_f
Edge list
type List

Returns: List: Node embeddings and time taken by the algorithm

predict_next_adj(*self*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

9.2 DynamicAE (dyngraph2vecAE)

class dynamicgem.embedding.dynAE.**DynAE**(*d*, **hyper_dict*, ***kwargs*)
Dynamic AutoEncoder

DynAE is a dynamic graph embedding algorithm which also takes different timestep graph with varying look-back to be considered in embedding the nodes using the autoencoder.

Parameters

- **d** (*int*) – dimension of the embedding
- **beta** (*float*) – penalty parameter in matrix B of 2nd order objective
- **n_prev_graphs** (*int*) – Lookback (number of previous graphs to be considered) for the dynamic graph embedding
- **nu1** (*float*) – L1-reg hyperparameter
- **nu2** (*float*) – L2-reg hyperparameter
- **K** (*float*) – number of hidden layers in encoder/decoder
- **rho** (*float*) – bounding ratio for number of units in consecutive layers (< 1)
- **n_units** (*list*) – vector of length K-1 containing #units in hidden layers of encoder/decoder, not including the units in the embedding layer
- **n_iter** (*int*) – number of sgd iterations for first embedding (const)
- **xeta** (*float*) – sgd step size parameter
- **n_batch** (*int*) – minibatch size for SGD
- **modelfile** (*str*) – Files containing previous encoder and decoder models
- **weightfile** (*str*) – Files containing previous encoder and decoder weights

Examples

```
>>> from dynamicgem.embedding.dynAE import DynAE
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 1000
>>> community_num = 2
>>> node_change_num = 10
>>> length =5
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
```

(continues on next page)

(continued from previous page)

```
                                                                        community_
↪num,
                                                                        length,
                                                                        1,
                                                                        node_
↪change_num)
>>> embedding = DynAE(d=dim_emb,
                      beta=5,
                      n_prev_graphs=lookback,
                      nu1=1e-6,
                      nu2=1e-6,
                      n_units=[500, 300, ],
                      rho=0.3,
                      n_iter=epochs,
                      xeta=args.learningrate,
                      n_batch=args.batch,
                      modelfile=['./intermediate/enc_model.json', './intermediate/dec_
↪model.json'],
                      weightfile=['./intermediate/enc_weights.hdf5', './intermediate/
↪dec_weights.hdf5'],
                      savefilesuffix="testing")
```

```
>>> graphs = [g[0] for g in dynamic_sbm_series]
>>> embs = []
```

```
>>> for temp_var in range(length):
>>>         emb, _ = embedding.learn_embeddings(graphs[temp_var])
>>>         embs.append(emb)
```

get_edge_weight(*self*, *i*, *j*, *embed=None*, *filesuffix=None*)

Function to get edge weight.

i

source node for the edge.

Type int

j

target node for the edge.

Type int

embed

Embedding values of all the nodes.

Type Matrix

filesuffix

File suffix to be used to load the embedding.

Type str

Returns Weight of the given edge.

Return type Float

get_embeddings(*self*)

Function to return the embeddings

get_method_name(*self*)

Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconst_from_embed(*self*, *embed*, *filesuffix=None*)
Function to reconstruct the graph from the embedding.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns REconstructed graph for the given nodes.

Return type List

get_reconstructed_adj(*self*, *embed=None*, *node_l=None*, *filesuffix=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embeddings(*self*, *graphs*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

predict_next_adj(*self*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

9.3 DynamicAERNN (dyngraph2vecAERNN)

class dynamicgem.embedding.dynAERNN.**DynAERNN**(*d*, **hyper_dict*, ***kwargs*)
Dynamic AutoEncoder with Recurrent Neural Network

dyngraph2vecAERNN or DynAERNN is a dynamic graph embedding algorithm which combines the auto-encoder with the recurrent neural network to perform the embedding for the temporally evolving graphs.

Parameters

- **d** (*int*) – dimension of the embedding
- **beta** (*float*) – penalty parameter in matrix B of 2nd order objective
- **n_prev_graphs** (*int*) – Lookback (number of previous graphs to be considered) for the dynamic graph embedding
- **nu1** (*float*) – L1-reg hyperparameter
- **nu2** (*float*) – L2-reg hyperparameter
- **K** (*float*) – number of hidden layers in encoder/decoder
- **rho** (*float*) – bounding ratio for number of units in consecutive layers (< 1)
- **n_aeunits** (*list*) –
- **List of embedding dimension for lstm layers** (*n_lstmunits=*) –
- **n_iter** (*int*) – number of sgd iterations for first embedding (const)
- **xeta** (*float*) – sgd step size parameter
- **n_batch** (*int*) – minibatch size for SGD
- **modelfile** (*str*) – Files containing previous encoder and decoder models
- **weightfile** (*str*) – Files containing previous encoder and decoder weights

Examples

```
>>> from dynamicgem.embedding.dynAERNN import DynAERNN
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 1000
>>> community_num = 2
>>> node_change_num = 10
>>> length =5
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
```

(continues on next page)

(continued from previous page)

```
                                                                        1,
                                                                        node_
↪change_num)
>>> embedding = DynAERNN(d=dim_emb,
                         beta=5,
                         n_prev_graphs=lookback,
                         nu1=1e-6,
                         nu2=1e-6,
                         n_units=[500, 300, ],
                         rho=0.3,
                         n_iter=epochs,
                         xeta=args.learningrate,
                         n_batch=args.batch,
                         modelfile=['./intermediate/enc_model.json', './intermediate/dec_
↪model.json'],
                         weightfile=['./intermediate/enc_weights.hdf5', './intermediate/
↪dec_weights.hdf5'],
                         savefilesuffix="testing")
```

```
>>> graphs = [g[0] for g in dynamic_sbm_series]
>>> embs = []
```

```
>>> for temp_var in range(length):
>>>         emb, _ = embedding.learn_embeddings(graphs[temp_var])
>>>         embs.append(emb)
```

get_edge_weight(*self*, *i*, *j*, *embed=None*, *filesuffix=None*)
Function to get edge weight.

i
source node for the edge.
Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embeddings(*self*)
Function to return the embeddings

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconst_from_embed(*self*, *embed*, *filesuffix=None*)
Function to reconstruct the graph from the embedding.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns REconstructed graph for the given nodes.

Return type List

get_reconstructed_adj(*self*, *embed=None*, *node_l=None*, *filesuffix=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embeddings(*self*, *graphs*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

predict_next_adj(*self*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

9.4 Dynamic TRIAD

class dynamicgem.embedding.dynamicTriad.**dynamicTriad**(**hyper_dict*, ***kwargs*)

Dynamic Triad Closure based embedding

DynamicTriad preserves both structural informa- tion and evolution patterns of a given network. The general idea of our approach is to impose triad, which is a group of three vertices and is one of the basic units of networks.

Parameters

- **niters** (*int*) – Number of iteration to run the algorithm
- **starttime** (*int*) – start time for the graph step
- **datafile** (*str*) – The file for the input graph
- **batchsize** (*int*) – batch size for training the algorithm
- **nsteps** (*int*) – total number of steps in the temporal graph
- **embdim** (*int*) – embedding dimension
- **stepsize** (*int*) – step size for the graph
- **stepstride** (*int*) – stride to consider for temporal stride
- **outdir** (*str*) – The output directory to store the result
- **cachefn** (*str*) – Directory to cache the temporary data
- **lr** (*float*) – Learning rate for the algorithm
- **beta** (*float*) – coefficients for triad component
- **negdup** (*float*) – neg/pos ratio during sampling
- **datasetmod** (*str*) – module name for dataset loading
- **trainmod** (*str*) – module name for training model
- **pretrain_size** (*int*) – size of the graph for pre-training
- **sampling_args** (*int*) – sampling size
- **validation** (*list*) – link_reconstruction validation data
- **datatype** (*str*) – type of network data
- **scale** (*int*) – scaling
- **classifier** (*str*) – type of classifier to be used
- **debug** (*bool*) – debugging flag
- **test** (*bool*) – type of test to perform
- **repeat** (*int*) – Number of times to repeat the learning
- **resultdir** (*str*) – directory to store the result
- **testDataType** (*str*) – type of test data

- **clname** (*str*) – classifier type
- **node_num** (*int*) – number of nodes

Examples

```
>>> from dynamicgem.embedding.dynamicTriad import dynamicTriad
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 200
>>> community_num = 2
>>> node_change_num = 2
>>> length =5
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
                                                                            1,
                                                                            node_
↪change_num)
>>> graphs = [g[0] for g in dynamic_sbm_series]
```

```
>>> datafile = dataprep_util.prep_input_dynTriad(graphs, length, args.
↪testDataType)
```

```
>>> embedding = dynamicTriad(niters=10,
                             starttime=0,
                             datafile=datafile,
                             batchsize=10,
                             nsteps=5,
                             embdim=16,
                             stepsize=1,
                             stepstride=1,
                             outdir='./output',
                             cachefn='./tmp',
                             lr=0.001,
                             beta=0.1,
                             negdup=1,
                             datasetmod='dynamicgem.utils.dynamictriad_utils.dataset.
↪adjlist',
                             trainmod='dynamicgem.utils.dynamictriad_utils.algorithm.
↪dynamic_triad',
                             pretrain_size=4,
                             sampling_args={},
                             validation='link_reconstruction',
                             datatype='sbm_cd',
                             scale=1,
                             classifier='lr',
                             debug=False,
                             test='link_predict',
                             repeat=1,
                             resultdir='./results_link_all',
                             testDataType='sbm_cd',
                             clname='lr',
                             node_num=node_num )
```

```
>>> embedding.learn_embedding()
>>> embedding.get_embedding()
>>> outdir = args.resultdir
>>> if not os.path.exists(outdir):
>>>     os.mkdir(outdir)
>>> outdir = outdir + '/' + args.testDataType
>>> if not os.path.exists(outdir):
>>>     os.mkdir(outdir)
>>> outdir = outdir + '/' + 'dynTRIAD'
>>> if not os.path.exists(outdir):
 >>>     os.mkdir(outdir)
```

```
>>> lp.expstaticLP_TRIAD(dynamic_sbm_series,
                         graphs,
                         embedding,
                         1,
                         outdir + '/',
                         'nm' + str(args.nodemigration) + '_l' + str(args.nsteps) + '_
↪emb' + str(args.embdim),
                         )
```

class ResultPresenter
result presenter class

export (*self*, *vertices*, *data*, *outdir*)
function to export the data

get_edge_weight (*self*, *t*, *i*, *j*)
Function to get edge weight.

i
source node for the edge.
Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embedding (*self*)
Function to return the embeddings

get_method_name (*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconstructed_adj(*self*, *t*, *X=None*, *node_l=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

X
Embedding values of all the nodes.
Type Matrix

t
Time step
Type int

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*)
Learns the embedding of the nodes.

Returns Node embeddings and time taken by the algorithm

Return type List

link_predict(*self*, *g*, *t*, *intv=0*, *repeat=1*)
Function to perform link prediction

load_datamod(*self*, *modname*)
Function to load the dataset module

load_embedding(*self*, *fn*, *vs*)
Function to load the embedding

load_or_update_cache(*self*, *ds*, *cachefn*)
Function to either update or load the cache

load_trainmod(*self*, *modname*)
Function to load the training module

plotresults(*self*, *dynamic_sbm_series*)
Function to plot the result

predict_next_adj(*self*, *t*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

sample_link_reconstruction(*self*, *g*, *sample_nodes=None*, *negdup=1*)
Function to sample the link reconstruction

9.5 Dynamic GEM (dynGEM)

class dynamicgem.embedding.dynGEM.**DynGEM**(**hyper_dict*, ***kwargs*)
Structural Deep Network Embedding

DynSDNE (also DynGEM) perfomr the dynamic network embedding while utilizing Structural Deep Network Embedding (SDNE) with dynamically evolving graphs as input.

Parameters

- **d** (*int*) – dimension of the embedding
- **beta** (*float*) – penalty parameter in matrix B of 2nd order objective
- **n_prev_graphs** (*int*) – Lookback (number of previous graphs to be considered) for the dynamic graph embedding
- **nu1** (*float*) – L1-reg hyperparameter
- **nu2** (*float*) – L2-reg hyperparameter
- **K** (*float*) – number of hidden layers in encoder/decoder
- **rho** (*float*) – bounding ratio for number of units in consecutive layers (< 1)
- **n_aeunits** (*list*) –
- **List of embedding dimension for lstm layers** (*n_lstmunits=*) –
- **n_iter** (*int*) – number of sgd iterations for first embedding (const)
- **xeta** (*float*) – sgd step size parameter
- **n_batch** (*int*) – minibatch size for SGD
- **modelfile** (*str*) – Files containing previous encoder and decoder models
- **weightfile** (*str*) – Files containing previous encoder and decoder weights

Examples

```
>>> from dynamicgem.embedding.dynSDNE import DynSDNE
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 1000
>>> community_num = 2
>>> node_change_num = 10
>>> length =5
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
                                                                            1,
                                                                            node_
↪change_num)
>>> graphs = [g[0] for g in dynamic_sbm_series]
>>> embedding = DynSDNE(d=128, beta=5, alpha=0, nu1=1e-6, nu2=1e-6, K=3,
```

(continues on next page)

(continued from previous page)

```
            n_units=[500, 300], n_iter=20, xeta=0.01,
            n_batch=500,
            modelfile=['./intermediate/enc_model.json',
                       './intermediate/dec_model.json'],
            weightfile=['./intermediate/enc_weights.hdf5',
                        './intermediate/dec_weights.hdf5'])
>>> embedding.learn_embedding(graph=graphs._graph, edge_f=None,
                      is_weighted=True, no_python=True)
```

get_edge_weight(*self*, *i*, *j*, *embed=None*, *filesuffix=None*)
Function to get edge weight.

i
source node for the edge.
Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embedding(*self*, *filesuffix=None*)
Function to return the embeddings

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconst_from_embed(*self*, *embed*, *node_l=None*, *filesuffix=None*)
Function to reconstruct the graph from the embedding.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns REconstructed graph for the given nodes.

Return type List

get_reconstructed_adj(*self*, *embed=None*, *node_l=None*, *filesuffix=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

9.6 Dynamic RNN (dyngraph2vecRNN)

class dynamicgem.embedding.dynRNN.**DynRNN**(*d*, **hyper_dict*, ***kwargs*)
Dynamic embedding with Recurrent Neural Network

dyngraph2vecRNN or DynRNN is a dynamic graph embedding algorithm which uses the recurrent neural network to perform the embedding for the temporally evolving graphs.

Parameters

- **d** (*int*) – dimension of the embedding
- **beta** (*float*) – penalty parameter in matrix B of 2nd order objective
- **n_prev_graphs** (*int*) – Lookback (number of previous graphs to be considered) for the dynamic graph embedding
- **nu1** (*float*) – L1-reg hyperparameter
- **nu2** (*float*) – L2-reg hyperparameter
- **K** (*float*) – number of hidden layers in encoder/decoder
- **rho** (*float*) – bounding ratio for number of units in consecutive layers (< 1)

- **n_enc_units** (*list*) –
- **n_dec_units** (*list*) –
- **n_iter** (*int*) – number of sgd iterations for first embedding (const)
- **xeta** (*float*) – sgd step size parameter
- **n_batch** (*int*) – minibatch size for SGD
- **modelfile** (*str*) – Files containing previous encoder and decoder models
- **weightfile** (*str*) – Files containing previous encoder and decoder weights

Examples

```
>>> from dynamicgem.embedding.dynRNN import DynRNN
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 1000
>>> community_num = 2
>>> node_change_num = 10
>>> length =5
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
                                                                            1,
                                                                            node_
↪change_num)
>>> embedding = DynRNN(d=dim_emb,
                       beta=5,
                       n_prev_graphs=lookback,
                       nu1=1e-6,
                       nu2=1e-6,
                       n_units=[500, 300, ],
                       rho=0.3,
                       n_iter=epochs,
                       xeta=args.learningrate,
                       n_batch=args.batch,
                       modelfile=['./intermediate/enc_model.json', './intermediate/dec_
↪model.json'],
                       weightfile=['./intermediate/enc_weights.hdf5', './intermediate/
↪dec_weights.hdf5'],
                       savefilesuffix="testing")
```

```
>>> graphs = [g[0] for g in dynamic_sbm_series]
>>> embs = []
```

```
>>> for temp_var in range(length):
>>>         emb, _ = embedding.learn_embeddings(graphs[temp_var])
>>>         embs.append(emb)
```

get_edge_weight(*self*, *i*, *j*, *embed=None*, *filesuffix=None*)

Function to get edge weight.

i

source node for the edge.

Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embeddings(*self*)
Function to return the embeddings

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconst_from_embed(*self*, *embed*, *filesuffix=None*)
Function to reconstruct the graph from the embedding.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns REconstructed graph for the given nodes.

Return type List

get_reconstructed_adj(*self*, *embed=None*, *node_l=None*, *filesuffix=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embeddings(*self*, *graphs*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

predict_next_adj(*self*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

9.7 Dynamic Graph Factorization

class dynamicgem.embedding.graphFac_dynamic.**GraphFactorization**(*d*, *n_iter*, *n_iter_sub*, *eta*, *regu*, *kappa*, *initEmbed=None*)

Graph Facgorization based network embedding

It utilizes factorization based method to acquire the embedding of the graph nodes.

Parameters

- **d** (*int*) – dimension of the embedding
- **eta** (*float*) – learning rate of sgd
- **regu** (*float*) – regularization coefficient of magnitude of weights
- **beta** (*float*) – penalty parameter in matrix B of 2nd order objective
- **n_iter** (*int*) – number of sgd iterations for first embedding (const)
- **method_name** (*str*) – method name
- **initEmbed** (*Matrix*) – Previous timestep embedding initialized for the current timestep

Examples

```
>>> from dynamicgem.embedding.graphFac_dynamic import GraphFactorization
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 100
>>> community_num = 2
>>> node_change_num = 2
>>> length = 5
>>> dynamic_sbm_series = dynamic_SBM_graph.get_random_perturbation_series_v2(node_
↪num, community_num, length,
                                                                              node_change_
↪num)
>>> dynamic_embeddings = GraphFactorization(100, 100, 10, 5 * 10 ** -2, 1.0, 1.0)
>>> dynamic_embeddings.learn_embeddings([g[0] for g in dynamic_sbm_series])
```

```
>>> plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding(dynamic_embeddings.get_
↪embeddings(), dynamic_sbm_series)
>>> plt.show()
```

getFVal (*self*, *adj_mtx*, *X*, *prev_step_emb=None*)
Function to Factorize the adjacency matrix.

get_edge_weight (*self*, *i*, *j*)
Function to get edge weight.

i
source node for the edge.
Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embedding (*self*)
Function to return a single graph embedding

get_embeddings (*self*)
Function to return the whole tempral graphs embeddings

get_method_name (*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary (*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconstructed_adj(*self*, *X=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*, *graph*, *prevEmbed=None*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

learn_embeddings(*self*, *graphs*, *prevStepInfo=False*)
Learning the graph embedding from the adjcency matrix. :param graphs: the graphs to embed in networkx DiGraph format

9.8 Dynamic SDNE

class dynamicgem.embedding.sdne_dynamic.**SDNE**(*d*, *beta*, *alpha*, *nu1*, *nu2*, *K*, *n_units*, *rho*, *n_iter*, *n_iter_subs*, *xeta*, *n_batch*, *modelfile=None*, *weightfile=None*, *node_frac=1*, *n_walks_per_node=5*, *len_rw=2*)
Structural Deep Network Embedding

SDNE perform the network embedding while utilizing Structural Deep Network Embedding (SDNE).

Parameters

- **d** (*int*) – dimension of the embedding
- **beta** (*float*) – penalty parameter in matrix B of 2nd order objective
- **n_prev_graphs** (*int*) – Lookback (number of previous graphs to be considered) for the dynamic graph embedding
- **nu1** (*float*) – L1-reg hyperparameter
- **nu2** (*float*) – L2-reg hyperparameter
- **K** (*float*) – number of hidden layers in encoder/decoder
- **rho** (*float*) – bounding ratio for number of units in consecutive layers (< 1)

- **n_aeunits** (*list*) –
- **List of embedding dimension for lstm layers** (*n_lstmunits=*) –
- **n_iter** (*int*) – number of sgd iterations for first embedding (const)
- **xeta** (*float*) – sgd step size parameter
- **n_batch** (*int*) – minibatch size for SGD
- **modelfile** (*str*) – Files containing previous encoder and decoder models
- **weightfile** (*str*) – Files containing previous encoder and decoder weights
- **node_frac** (*float*) – Fraction of nodes to use for random walk
- **n_walks_per_node** (*int*) – Number of random walks to do for each selected nodes
- **len_rw** (*int*) – Length of every random walk

Examples

```
>>> from dynamicgem.embedding.sdne_dynamic import SDNE
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 100
>>> community_num = 2
>>> node_change_num = 2
>>> length = 2
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series(node_num,
 ↪ community_num, length, 1, node_change_num)
>>> dynamic_embedding = SDNE(d=16, beta=5, alpha=1e-5, nu1=1e-6, nu2=1e-6, K=3, n_
 ↪units=[500, 300,], rho=0.3, n_iter=2, n_iter_subs=5, xeta=0.01, n_batch=500,␣
 ↪modelfile=['./intermediate/enc_model.json', './intermediate/dec_model.json'],␣
 ↪weightfile=['./intermediate/enc_weights.hdf5', './intermediate/dec_weights.hdf5
 ↪'], node_frac=1, n_walks_per_node=10, len_rw=2)
>>> dynamic_embedding.learn_embeddings([g[0] for g in dynamic_sbm_series], False,␣
 ↪subsample=False)
>>> plot_dynamic_sbm_embedding.plot_dynamic_sbm_embedding(dynamic_embedding.get_
 ↪embeddings(), dynamic_sbm_series)
>>> plt.savefig('result/visualization_sdne_cd.png')
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*, *embed=None*, *filesuffix=None*)

Function to get edge weight.

i

source node for the edge.

Type int

j

target node for the edge.

Type int

embed

Embedding values of all the nodes.

Type Matrix

filesuffix

File suffix to be used to load the embedding.

Type str

Returns Weight of the given edge.

Return type Float

get_embedding(*self*, *filesuffix*)
Function to return sinfle embedding

get_embeddings(*self*)
Function to return all the embeddings

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconst_from_embed(*self*, *embed*, *filesuffix=None*)
Function to reconstruct the graph from the embedding.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns REconstructed graph for the given nodes.

Return type List

get_reconstructed_adj(*self*, *embed=None*, *filesuffix=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*, *graph*, *prevStepInfo=True*, *loadfilesuffix=None*, *savefilesuffix=None*, *subsample=False*)

Learns the embedding of the nodes.

graph
Networkx Graph Object
Type Object

prevStepInfo
Incorporate previous step info
Type bool

loadfilesuffix
file suffix for loading the previos data
Type str

savefilesuffix
file suffix to be used for saving the data
type str

Returns: List: Node embeddings and time taken by the algorithm

learn_embeddings(*self*, *graphs*, *prevStepInfo=False*, *loadsuffixinfo=None*, *savesuffixinfo=None*, *subsample=False*)

Learns the embedding of the nodes.

graph
Networkx Graph Object
Type Object

prevStepInfo
Incorporate previous step info
Type bool

loadfilesuffix
file suffix for loading the previos data
Type str

savefilesuffix
file suffix to be used for saving the data
type str

Returns: List: Node embeddings and time taken by the algorithm

9.9 TIMERS

class dynamicgem.embedding.TIMERS.**TIMERS**(**hyper_dict*, ***kwargs*)

Timers: Dynamic graph embedding

Timers perfomrs dynamic graph embedding by utilizing the SVDS decomposition of incremental graph.

Parameters **Args** – K (int): dimension of the embedding theta (float): threshold for rerun datafile (str): location of the data file length (int) : total timesteps of the data nodemigraiton (int): number of nodes to migrate for sbm_cd datatype resultdir (str): directory to save the result datatype (str): sbm_cd, enron, academia, hep, AS

Examples

```
>>> from dynamicgem.embedding.TIMERS import TIMERS
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 100
>>> community_num = 2
>>> node_change_num = 2
>>> length =5
>>> resultdir='./results_link_all'
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
                                                                            1,
                                                                            node_
↪change_num)
>>> graphs = [g[0] for g in dynamic_sbm_series]
```

```
>>> datafile = dataprep_util.prep_input_TIMERS(graphs, length, args.testDataType)
```

```
>>> embedding = TIMERS(K=16,
                       Theta=0.5,
                       datafile=datafile,
                       length=length,
                       nodemigration=node_change_num,
                       resultdir=resultdir,
                       datatype='sbm_cd'
                       )
>>> outdir_tmp = './output'
>>> if not os.path.exists(outdir_tmp):
>>>     os.mkdir(outdir_tmp)
>>> outdir_tmp = outdir_tmp + '/sbm_cd'
>>> if not os.path.exists(outdir_tmp):
>>>    os.mkdir(outdir_tmp)
>>> if not os.path.exists(outdir_tmp + '/incrementalSVD'):
>>>    os.mkdir(outdir_tmp + '/incrementalSVD')
>>> if not os.path.exists(outdir_tmp + '/rerunSVD'):
>>>     os.mkdir(outdir_tmp + '/rerunSVD')
>>> if not os.path.exists(outdir_tmp + '/optimalSVD'):
>>>     os.mkdir(outdir_tmp + '/optimalSVD')
```

```
>>>  embedding.learn_embedding()
```

```
>>>  outdir = resultdir
>>>  if not os.path.exists(outdir):
>>>     os.mkdir(outdir)
>>>  outdir = outdir + '/' + args.testDataType
>>>  if not os.path.exists(outdir):
>>>      os.mkdir(outdir)
```

```
>>>  embedding.get_embedding(outdir_tmp, 'incrementalSVD')
     # embedding.plotresults()
>>>  outdir1 = outdir + '/incrementalSVD'
>>>  if not os.path.exists(outdir1):
>>>      os.mkdir(outdir1)
```

(continues on next page)

(continued from previous page)

```
>>>  lp.expstaticLP_TIMERS(dynamic_sbm_series,
                           graphs,
                           embedding,
                           1,
                           outdir1 + '/',
                           'nm' + str(args.nodemigration) + '_l' + str(length) + '_
↪emb' + str(int(dim_emb)),
                           )
```

```
>>>  embedding.get_embedding(outdir_tmp, 'rerunSVD')
>>>  outdir1 = outdir + '/rerunSVD'
    # embedding.plotresults()
>>>  if not os.path.exists(outdir1):
>>>      os.mkdir(outdir1)
>>>  lp.expstaticLP_TIMERS(dynamic_sbm_series,
                           graphs,
                           embedding,
                           1,
                           outdir1 + '/',
                           'nm' + str(args.nodemigration) + '_l' + str(length) + '_
↪emb' + str(int(dim_emb)),
                           )
```

```
>>>  embedding.get_embedding(outdir_tmp, 'optimalSVD')
    # embedding.plotresults()
>>>  outdir1 = outdir + '/optimalSVD'
>>>  if not os.path.exists(outdir1):
>>>      os.mkdir(outdir1)
>>>   lp.expstaticLP_TIMERS(dynamic_sbm_series,
                           graphs,
                           embedding,
                           1,
                           outdir1 + '/',
                           'nm' + str(args.nodemigration) + '_l' + str(length) + '_
↪emb' + str(int(dim_emb)),
                           )
```

get_edge_weight (*self*, *t*, *i*, *j*)

Function to get edge weight.

i

source node for the edge.

Type int

j

target node for the edge.

Type int

embed

Embedding values of all the nodes.

Type Matrix

filesuffix

File suffix to be used to load the embedding.

Type str

Returns Weight of the given edge.

Return type Float

get_embedding(*self*, *outdir_tmp*, *method*)
Function to return the embeddings

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconstructed_adj(*self*, *t*, *X=None*, *node_l=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*, *graph=None*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

plotresults(*self*, *dynamic_sbm_series*)
Function to plot the results

predict_next_adj(*self*, *t*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

9.10 incremental SVD

class dynamicgem.embedding.incrementalSVD.**incSVD**(**hyper_dict*, ***kwargs*)

Incremental Singular Value Decomposition

Utilizes the incremental SVD decomposition to acquire the embedding of the nodes.

Parameters **Args** – K (int): dimension of the embedding theta (float): threshold for rerun datafile (str): location of the data file length (int) : total timesteps of the data nodemigraiton (int): number of nodes to migrate for sbm_cd datatype resultdir (str): directory to save the result datatype (str): sbm_cd, enron, academia, hep, AS

Examples

```
>>> from dynamicgem.embedding.incrementalSVD import incSVD
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 100
>>> community_num = 2
>>> node_change_num = 2
>>> length =5
>>> resultdir='./results_link_all'
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
                                                                            1,
                                                                            node_
↪change_num)
>>> graphs = [g[0] for g in dynamic_sbm_series]
```

```
>>> datafile = dataprep_util.prep_input_TIMERS(graphs, length, args.testDataType)
```

```
>>> embedding = incSVD(K=16,
                       Theta=0.5,
                       datafile=datafile,
                       length=length,
                       nodemigration=node_change_num,
                       resultdir=resultdir,
                       datatype='sbm_cd'
                       )
>>> outdir_tmp = './output'
>>> if not os.path.exists(outdir_tmp):
>>>     os.mkdir(outdir_tmp)
>>> outdir_tmp = outdir_tmp + '/sbm_cd'
>>> if not os.path.exists(outdir_tmp):
>>>    os.mkdir(outdir_tmp)
>>> if not os.path.exists(outdir_tmp + '/incrementalSVD'):
>>>    os.mkdir(outdir_tmp + '/incrementalSVD')
>>>  embedding.learn_embedding()
>>>  outdir = resultdir
>>>  if not os.path.exists(outdir):
>>>     os.mkdir(outdir)
>>>  outdir = outdir + '/' + args.testDataType
>>>  if not os.path.exists(outdir):
```

(continues on next page)

(continued from previous page)

```
>>>      os.mkdir(outdir)
>>>  embedding.get_embedding(outdir_tmp, 'incrementalSVD')
     # embedding.plotresults()
>>>  outdir1 = outdir + '/incrementalSVD'
>>>  if not os.path.exists(outdir1):
>>>      os.mkdir(outdir1)
>>>  lp.expstaticLP_TIMERS(dynamic_sbm_series,
                          graphs,
                          embedding,
                          1,
                          outdir1 + '/',
                          'nm' + str(args.nodemigration) + '_l' + str(length) + '_
↪emb' + str(int(dim_emb)),
                          )
```

get_edge_weight (*self*, *t*, *i*, *j*)
Function to get edge weight.

i
source node for the edge.
Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embedding (*self*, *outdir_tmp*, *method*)
Function to return the embeddings

get_method_name (*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary (*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconstructed_adj (*self*, *t*, *X=None*, *node_l=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.

Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*, *graph=None*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

plotresults(*self*, *dynamic_sbm_series*)
Function to plot the results

predict_next_adj(*self*, *t*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

9.11 rerunSVD

class dynamicgem.embedding.rerunSVD.**rerunSVD**(**hyper_dict*, ***kwargs*)
Timers: rerun Singular Value Decomposition

Timers perfomrs dynamic graph embedding by utilizing the SVDS decomposition of incremental graph with a bound to trigger the update.

Parameters **Args** – K (int): dimension of the embedding theta (float): threshold for rerun datafile (str): location of the data file length (int) : total timesteps of the data nodemigraiton (int): number of nodes to migrate for sbm_cd datatype resultdir (str): directory to save the result datatype (str): sbm_cd, enron, academia, hep, AS

Examples

```
>>> from dynamicgem.embedding.rerunSVD import rerunSVD
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 100
>>> community_num = 2
```

(continues on next page)

(continued from previous page)

```
>>> node_change_num = 2
>>> length =5
>>> resultdir='./results_link_all'
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
                                                                            1,
                                                                            node_
↪change_num)
>>> graphs = [g[0] for g in dynamic_sbm_series]
```

```
>>> datafile = dataprep_util.prep_input_TIMERS(graphs, length, args.testDataType)
```

```
>>> embedding = rerunSVD(K=16,
                         Theta=0.5,
                         datafile=datafile,
                         length=length,
                         nodemigration=node_change_num,
                         resultdir=resultdir,
                         datatype='sbm_cd'
                         )
>>> outdir_tmp = './output'
>>> if not os.path.exists(outdir_tmp):
>>>     os.mkdir(outdir_tmp)
>>> outdir_tmp = outdir_tmp + '/sbm_cd'
>>> if not os.path.exists(outdir_tmp):
>>>    os.mkdir(outdir_tmp)
>>> if not os.path.exists(outdir_tmp + '/rerunSVD'):
>>>     os.mkdir(outdir_tmp + '/rerunSVD')
>>>  embedding.learn_embedding()
>>>  outdir = resultdir
>>>  if not os.path.exists(outdir):
>>>     os.mkdir(outdir)
>>>  outdir = outdir + '/' + args.testDataType
>>>  if not os.path.exists(outdir):
>>>      os.mkdir(outdir)
>>>  embedding.get_embedding(outdir_tmp, 'rerunSVD')
>>>  outdir1 = outdir + '/rerunSVD'
    # embedding.plotresults()
>>>  if not os.path.exists(outdir1):
>>>      os.mkdir(outdir1)
>>>  lp.expstaticLP_TIMERS(dynamic_sbm_series,
                           graphs,
                           embedding,
                           1,
                           outdir1 + '/',
                           'nm' + str(args.nodemigration) + '_l' + str(length) + '_
↪emb' + str(int(dim_emb)),
                           )
```

get_edge_weight (*self*, *t*, *i*, *j*)

Function to get edge weight.

i

source node for the edge.

Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embedding(*self*, *outdir_tmp*, *method*)
Function to return the embeddings

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconstructed_adj(*self*, *t*, *X=None*, *node_l=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*, *graph=None*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

plotresults (*self*, *dynamic_sbm_series*)
Function to plot the results

predict_next_adj (*self*, *t*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

9.12 optimalSVD

class dynamicgem.embedding.optimalSVD.**optimalSVD** (**hyper_dict*, ***kwargs*)
Optimal Singular Value Decomposition

It performs the SVD of each new graph.

Parameters **Args** – K (int): dimension of the embedding theta (float): threshold for rerun datafile (str): location of the data file length (int) : total timesteps of the data nodemigraiton (int): number of nodes to migrate for sbm_cd datatype resultdir (str): directory to save the result datatype (str): sbm_cd, enron, academia, hep, AS

Examples

```
>>> from dynamicgem.embedding.optimalSVD import optimalSVD
>>> from dynamicgem.graph_generation import dynamic_SBM_graph
>>> node_num = 100
>>> community_num = 2
>>> node_change_num = 2
>>> length =5
>>> resultdir='./results_link_all'
>>> dynamic_sbm_series = dynamic_SBM_graph.get_community_diminish_series_v2(node_
↪num,
                                                                            community_
↪num,
                                                                            length,
                                                                            1,
                                                                            node_
↪change_num)
>>> graphs = [g[0] for g in dynamic_sbm_series]
```

```
>>> datafile = dataprep_util.prep_input_TIMERS(graphs, length, args.testDataType)
```

```
>>> embedding = optimalSVD(K=16,
                           Theta=0.5,
                           datafile=datafile,
                           length=length,
                           nodemigration=node_change_num,
                           resultdir=resultdir,
                           datatype='sbm_cd'
```

(continues on next page)

(continued from previous page)

```
                    )
>>> outdir_tmp = './output'
>>> if not os.path.exists(outdir_tmp):
>>>     os.mkdir(outdir_tmp)
>>> outdir_tmp = outdir_tmp + '/sbm_cd'
>>> if not os.path.exists(outdir_tmp):
>>>    os.mkdir(outdir_tmp)
>>> if not os.path.exists(outdir_tmp + '/optimalSVD'):
>>>     os.mkdir(outdir_tmp + '/optimalSVD')
>>>  embedding.learn_embedding()
>>>  outdir = resultdir
>>>  if not os.path.exists(outdir):
>>>     os.mkdir(outdir)
>>>  outdir = outdir + '/' + args.testDataType
>>>  if not os.path.exists(outdir):
>>>      os.mkdir(outdir)
>>>  embedding.get_embedding(outdir_tmp, 'optimalSVD')
    # embedding.plotresults()
>>>  outdir1 = outdir + '/optimalSVD'
>>>  if not os.path.exists(outdir1):
>>>      os.mkdir(outdir1)
>>>   lp.expstaticLP_TIMERS(dynamic_sbm_series,
                           graphs,
                           embedding,
                           1,
                           outdir1 + '/',
                           'nm' + str(args.nodemigration) + '_l' + str(length) + '_
↪emb' + str(int(dim_emb)),
                           )
```

get_edge_weight(*self*, *t*, *i*, *j*)
Function to get edge weight.

i
source node for the edge.
Type int

j
target node for the edge.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Weight of the given edge.

Return type Float

get_embedding(*self*, *outdir_tmp*, *method*)
Function to return the embeddings

get_method_name(*self*)
Function to return the method name.

Returns Name of the method.

Return type String

get_method_summary(*self*)
Function to return the summary of the algorithm.

Returns Method summary

Return type String

get_reconstructed_adj(*self*, *t*, *X=None*, *node_l=None*)
Function to reconstruct the adjacency list for the given node.

node_l
node for which the adjacency list will be created.
Type int

embed
Embedding values of all the nodes.
Type Matrix

filesuffix
File suffix to be used to load the embedding.
Type str

Returns Adjacency list of the given node.

Return type List

learn_embedding(*self*, *graph=None*)
Learns the embedding of the nodes.

graph
Networkx Graph Object
type Object

Returns: List: Node embeddings and time taken by the algorithm

plotresults(*self*, *dynamic_sbm_series*)
Function to plot the results

predict_next_adj(*self*, *t*, *node_l=None*)
Function to predict the next adjacency for the given node.

node_l
node for which the adjacency list will be created.
Type int

Returns Reconstructed adjancey list.

Return type List

CHAPTER

TEN

EVALUATION FUNCTIONS

10.1 Graph Reconstruction

dynamicgem.evaluation.evaluate_graph_reconstruction.**evaluateStaticGraphReconstruction**(*digrap*
*graph_*
X_stat
node_l
sam-
ple_ra
file_su
is_und
is_wei

Function to evaluate static graph reconstruction

dynamicgem.evaluation.evaluate_graph_reconstruction.**digraph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_graph_reconstruction.**graph_embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_graph_reconstruction.**X_stat**
Embedding values of the graph.

Type ndarray

dynamicgem.evaluation.evaluate_graph_reconstruction.**node_l**
Total number of nodes.

Type int

dynamicgem.evaluation.evaluate_graph_reconstruction.**sammple_ratio_e**
SAmpling ration for testing. Only sample number of nodes are tested.

Type float

dynamicgem.evaluation.evaluate_graph_reconstruction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_graph_reconstruction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_graph_reconstruction.**is_weighted**

Flag denoting if the graph has weighted edge.

type bool

Returns: ndarray: MAP, precision curve, error values and error baselines

dynamicgem.evaluation.evaluate_graph_reconstruction.**expGR**(*digraph*, *graph_embedding*, *X*, *n_sampled_nodes*, *rounds*, *res_pre*, *m_summ*, *file_suffix=None*, *is_undirected=True*, *sampling_scheme='rw'*)

Function to evaluate graph reconstruction

dynamicgem.evaluation.evaluate_graph_reconstruction.**digraph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_graph_reconstruction.**graph_embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_graph_reconstruction.**X_stat**
Embedding values of the graph.

Type ndarray

dynamicgem.evaluation.evaluate_graph_reconstruction.**n_sampled_nodes**
Total number of nodes.

Type int

dynamicgem.evaluation.evaluate_graph_reconstruction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_graph_reconstruction.**res_pre**
prefix to be used to store the result.

Type str

dynamicgem.evaluation.evaluate_graph_reconstruction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_graph_reconstruction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_graph_reconstruction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_graph_reconstruction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

10.2 Link Prediction

dynamicgem.evaluation.evaluate_link_prediction.**evaluateDynamicLinkPrediction**(*graph*, *embedding*, *rounds*, *n_sample_nodes=N*, *no_python=False*, *is_undirected=True*, *sampling_scheme='u_r*

Function to evaluate Dynamic Link Prediction

dynamicgem.evaluation.evaluate_link_prediction.**graph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**n_sample_nodes**
sampled nodes

Type list

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

Sampling scheme to be used.

type str

Returns: ndarray: MAP, precision curve

dynamicgem.evaluation.evaluate_link_prediction.**evaluateDynamicLinkPrediction_TIMERS**(*graph*, *embedding*, *t*, *rounds*, *n_sample*, *no_pytho*, *is_undire*, *sampling_sch*

Function to evaluate Dynamic Link Prediction for TIMERS

dynamicgem.evaluation.evaluate_link_prediction.**graph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**t**
sequence of the graph

Type int

dynamicgem.evaluation.evaluate_link_prediction.**n_sample_nodes**
sampled nodes

Type list

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

Sampling scheme to be used.

type str

Returns: ndarray: MAP, precision curve

dynamicgem.evaluation.evaluate_link_prediction.**evaluateDynamic_changed_LinkPrediction**(*graph*, *embedding*, *rounds*, *edges_*, *edges_*, *n_sam*, *no_py*, *is_und*, *sampling_sampling_*

Function to evaluate dynamic changed link prediction

dynamicgem.evaluation.evaluate_link_prediction.**graph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding.

Type object

dynamicgem.evaluation.evaluate_link_prediction.**edges_add**
list of edges to be added.

Type list

dynamicgem.evaluation.evaluate_link_prediction.**edges_rm**
list of edges to be removed.

Type list

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**evaluateDynamic_changed_LinkPrediction_v2**(*gr*
en
be
di
ro
ea
ea
*n_*
no
*is_*
sa
pl

Function to evaluate dynamic changed link prediction

dynamicgem.evaluation.evaluate_link_prediction.**graph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding.

Type object

dynamicgem.evaluation.evaluate_link_prediction.**edges_add**
list of edges to be added.

Type list

dynamicgem.evaluation.evaluate_link_prediction.**edges_rm**
list of edges to be removed.

Type list

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**expLP**(*graphs*, *embedding*, *rounds*, *res_pre*, *m_summ*, *n_sample_nodes=1000*, *train_ratio_init=0.5*, *no_python=False*, *is_undirected=True*, *sampling_scheme='u_rand'*)

Function to evaluate link prediction

dynamicgem.evaluation.evaluate_link_prediction.**digraph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_link_prediction.**graph_embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**X_stat**
Embedding values of the graph.

Type ndarray

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**res_pre**
prefix to be used to store the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**exp_changedLP**(*graphs*, *embedding*, *rounds*, *res_pre*, *m_summ*, *n_sample_nodes=1000*, *train_ratio_init=0.5*, *no_python=False*, *is_undirected=True*, *sampling_scheme='u_rand'*)

Function to evaluate only changed link prediction

dynamicgem.evaluation.evaluate_link_prediction.**digraph**
Networkx Graph Object

Type Object

dynamicgem.evaluation.evaluate_link_prediction.**graph_embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**X_stat**
Embedding values of the graph.

Type ndarray

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**res_pre**
prefix to be used to store the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**expstaticLP**(*dynamic_sbm_series*, *graphs*, *embedding*, *rounds*, *res_pre*, *m_summ*, *n_sample_nodes=1000*, *train_ratio_init=0.5*, *no_python=False*, *is_undirected=True*, *sampling_scheme='u_rand'*)

Function to evaluate statically changed link prediction

dynamicgem.evaluation.evaluate_link_prediction.**dynamic_sbm_series**
list of Networkx Graph Object

Type list

dynamicgem.evaluation.evaluate_link_prediction.**gaphs**
Networkx graphs

Type object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**res_pre**
prefix to be used to store the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**expstaticLP_TIMERS**(*dynamic_sbm_series*, *graphs*, *embedding*, *rounds*, *res_pre*, *m_summ*, *n_sample_nodes=1000*, *train_ratio_init=0.5*, *no_python=False*, *is_undirected=True*, *sampling_scheme='u_rand'*)

Function to evaluate statically changed link prediction for TIMERS

dynamicgem.evaluation.evaluate_link_prediction.**dynamic_sbm_series**
list of Networkx Graph Object

Type list

dynamicgem.evaluation.evaluate_link_prediction.**gaphs**
Networkx graphs

Type object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**res_pre**
prefix to be used to store the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**expstaticLP_TRIAD**(*dynamic_sbm_series*, *graphs*, *embedding*, *rounds*, *res_pre*, *m_summ*, *n_sample_nodes=1000*, *train_ratio_init=0.5*, *no_python=False*, *is_undirected=True*, *sampling_scheme='u_rand'*)

Function to evaluate statically changed link prediction for dynamic Triad

dynamicgem.evaluation.evaluate_link_prediction.**dynamic_sbm_series**
list of Networkx Graph Object

Type list

dynamicgem.evaluation.evaluate_link_prediction.**gaphs**
Networkx graphs

Type object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**res_pre**
prefix to be used to store the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**expstatic_changedLP**(*dynamic_sbm_series*, *graphs*, *embedding*, *rounds*, *res_pre*, *m_summ*, *n_sample_nodes=1000*, *train_ratio_init=0.5*, *no_python=False*, *is_undirected=True*, *sampling_scheme='u_rand'*)

Function to evaluate statically changed link prediction

dynamicgem.evaluation.evaluate_link_prediction.**dynamic_sbm_series**
list of Networkx Graph Object

Type list

dynamicgem.evaluation.evaluate_link_prediction.**gaphs**
Networkx graphs

Type object

dynamicgem.evaluation.evaluate_link_prediction.**embedding**
Algorithm for learning graph embedding

Type object

dynamicgem.evaluation.evaluate_link_prediction.**n_sampled_nodes**
List of sampled nodes.

Type int

dynamicgem.evaluation.evaluate_link_prediction.**train_ratio_init**
sample to be used for training and testing.

Type float

dynamicgem.evaluation.evaluate_link_prediction.**rounds**
Number of times to run the experiment

Type int

dynamicgem.evaluation.evaluate_link_prediction.**res_pre**
prefix to be used to store the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**m_summ**
summary to be used to save the result.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**file_suffix**
Suffix for file name.

Type str

dynamicgem.evaluation.evaluate_link_prediction.**is_undirected**
Flag to denote if the graph is directed.

Type bool

dynamicgem.evaluation.evaluate_link_prediction.**sampling_scheme**

sampling scheme for selecting the nodes.

type str

Returns: ndarray: Mean Average precision

dynamicgem.evaluation.evaluate_link_prediction.**getchangedlinks**(*G*, *Gnew*)
Functionto get all the changed links

10.3 Metrics

dynamicgem.evaluation.metrics.**checkedges**(*edge_list*, *e*)
Function to check if the given edgelist matches.

dynamicgem.evaluation.metrics.**edge_list**
List of predicted edges.

Type list

dynamicgem.evaluation.metrics.**e**

Original edge list

type list

Returns: bool: Boolean result to denoe if all the edges matches.

dynamicgem.evaluation.metrics.**computeMAP**(*predicted_edge_list*, *true_digraph*, *max_k=-1*)
Function to calculate Mean Average Precision

dynamicgem.evaluation.metrics.**predicted_edge_list**
List of predicted edges.

Type list

dynamicgem.evaluation.metrics.**true_digraph**
original graph

Type object

dynamicgem.evaluation.metrics.**max_k**

precision@k

type int

Returns: Float: Mean Average Precision score

dynamicgem.evaluation.metrics.**computeMAP_changed**(*predicted_edge_list*, *true_digraph*, *node_dict*, *edges_rm*, *max_k=-1*)

Function to calculate MAP of the change graph

dynamicgem.evaluation.metrics.**predicted_edge_list**
List of predicted edges.

Type list

dynamicgem.evaluation.metrics.**true_digraph**
original graph

Type object

dynamicgem.evaluation.metrics.**node_dict**
Dictionary for the nodes.

Type dict

dynamicgem.evaluation.metrics.**node_edges_rm**
list of edges removed from the original graph.

Type list

dynamicgem.evaluation.metrics.**max_k**

precision@k

type int

Returns: Float: Mean Average Precision score

dynamicgem.evaluation.metrics.**computePrecisionCurve**(*predicted_edge_list*, *true_digraph*, *max_k=-1*)

Function to calculate the precision curve

dynamicgem.evaluation.metrics.**predicted_edge_list**
List of predicted edges.

Type list

dynamicgem.evaluation.metrics.**true_digraph**
original graph

Type object

dynamicgem.evaluation.metrics.**max_k**

precision@k

type int

Returns: ndarray: precision_scores, delta_factors

dynamicgem.evaluation.metrics.**computePrecisionCurve_changed**(*predicted_edge_list*, *true_digraph*, *node_edges_rm*, *max_k=-1*)

Function to calculate Preicison curve of changed graph

dynamicgem.evaluation.metrics.**predicted_edge_list**
List of predicted edges.

Type list

dynamicgem.evaluation.metrics.**true_digraph**
original graph

Type object

dynamicgem.evaluation.metrics.**node_edges_rm**
list of edges removed from the original graph.

Type list

dynamicgem.evaluation.metrics.**max_k**

precision@k

type int

Returns: Float: Mean Average Precision score

dynamicgem.evaluation.metrics.**getEmbeddingShift**(*X1*, *X2*, *S1*, *S2*)
Function to get the shift in embedding

dynamicgem.evaluation.metrics.**getMetricsHeader**()
Function to get the header for storing the result

dynamicgem.evaluation.metrics.**getNodeAnomaly**(*X_dyn*)
Function to get the node anomaly

dynamicgem.evaluation.metrics.**getPrecisionReport**(*prec_curv*, *edge_num*)
Function to get the report summary for precision

dynamicgem.evaluation.metrics.**getStabilityDev**(*X1*, *X2*, *S1*, *S2*)
Function to get the deviation froms stability

10.4 Embedding Visualization

dynamicgem.evaluation.visualize_embedding.**expVis**(*X*, *res_pre*, *m_summ*, *node_labels=None*, *di_graph=None*)
Function to perform visualixe the experiments of dynamic graph

dynamicgem.evaluation.visualize_embedding.**plot_embedding2D**(*node_pos*, *node_colors=None*, *di_graph=None*)
Function to plot the embedding in two dimension using TSNE to reduce the dimension

dynamicgem.evaluation.visualize_embedding.**plot_single_step**(*node_pos*, *graph_info*, *dyn_changed_node*)
Function to plot a single step

dynamicgem.evaluation.visualize_embedding.**plot_static_sbm_embedding**(*nodes_pos_list*, *dynamic_sbm_series*)
Function to plot the static sbm embedding

CHAPTER

ELEVEN

EXPERIMENTS

11.1 Config

11.2 Experiments

dynamicgem.experiments.exp.**call_exps**(*params*, *data_set*, *n_graphs*)
Function to run the experiments

dynamicgem.experiments.exp.**n_graphs**
Total number of graphs in a sequence.

Type int

dynamicgem.experiments.exp.**data_set**
Name of the dataset to be used for the experiment

Type str

dynamicgem.experiments.exp.**params**
Dictionary of parameters necessary for running the experiment

Type dict

dynamicgem.experiments.exp.**call_plot_hyp**(*data_set*, *params*)
Function to plot the result of hyperparameter search

dynamicgem.experiments.exp.**data_set**
Name of the dataset to be used for the experiment

Type str

dynamicgem.experiments.exp.**params**
Dictionary of parameters necessary for running the experiment

Type dict

dynamicgem.experiments.exp.**call_plot_hyp_all**(*data_sets*, *params*)
Function to plot the the result of all the hyper-parameters

dynamicgem.experiments.exp.**data_set**
Name of the dataset to be used for the experiment

Type str

dynamicgem.experiments.exp.**params**
Dictionary of parameters necessary for running the experiment

Type dict

dynamicgem.experiments.exp.**choose_best_hyp**(*data_set*, *graphs*, *params*)
Function to get the best hyperparameter using a grid search

dynamicgem.experiments.exp.**data_set**
Name of the dataset to be used for the experiment

Type str

dynamicgem.experiments.exp.**graphs**
Networkx Graph Object

Type Object

dynamicgem.experiments.exp.**params**
Dictionary of parameters necessary for running the experiment

Type dict

dynamicgem.experiments.exp.**get_max**(*val*, *val_max*, *idx*, *idx_max*)
Function to get the maximum value.

dynamicgem.experiments.exp.**learn_emb**(*MethObj*, *graphs*, *params*, *res_pre*, *m_summ*)
Function to learn embedding

dynamicgem.experiments.exp.**MethObj**
Object of the algorithm class

Type obj

dynamicgem.experiments.exp.**graphs**
Networkx Graph Object

Type Object

dynamicgem.experiments.exp.**params**
Dictionary of parameters necessary for running the experiment

Type dict

dynamicgem.experiments.exp.**res_pre**
Prefix of the filename for saving the result.

Type str

dynamicgem.experiments.exp.**m_summ**

summary added to the filename of the result.

type str

Returns: ndarray: Learned embedding

dynamicgem.experiments.exp.**run_exps**(*MethObj*, *meth*, *dim*, *graphs*, *data_set*, *params*)
Function to run the experiment

dynamicgem.experiments.exp.**MethObj**
Object of the algorithm class

Type obj

dynamicgem.experiments.exp.**meth**
Name of the method

Type str

dynamicgem.experiments.exp.**dim**
Dimension of the embedding

Type int

dynamicgem.experiments.exp.**graphs**
Networkx Graph Object

Type Object

dynamicgem.experiments.exp.**data_set**
Name of the dataset to be used for the experiment

Type str

dynamicgem.experiments.exp.**params**

Dictionary of parameters necessary for running the experiment

type dict

Returns: ndarray: Learned embedding

CHAPTER

TWELVE

GRAPH GENERATION

12.1 Dynamic Stochastic Block Model Graph

dynamicgem.graph_generation.dynamic_SBM_graph.**diminish_community**(*sbm_graph*, *community_id*, *nodes_to_purturb*, *criteria*, *criteria_r*)

Function to diminsh the SBM community

dynamicgem.graph_generation.dynamic_SBM_graph.**sbm_graph**
Networkx Graph Object

Type Object

dynamicgem.graph_generation.dynamic_SBM_graph.**community_id**
Community to diminish

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**criteria**
Criteria used to diminish the community

Type str

dynamicgem.graph_generation.dynamic_SBM_graph.**criteria_r**
Used to sort the nodes in reverse once order based on criteria

Type bool

dynamicgem.graph_generation.dynamic_SBM_graph.**nodes_to_purturb**
Number of nodes to perturb

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**diminish_community_v2**(*sbm_graph*, *community_id*, *nodes_to_purturb*, *chngnodes*)

Function to diminsh the SBM community

dynamicgem.graph_generation.dynamic_SBM_graph.**sbm_graph**
Networkx Graph Object

Type Object

dynamicgem.graph_generation.dynamic_SBM_graph.**community_id**
Community to diminish

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**nodes_to_purturb**
Number of nodes to perturb

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**chngnodes**
List of nodes that is perturbed

Type list

dynamicgem.graph_generation.dynamic_SBM_graph.**drawGraph**(*node_num*, *community_num*)
Function to draw the graphs

dynamicgem.graph_generation.dynamic_SBM_graph.**dyn_node_chng**(*sbm_graph*, *node_id*)
Function to dynamically change the nodes

dynamicgem.graph_generation.dynamic_SBM_graph.**sbm_graph**
Networkx Graph Object

Type Object

dynamicgem.graph_generation.dynamic_SBM_graph.**node_id**
Id of the node to resample

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**dyn_node_chng_v2**(*sbm_graph*, *node_id*)
Function to dynamically change the nodes

dynamicgem.graph_generation.dynamic_SBM_graph.**sbm_graph**
Networkx Graph Object

Type Object

dynamicgem.graph_generation.dynamic_SBM_graph.**node_id**
Id of the node to resample

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**get_community_diminish_series**(*node_num*, *community_num*, *length*, *community_id*, *nodes_to_purturb*, *criteria*, *criteria_r*)
Function to get diminshing community series

dynamicgem.graph_generation.dynamic_SBM_graph.**node_num**
Total number of nodes

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**community_num**
Total number of community

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**nodes_to_purturb**
Number of nodes to perturb

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**length**
Length of the graph sequence

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**community_id**
Community to diminish

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**criteria**
Criteria used to diminish the community

Type str

dynamicgem.graph_generation.dynamic_SBM_graph.**criteria_r**
Used to sort the nodes in reverse once order based on criteria

Type bool

dynamicgem.graph_generation.dynamic_SBM_graph.**get_community_diminish_series_v2**(*node_num*, *community_num*, *length*, *community_id*, *nodes_to_purturb*

Function to get diminishing community series

dynamicgem.graph_generation.dynamic_SBM_graph.**node_num**
Total number of nodes

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**community_num**
Total number of community

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**nodes_to_purturb**
Number of nodes to perturb

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**length**
Length of the graph sequence

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**community_id**
Community to diminish

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**get_random_perturbation_series**(*node_num*, *community_num*, *length*, *nodes_to_purturb*)

Function to get random perturbation

dynamicgem.graph_generation.dynamic_SBM_graph.**node_num**
Total number of nodes

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**community_num**
Total number of community

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**nodes_to_purturb**
Number of nodes to perturb

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**length**
Length of the graph sequence

Type int

dynamicgem.graph_generation.dynamic_SBM_graph.**random_node_perturbation**(*sbm_graph*, *nodes_to_purturb*)

Function to randomly perturb the nodes

dynamicgem.graph_generation.dynamic_SBM_graph.**sbm_graph**
Networkx Graph Object

Type Object

dynamicgem.graph_generation.dynamic_SBM_graph.**nodes_to_purturb**
Number of nodes to perturb

Type int

CHAPTER

THIRTEEN

CONTRIBUTE

Contributing to dynamicgem

We feel humbled that you have decided to contribute to the dynamicgem repository. Thank you! Please read the following guidelines to checkout how you can contribute.

You can contribute to this code through Pull Request on GitHub. Please, make sure that your code is coming with unit tests to ensure full coverage and continuous integration in the API.

- **Reporting Bugs**: Please use the issue Template to report bugs.
- **Suggesting Enhancements**: If you have any suggestion for enhancing any of the modules please send us an enhancement using the issue Template as well.
- **Adding Algorithm**: We are continually striving to add the state-of-the-art algorithms in the library. If you want to suggest adding any algorithm or add your algoirithm to the library.
- **Adding Evaluation Metric**: We are always eager to add more evaluation metrics for link prediction, triple classification, and so on. You may create a new evaluation process in dynamicgem/evaluation.py to add the metric.

CHAPTER

FOURTEEN

AUTHORS

14.1 Core Development

Sujit Rokka Chhetri (Ph.D)
University of California, Irvine
Email: schhetri@uci.edu

Palash Goyal (Ph.D)
University of Southern California
palashgo@usc.edu

Arquimedes Martinez Canedo (Ph.D)
Principal Scientist
Siemens Corporate Technology
arquimedes.canedo@siemens.com

14.2 Contributors

Ninareh Mehrabi
University of Southern California
ninarehm@usc.edu

Emilio Ferrara
Associate Professor
University of Southern California
emiliofe@usc.edu

CHAPTER

FIFTEEN

CITING DYNAMICGEM

If you found this open source library useful, please kindly cite us:

```
@article{goyal2018dynamicgem,
title={DynamicGEM: A Library for Dynamic Graph Embedding Methods},
author={Goyal, Palash and Chhetri, Sujit Rokka and Mehrabi, Ninareh and Ferrara,␣
↪Emilio and Canedo, Arquimedes},
journal={arXiv preprint arXiv:1811.10734},
year={2018}
}
```

CHAPTER

SIXTEEN

LICENSE

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX

E

I

J

L

M

N

O

P

R

S

T

X