gemben

Release 0.0.3

Palash Goyal, Di Huang, Ankita Goswami, Sujit Rokka Chhetri, Ar

Aug 14, 2019

QUICK START

GEM-Benchmark is a principled framework to compare graph embedding methods. We test embedding methods on a corpus of real-world networks with varying properties and provide insights into existing state-of-the-art embedding approaches. We cluster the input networks in terms of their properties to get a better understanding of embedding performance. Furthermore, we compare embedding methods with traditional link prediction techniques to evaluate the utility of embedding approaches. We use the comparisons on benchmark graphs to define a score, called GFS-score, that can apply to measure any embedding method. We rank the state-of-the-art embedding approaches using the GFS-score and show that it can be used to understand and evaluate a novel embedding approach. We envision that the proposed framework may serve as a community benchmark to test and compare the performance of future graph embedding techniques.

See the source code repository on github: https://github.com/palash1992/GEM-benchmark

CHAPTER

ONE

DEPENDENCIES

You will need following modules to be installed for using the gemben library:

- matplotlib>=3.1.0
- cmake>=3.14.4
- pandas>=0.24.2
- seaborn>=0.9.0
- tables>=3.5.2
- networkx>=2.3
- Keras>=2.2.4
- six>=1.11.0
- Theano>=1.0.4
- numpy>=1.16.4
- scipy>=1.3.0
- sklearn>=0.0
- bayesian-optimization>=1.0.1
- networkit>=5.0
- tensorflow==1.14.0

The following are documentaiton generation dependecies:

- numpydoc>=0.9.1
- Sphinx>=2.1.2
- sphinx-rtd-theme>=0.4.3
- sphinx-gallery>=0.4.0

CHAPTER

TWO

INSTALLATION

gemben is available in the PyPi's repository and is pip installable. Please follow the following steps to install the library.

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
```

Install gemben using 'pip':

```
(venv) $ pip install gemben
```

Install stable version directly from github repo:

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

Install development version directly from github repo:

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

CHAPTER

THREE

TESTING

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

```
pytest
```

CHAPTER

FOUR

INTRODUCTION

Graphs are a natural way to represent relationships and interactions between entities in real systems. For example, people on social networks, proteins in biological networks, and authors in publication networks can be represented by nodes in a graph, and their relationships such as friendships, protein-protein interactions, and co-authorship are represented by edges in a graph. These graphical models enable us to understand the behavior of systems and to gain insight into their structure. These insights can further be used to predict future interactions and missing information in the system. These tasks are formally defined as link prediction and node classification. Link prediction estimates the likelihood of a relationship among two entities. This is used, for example, to recommend friends on social networks and to sort probable protein-protein interactions on biological networks. Similarly, node classification estimates the likelihood of a node's label. This is used, for example, to infer missing meta-data on social media profiles, and genes in proteins.

Numerous graph analysis methods have been developed. Broadly, these methods can be categorized as non-parametric and parametric. Non-parametric methods operate directly on the graph whereas parametric methods represent the properties of nodes and edges in the graph in a low-dimensional space. Non-parametric methods such as Common Neighbors, Adamic Adar and Jaccard coefficient require access and knowledge of the entire graph for the prediction. On the other hand, parametric models such as Thor et. al. employ graph summarization and define super nodes and super edges to perform link prediction. Kim et. al. use Expectation Maximization to fit the real network as a Kronecker graph and estimate the parameters. Another class of parametric models that have gained much attention recently are graph embeddings. Graph embedding methods define a low-dimensional vector for each node and a distance metric on the vectors. These methods learn the representation by preserving certain properties of the graph. Graph Factorization preserves visible links, HOPE aims to preserve higher order proximity, and node2vec preserves both structural equivalence and higher order proximity. In this benchmark library, we focus our attention on graph embedding methods. While this is a very active area of research that continues to gain popularity among researchers, there are several challenges that must be addressed before graph embedding algorithms become mainstream.

CHAPTER

FIVE

CHALLENGES

Most research on graph embedding has focused on the development of mechanisms to preserve various characteristics of the graph in the low-dimensional space. However, very little attention has been dedicated to the development of mechanisms to rigorously compare and evaluate different graph embedding methods. To make matters worse, most of the existing work use simple synthetic data sets for visualization and a few real networks for quantitative comparison. Goyal et. al. use Stochastic Block Models to visualize the results of graph embedding methods. Salehi at. al. use the Barabasi-Albert graph to understand the properties of embeddings. Such evaluation strategy suffers from the following challenges:

- Properties of real networks vary according to the domain. Therefore it is often difficult to ascertain the reason behind the performance improvement of a given method on a particular real dataset (as shown in Goyal et. al.
- As demonstrated in this benchmark library, the performance of embedding approaches vary greatly, and according to the properties of different graphs. Therefore, the utility of any specific method is difficult to establish and to characterize. In practice, the performance improvement of a method can be attributed to stochasticity.
- Different methods use different metrics for evaluation. This makes it very difficult to compare the performance of different graph embedding methods on a given problem.
- Typically, each graph embedding method has a reference implementation. This implementation makes specific assumptions about the data, representation, etc. This further complicates the comparison between methods.

CHAPTER

SIX

WHY THIS BENCHMARK PACKAGE?

In this benchmark libarry, we aim to:

- provide a unifying framework for comparing the performance of state-of-the-art and future graph embedding methods;
- establish a benchmark comprised of 100 real-world graphs that exhibit different structural properties; and
- provide users with a fully automated library that selects the best graph embedding method for their graph data.

We address the above challenges with the following contributions:

- We propose an evaluation benchmark to compare and evaluate embedding methods. This benchmark consists of 100 real-world graphs categorized into four domains: social, biology, technological and economic.
- Using our evaluation benchmark, we evaluate and compare 8 state-of-the-art methods and provide, for the first time, a characterization of their performance against graphs with different properties. We also compare their scores with traditional link prediction methods and ascertain the general utility of embedding methods.
- A new score, GFS-score, is introduced to compare various graph embedding methods for link prediction. The GFS-score provides a robust metric to evaluate a graph embedding approach by averaging over 100 graphs. It further has many components based on the type and property of graph yielding insights into the methods.
- A Python library comprised of 4 state-of-the-art embedding methods, and 4 traditional link prediction methods. This library automates the evaluation, comparison against all the other methods, and performance plotting of any new graph embedding method.

CHAPTER

SEVEN

ORGANIZATION

The various components of the gemben library are as follows:

1) `embedding` - It consists of some of the baseline state-of-the-art graph embedding algorithms againsts which the benchmark can be evaluated.
2) `evaluation` - This module consists of the evaluation functions for graph reconstruction, link prediction, and node classification along with definition of the metrics used for evaluation. It also consists of function to visualize the embeddings.
3) `experiments` - This is a self-contained module which can be used to test various state-of-the-art algorithms agains the benchmarks.
4) `plots` -This modules handles the plotting of the compartive quantitative results.
5) `utils` - The mosudle consists of various utility function succ as bayesian hyper-parameter optimization, evaluation, graph generation, etc.

CHAPTER

EIGHT

BACKGROUND

This section introduces the notation used in this benchmark library, and provides a brief overview of graph embedding methods. In-depth analysis of graph embedding theory we refer the reader to here.

$G(V, E)$ denotes a weighted graph where V is the set of vertices and E is the set of edges. We represent W as the adjacency matrix of G, where $W_{ij} = 1$ represents the presence of an edge between i and j. A graph embedding is a mapping $f : V- > \mathbb{R}^d$, where $d << |V|$ and the function f preserves some proximity measure defined on graph G. It aims to map similar nodes close to each other. Function f when applied on the graph G yields an embedding Y.

In this benchmark library, we evaluate four state-of-the-art graph embedding methods on a set of real graphs denoted by $\mathcal{R}$ and synthetic graphs denoted by $\mathcal{S}$. To analyze the performance of methods, we categorize the graphs into a set of domains $\mathcal{D} = \{$ Social, Economic, Biology, Technological:math:*rbrace*. The set of graphs in a domain $D \in \mathcal{D}$ is represented as $\mathcal{R}_D$. We use multiple evaluation metrics on graph embedding methods to draw insights into each approach. We denote this set of metrics as $\mathcal{M}$. The notations are summarized as follows:.

- G : Graphical representation of the data
- E : Set of edges in the graph
- W : Adjacency matrix of the graph, $|V| \times |V|$
- f : Embedding function
- $\mathcal{S}$: Set of synthetic graphs
- $\mathcal{R}_D$: Set of real graphs in domain D
- $\mathcal{D}$: Set of domains
- $\mathcal{M}$: Set of evaluation metrics
- e_m : Evaluation function for metric m
- $\mathcal{A}$: Set of graph and embedding attributes
- d : Number of embedding dimensions
- Y : Embedding of the graph, $|V| \times d$

8.1 Graph Embedding Methods

Graph embedding methods embed graph vertices into a low-dimensional space. The goal of graph embedding is to preserve certain properties of the original graph such as distance between nodes and neighborhood structure. Based upon the function f used for embedding the graph, existing methods can be classified into three categories: **factorization based**, **random walk based** and **deep learning based**.

8.2 Factorization based approaches

Factorization based approaches apply factorization on graph related matrices to obtain the node representation. Graph matrices such as the adjacency matrix, Laplacian matrix, and Katz similarity matrix contain information about node connectivity and the graph's structure. Other matrix factorization approaches use the eigenvectors from spectral decomposition of a graph matrix as node embeddings. For example, to preserve locality, LLE uses d eigenvectors corresponding to eigenvalues from second smallest to $(d+1)^{th}$ smallest from the sparse matrix $(I-W)^\intercal(I-W)$. It assumes that the embedding of each node is a linear weighted combination of the neighbor's embeddings. Laplacian Eigenmaps take the first d eigenvectors with the smallest eigenvalues of the normalized Laplacian $D^{-1/2}LD^{-1/2}$. Both LLE and Laplacian Eigenmaps were designed to preserve the local geometric relationships of the data. Another type of matrix factorization methods learn node embeddings under different optimization functions in order to preserve certain properties. Structural Preserving Embedding builds upon Laplacian Eigenmaps to recover the original graph. Cauchy Graph Embedding uses a quadratic distance formula in the objective function to emphasize similar nodes instead of dissimilar nodes. Graph Factorization uses an approximation function to factorize the adjacency matrix in a more scalable manner. GraRep and HOPE were invented to keep the high order proximity in the graph. Factorization based approaches have been widely used in practical applications due to their scalability. The methods are also easy to implement and can yield quick insights into the data set.

8.3 Random walk approaches

Random walk based algorithms are more flexible than factorization methods to explore the local neighborhood of a node for high-order proximity preservation. DeepWalk and Node2vec aim to learn a low-dimensional feature representation for nodes through a stream of random walks. These random walks explore the nodes' variant neighborhoods. Thus, random walk based methods are much more scalable for large graphs and they generate informative embeddings. Although very similar in nature, DeepWalk simulates uniform random walks and Node2vec employs search-biased random walks, which enables embedding to capture the community or structural equivalence via different bias settings. LINE combines two phases for embedding feature learning: one phase uses a breadth-first search (BFS) traversal across first-order neighbors, and the second phase focuses on sampling nodes from second-order neighbors. HARP improves DeepWalk and Node2vec by creating a hierarchy for nodes and using the embedding of the coarsened graph as a better initialization in the original graph. Walklets extended Deepwalk by using multiple skip lengths in random walking. Random walk based approaches tend to be more computationally expensive than factorization based approaches but can capture complex properties and longer dependencies between nodes.

8.4 Neural network approaches

The third category of graph embedding approaches is based on neural networks. Deep neural networks based approaches capture highly non-linear network structure in graphs, which is neglected by factorization based and random walk based methods. One type of deep learning based methods such as SDNE uses a deep autoencoder to provide non-linear functions to preserve the first and second order proximities jointly. Similarly, DNGR applies random surfing on input graph before a stacked denoising autoencoder and makes the embedding robust to noise in graphs. Another genre of methods use Graph Neural Networks(*GNNs*) and Graph Convolutional Networks (*GCNs*) (bruna2013spectral, henaff2015deep, li2015gated, hamilton2017inductive) to aggregate the neighbors embeddings and features via convolutional operators, including spatial or spectral filters. GCNs learn embeddings in a semi-supervised manner and have shown great improvement and scalability on large graphs compared to other methods. SEAL learns a wide range of link prediction heuristics from extracted local enclosing subgraphs with GNN. DIFFPOOL employs a differentiable graph pooling module on GNNs to learn hierarchical embeddings of graphs. Variational Graph Auto Encoders (*VGAE*) utilizes a GCN as encoder and inner product as decoder, which provides embedding with higher quality than autoencoders. Deep neural network based algorithms like *SDNE* and *DNGR* can be computational costly since they require the global information such as adjacency matrix for each node as input. GCNs based methods are more scalable and flexible to characterize global and local neighbours through variant convolutional and pooling layers.

CHAPTER

NINE

GEM-BENCHMARK

Unlike other fields with well established benchmark datasets (e.g. community detection), the graph embedding community has adopted an ad-hoc approach to evaluate new methods. Typically, graph embedding methods are evaluated on only a few real networks, and these are biased towards specific properties. This ad-hoc evaluation approach restricts us from understanding how the algorithm would behave if we vary a certain property of the graph, or how the algorithm performs on other types of graphs. In order to propose a more rigorous evaluation approach, we must first to understand the key attributes that govern the performance of graph embedding methods. First, the *size of the graph* (**A1**) is a challenge for any method. Real graphs vary in the number of nodes, from a few hundred to millions of nodes. Different methods make different assumptions on how to capture the higher order proximities and structural dependencies between nodes, and this greatly affects their scalability. Second, the *density of the graph* (**A2**) plays an important role in defining its structure. Lower density results in lesser information about the nodes which may hamper the performance of some methods. Third, the *dimension of the embedding* (**A3**) determines how concisely the method can store the information about a given graph. Higher dimension of the embedding may lead to overfitting of the graph whereas lower dimension of the embedding may not be enough to capture the information the graph provides resulting in underfitting. Fourth, the *evaluation metric* (**A4**) used to evaluate the method captures different aspects of the prediction. Global metrics are often biased towards high degree nodes whereas local metrics can be biased towards lower degree nodes.

In this benchmark libary, we take the first step towards establishing a graph embedding benchmark. We propose a benchmark evaluation framework to answer the following questions:

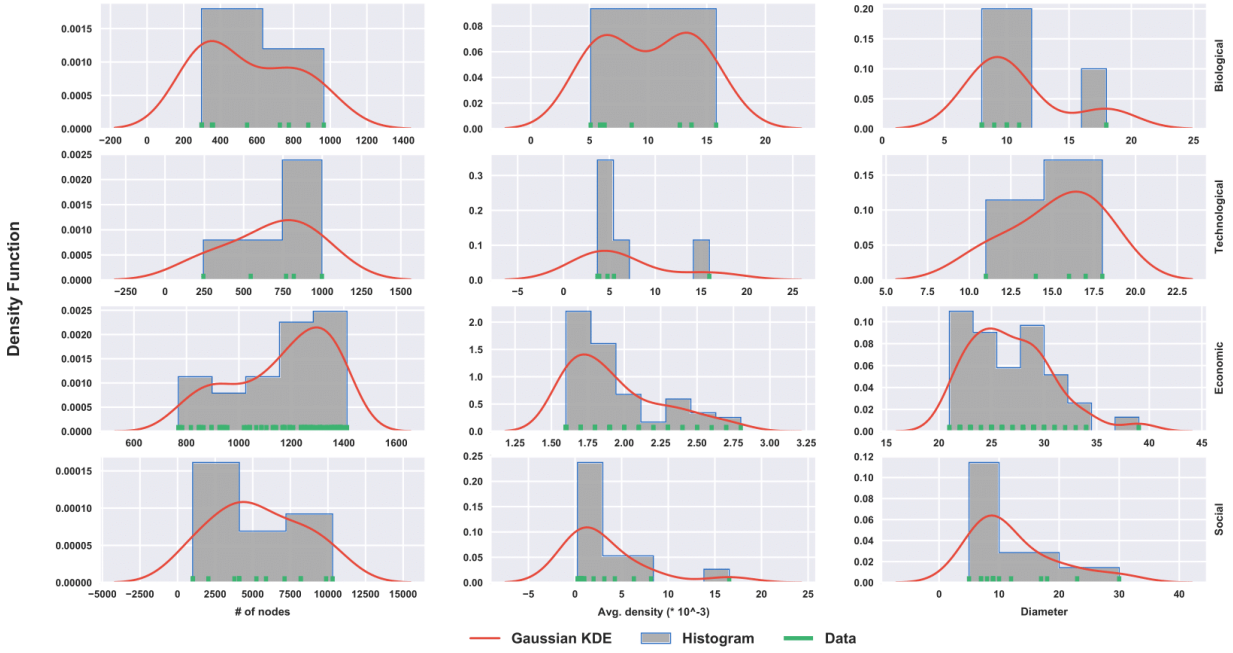
- **Q1**: How does the performance of embedding methods vary with the increasing size of the graph?
- **Q2**: How does increasing the density of graph affect the model?
- **Q3**: How does the optimal embedding dimension vary with an increasing number of nodes in the graph?
- **Q4**: How does the performance vary with respect to the evaluation metric?

To address the above questions, we introduce a suite of 100 real graphs and vary the above attributes (**A1**, . . . , **A4**) in the graphs and the embedding methods. Varying the *size of the graph* (**A1**) in terms of number of nodes answers the first question (**Q1**) and helps us understand which methods are best when used in small, medium, and large graphs. Similarly, varying the *density of the graph* (**A2**) in terms of the average degree of nodes helps us understand its effect in the embedding performance. This answers the second question (**Q2**). Furthermore, varying the *dimension of the embedding* (**A3**) helps us draw insights into the information compression power of the embedding approach. This answers the third question (**Q3**). Finally, by varying the *evaluation metrics* (**A4**) we can analyze the performance sensitivity of the method and can help us infer the bias of the embedding method towards specific nodes int he graph. This answers the fourth question (**Q4**).

9.1 Real Graphs

We propose a novel data set containing 100 real world graphs from four domains: social, biology, economic, and technological. To demonstrate the usefulness of this benchmark, we evaluate eight graph embedding methods and

measure their performance. This provides valuable insights about every method and their sensitivity to different graph properties. This paves the way towards a framework that can be used to recommend the best embedding approaches for a given graph with a unique set of properties.



The above figure summarizes the main properties of the graphs from different domains in the data set. We observe that economic graphs have a lower average density varying between **0.00160** and **0.00280** with a higher number of nodes concentrated in lower density spectrum. Technological and social graphs are denser with an average density between **0.0030** to **0.0160**. It is interesting to note that despite the wide average density range densities are concentrated primarily in the lower and higher values with a gap in between. Biological graphs have an almost uniform distribution of densities ranging from **0.005** to **0.0155**.

Next, we observe the domain-wise pattern of diameters. Economic graphs have the widest range(**20 - 40**) and the highest values of diameters which justifies the lowest average densities observed. Technological graphs with diameter ranges between **11** and **17.5** are less sparse when compared with economic graphs. Biological graphs have a good combination of both dense and sparse graphs with a majority of graphs lying in small diameter range. Biological graphs typically have short long diameter ranges as (**8** to **12**) and (**16** to **18**) respectively. Social graphs have in general a lower diameter around 10 although some of them have higher diameters.

On further investigation, we observe that biological networks have the highest clustering tendencies with an average clustering coefficient as **0.10**. However, economic graphs stand in absolute contrast to them with very low clustering coefficient of **0.00016** as the highest recorded average clustering coefficient. Technological networks are somewhere in between the aforementioned extremes with **0.03** as the highest recorded average clustering coefficients. Clustering tendencies can be sought to have a high correlation with average density and diameter observations.

> Note that these 100 graphs include a very diverse set of graphs in terms of the *size of the graph* (**A1**) ranging from **200** to **1500** nodes, and in terms of the *density of the graph* (**A2**) ranging from an average density between **0.0015** to **0.020**. This graph diversity is helpful in characterizing the performance of different embedding methods.

9.2 Evaluation Metrics

In the graph embedding literature, there are two primary metrics that are used to evaluate the performance of the methods on link prediction:

1) Precision at k ($P@k$) and

2) Mean Average Precision (MAP) These metrics are defined as follows:

P@k is the fraction of correct predictions in the top k predictions. It is defined as $P@k = \frac{|E_{pred}(1:k) \cap E_{obs}|}{k}$,

where $E_{pred}(1:k)$ are the top k predictions and E_{obs} are the observed edges/hidden edges.

MAP estimates the prediction precision for every node and computes the prediction average over all nodes, as follows:

$MAP = \frac{\sum_i AP(i)}{|V|}$

where $AP(i) = \frac{\sum_k P@k(i) \cdot \mathbb{I}\{E_{pred_i}(k) \in E_{obs_i}\}}{|\{k : E_{pred_i}(k) \in E_{obs_i}\}|}$,

$P@k(i) = \frac{|E_{pred_i}(1:k) \cap E_{obs_i}|}{k}$,

and E_{pred_i} and E_{obs_i} are the predicted and observed edges for node i respectively.

Intuitively, $P@k$ is a global metric that measures the accuracy of the most likely links predicted. On the other hand, *MAP* measures the accuracy of prediction for each node and computes their average. These metrics are often uncorrelated and reflect the properties captured by the prediction method at different levels (MAP on local level and $P@k$ on global level). In this benchmark library, we present results using both these metrics to analyze each approach.

9.3 GFS-score

We now define a set of scores to evaluate a graph embedding model on our data set. The scores are divided into components to draw insights into a method's approach across domains and metrics. We further plot the metrics varying various graph properties to understand the sensitivity of the models to these properties.

Given a set of graph domains $\mathcal{D}$, a set of evaluation metrics $\mathcal{M}$ and evaluation function $e_m(graph, approach)$ for $m \in \mathcal{M}$, we define GFS-score for an approach a as follows:

$micro - GFS - m(a) = \frac{\sum_{g \in \mathcal{G}} (e_m(g,a)/e_m(g,random))}{|\mathcal{G}|}$

$macro - GFS - m(a) = \frac{\sum_{d \in \mathcal{D}} GFS - m(d,a)}{|\mathcal{D}|}$

$GFS - m(d,a) = \frac{\sum_{g \in \mathcal{G}_d} (e_m(g,a)/e_m(g,random))}{|\mathcal{G}_d|}$

where $\mathcal{G}_d$ is the set of graphs in domain d.

The GFS-score is a robust score which averages over a set of real graphs with varying properties. It is normalized in order to ascertain the gain in performance with respect to a random prediction. The domain scores provide insights into the applicability of each approach to the different graph categories.

9.4 Link Prediction Baselines

Our link prediction baselines were selected to showcase the utility of embedding approaches on real graphs and establish the ground truth for comparison between the state-of-the-art methods. The link prediction baselines are:

- Preferential Attachment :is based on the assumption that the connection to a node is proportional to its degree. It defines the similarity between the nodes as the product of their degrees.

- Common Neighbors: defines the similarity between nodes as the number of common neighbors between them.
- Adamic Adar: is based on the intuition that common neighbors with very large neighbourhoods are less significant than common neighbors with small neighborhoods when predicting a connection between two nodes. Formally, it is defined as the sum of the inverse logarithmic degree centrality of the neighbours shared by the two nodes.
- Jaccards Coefficient: measures the probability that two nodes i and j have a connection to node k, for a randomly selected node k from the neighbors of i and j.

9.5 Embedding Approaches

We illustrate the benchmark data set on four popular graph embedding techniques to illustrate the utility of the benchmark and rank the state-of-the-art embedding approaches. The techniques preserve various properties including local neighborhood, higher order proximity and structure.

- Laplacian Eigenmaps : It penalizes the weighted square of distance between neighbors. This is equivalent to factorizing the normalized Laplacian matrix.
- Graph Factorization : It factorizes the adjacency matrix with regularization.
- Higher Order Proximity Preserving (**HOPE**): It factorizes the higher order similarity matrix between nodes using generalized singular value decomposition.
- Structural Deep Network Embedding (**SDNE**) : It uses deep autoencoder along with Laplacian Eigenmaps objective to preserve first and second order proximities.

CHAPTER

TEN

BENCHMARK DATASET

The list of graphs included in the benchmark library are as follows:

- barbell graph
- binary community graph
- barabasi albert graph
- random geometric graph
- waxman graph
- watts strogatz graph
- duplication divergence graph
- powerlaw cluster graph
- stochastic block model
- r mat graph
- hyperbolic graph
- stochastic kronecker graph
- lfr benchmark graph

We vary various attributes of these graphs to generate the benchmark.

CHAPTER

ELEVEN

BASELINE GRAPH EMBEDDING ALGORITHMS

11.1 Adamic Adar

class gemben.embedding.aa.**AdamicAdar**(**hyper_dict*, ***kwargs*)

Adamic Adar.

Adamic Adar is based on the intuition that common neighbors with very large neighbourhoods are less significant than common neighbors with small neighborhoods when predicting a connection between two nodes. Formally, it is defined as the sum of the inverse logarithmic degree centrality of the neighbours shared by the two nodes.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.aa import AdamicAdar
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = AdamicAdar(4, 0.01)
>>> embedding.learn_embedding(graph=G, edge_f=None, is_weighted=True, no_
↪python=True)
>>> print('Adamic Adar: Training time: %f' % (time() - t1))
```

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

Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)

Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)

Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.2 Static Graph Embedding

11.3 Common Neighbors

class gemben.embedding.cn.**CommonNeighbors**(**hyper_dict*, ***kwargs*)
Common Neighbors.

Common Neighbors defines the similarity between nodes as the number of common neighbors between them.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.cn import CommonNeighbors
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = CommonNeighbors(4, 0.01)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                          is_weighted=True, no_python=True)
>>> print('Common Neighbors:
Training time: %f' % (time() - t1))
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.4 Graph Convolution Network

11.5 Graph Factorization

class gemben.embedding.gf.**GraphFactorization**(**hyper_dict*, ***kwargs*)
Graph Factorization.

Graph Factorization factorizes the adjacency matrix with regularization.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.gf import GraphFactorization
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = GraphFactorization(2, 100000, 1 * 10**-4, 1.0)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
>>> print ('Graph Factorization:Training time: %f' % (time() - t1))
>>> viz.plot_embedding2D(embedding.get_embedding(),
                         di_graph=G, node_colors=None)
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=True*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.6 HOPE

class gemben.embedding.hope.**HOPE**(**hyper_dict*, ***kwargs*)
Higher Order Proximity Preserving.

Higher Order Proximity factorizes the higher order similarity matrix between nodes using generalized singular value decomposition.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.hope import HOPE
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = HOPE(4, 0.01)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
>>> print('HOPE:Training time: %f' % (time() - t1))
>>> viz.plot_embedding2D(embedding.get_embedding()[:, :2],
                         di_graph=G, node_colors=None)
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.7 Jaccard Coefficient

class gemben.embedding.jc.**JaccardCoefficient**(**hyper_dict*, ***kwargs*)
Jaccard Coefficient.

Jaccard Coefficient measures the probability that two nodes i and j have a connection to node k, for a randomly selected node k from the neighbors of i and j .

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.jc import JaccardCoefficient
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = JaccardCoefficient(4, 0.01)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
>>> print('Adamic Adar:Training time: %f' % (time() - t1))
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.8 Laplacian Eigenmaps

class gemben.embedding.lap.**LaplacianEigenmaps**(**hyper_dict*, ***kwargs*)
Laplacian Eigenmaps.

LaplacianEigenmaps penalizes the weighted square of distance between neighbors. This is equivalent to factorizing the normalized Laplacian matrix.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.lap import LaplacianEigenmaps
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = LaplacianEigenmaps(2)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
>>> print('Laplacian Eigenmaps:Training time: %f' % (time() - t1))
>>> viz.plot_embedding2D(embedding.get_embedding(),
                         di_graph=G, node_colors=None)
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.9 Locally Linear Embedding

class gemben.embedding.lle.**LocallyLinearEmbedding**(**hyper_dict*, ***kwargs*)
Locally Linear Embedding.

Locally Linear Embedding uses d eigenvectors corresponding to eigenvalues from second smallest to $(d+1)^{th}$ smallest from the sparse matrix $(I - W)^\intercal(I - W)$. It assumes that the embedding of each node is a linear weighted combination of the neighbor's embeddings.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.lle import LocallyLinearEmbedding
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = LocallyLinearEmbedding(2)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
```

(continues on next page)

(continued from previous page)

```
>>> print('Graph Factorization:Training time: %f' % (time() - t1))
>>> viz.plot_embedding2D(embedding.get_embedding(),
                         di_graph=G, node_colors=None)
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.10 node2vec

class gemben.embedding.node2vec.**node2vec**(**hyper_dict*, ***kwargs*)
node2vec.

Node2Vec aim to learn a low-dimensional feature representation for nodes through a stream of random walks. These random walks explore the nodes' variant neighborhoods. Thus, random walk based methods are much more scalable for large graphs and they generate informative embeddings.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.node2vec import node2vec
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
```

(continues on next page)

(continued from previous page)

```
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = node2vec(2, 1, 80, 10, 10, 1, 1)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
>>> print('node2vec:Training time: %f' % (time() - t1))
>>> viz.plot_embedding2D(embedding.get_embedding(),
                         di_graph=G, node_colors=None)
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.11 Preferential Attachment

class gemben.embedding.pa.**PreferentialAttachment**(**hyper_dict*, ***kwargs*)
Preferential Attachment.

Preferential Attachment is based on the assumption that the connection to a node is proportional to its degree. It defines the similarity between the nodes as the product of their degrees.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.pa import PreferentialAttachment
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = PreferentialAttachment(2)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
>>> print('PreferentialAttachment:Training time: %f' % (time() - t1))
>>> viz.plot_embedding2D(embedding.get_embedding(),
                         di_graph=G, node_colors=None)
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.12 Random Embedding

class gemben.embedding.rand.**RandomEmb**(**hyper_dict*, ***kwargs*)
Random Embedding.

It generates the random embedding for the graph.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.
- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.rand import RandomEmb
>>> edge_f = 'data/karate.edgelist'
>>> G = graph_util.loadGraphFromEdgeListTxt(edge_f, directed=False)
>>> G = G.to_directed()
>>> res_pre = 'results/testKarate'
>>> graph_util.print_graph_stats(G)
>>> t1 = time()
>>> embedding = PreferentialAttachment(2)
>>> embedding.learn_embedding(graph=G, edge_f=None,
                              is_weighted=True, no_python=True)
>>> print('PreferentialAttachment:Training time: %f' % (time() - t1))
>>> viz.plot_embedding2D(embedding.get_embedding(),
                         di_graph=G, node_colors=None)
>>> plt.show()
```

get_edge_weight(*self*, *i*, *j*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *X=None*, *node_l=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.13 SDNE

class gemben.embedding.sdne.**SDNE**(**hyper_dict*, ***kwargs*)
SDNE.

SDNE uses a deep autoencoder to provide non-linear functions to preserve the first and second order proximities jointly.

Parameters

- **hyper_dict** (*object*) – Hyper parameters.

- **kwargs** (*dict*) – keyword arguments, form updating the parameters

Examples

```
>>> from gemben.embedding.sdne import SDNE
>>> file_prefix = "gemben/data/sbm/graph.gpickle"
>>> G = nx.read_gpickle(file_prefix)
>>> node_colors = pickle.load(
    open('gemben/data/sbm/node_labels.pickle', 'rb') )
>>> embedding = SDNE(d=128, beta=5, alpha=1e-5, nu1=1e-6, nu2=1e-6,
               K=3, n_units=[500, 300, ],
               n_iter=30, xeta=1e-3,
               n_batch=500,
               modelfile=['gemben/intermediate/enc_model.json',
                          'gemben/intermediate/dec_model.json'],
               weightfile=['gemben/intermediate/enc_weights.hdf5',
                           'gemben/intermediate/dec_weights.hdf5'])
>>> G_X = nx.to_numpy_matrix(G)
>>> embedding.learn_embedding(G)
>>> G_X_hat = embedding.get_reconstructed_adj()
>>> rec_norm = np.linalg.norm(G_X - G_X_hat)
>>> print(rec_norm)
>>> node_colors_arr = [None] * node_colors.shape[0]
>>> for idx in range(node_colors.shape[0]):
        node_colors_arr[idx] = np.where(node_colors[idx, :].toarray() == 1)[1][0]
>>> viz.plot_embedding2D(G_X,
                         di_graph=G,
                         node_colors=node_colors_arr)
>>> plt.savefig('sdne_sbm_g_x.pdf', bbox_inches='tight')
```

get_edge_weight(*self*, *i*, *j*, *embed=None*, *filesuffix=None*)
Compute the weight for edge between node i and node j

Parameters **j** (*i,*) – two node id in the graph for embedding

Returns A single number represent the weight of edge between node i and node j

get_embedding(*self*, *filesuffix=None*)
Returns the learnt embedding

Returns A numpy array of size #nodes * d

get_method_name(*self*)
Returns the name for the embedding method

Returns The name of embedding

get_method_summary(*self*)
Returns the summary for the embedding include method name and paramater setting

Returns A summary string of the method

get_reconstructed_adj(*self*, *embed=None*, *node_l=None*, *filesuffix=None*)
Compute the adjacency matrix from the learned embedding

Returns A numpy array of size #nodes * #nodes containing the reconstructed adjacency matrix.

learn_embedding(*self*, *graph=None*, *edge_f=None*, *is_weighted=False*, *no_python=False*)
Learning the graph embedding from the adjcency matrix.

Parameters **graph** – the graph to embed in networkx DiGraph format

11.14 Variational Auto-Encoder

CHAPTER

TWELVE

GEMBEN UTILITY FUNCTION

12.1 Static Graph Embedding utils

12.2 Visualization

gemben.evaluation.visualize_embedding.**expVis**(*X*, *res_pre*, *m_summ*, *node_labels=None*, *di_graph=None*)

Function used to visualize the experiment.

Parameters

- **X** (*Vetor*) – Embedding values of the nodes.
- **res_pre** (*Str*) – Prefix to be used to save the result.
- **m_summ** (*Str*) – String to denote the name of the summary file.
- **node_pos** (*Vector*) – High dimensional embedding values of each nodes.
- **node_labels** (*List*) – List consisting of node labels.
- **di_graph** (*Object*) – network graph object of the original network.

gemben.evaluation.visualize_embedding.**plot_embedding2D**(*node_pos*, *node_colors=None*, *di_graph=None*)

Function to plot the embedding in two dimension.

Parameters

- **node_pos** (*Vector*) – High dimensional embedding values of each nodes.
- **node_colors** (*List*) – List consisting of node colors.
- **di_graph** (*Object*) – network graph object of the original network.

12.3 Embeding Utility Function

gemben.utils.embed_util.**reorient**(*embed1*, *embed2*)

Function to re-orient the two embedding values by projection.

12.4 Bayesian Optimizer

class gemben.utils.bayesian_opt.**BayesianOpt**(**args*, ***kwargs*)
bayesian global optimization with Gaussian Process

Bayesian optimization is a module to perform hyper-paramter tuning. It can be utilized to find the best hyper-parameter with fewer iterations.

Examples

```
>>> from bayes_opt import BayesianOptimization
>>> def black_box_function(x, y):
                return x+y
>>> pbounds = {'x': (2, 4), 'y': (-3, 3)}
>>> optimizer = BayesianOptimization(f=black_box_function,
                                                                        ␣
↪pbounds=pbounds,
                                                                        verbose=2,
↪ # verbose = 1 prints only when a maximum is observed, verbose = 0 is silent
                                                                        random_
↪state=1,)
>>> optimizer.maximize(init_points=2,  n_iter=3,)
>>> print(optimizer.max)
```

Methods

optimization_func(self, **hyp_space)	The main method to be optimized
optimize(self[, random_state, verbose, ...])	Function to perform the actual optimization.
search_space(self, hyp_range)	Function to define the search space

optimization_func(*self*, ***hyp_space*)
The main method to be optimized

optimize(*self*, *random_state=5*, *verbose=2*, *init_points=10*, *n_iter=5*, *acq='poi'*)
Function to perform the actual optimization.

search_space(*self*, *hyp_range*)
Function to define the search space

12.5 Evaluation Utility Function

class gemben.utils.bayesian_opt.**BayesianOpt**(**args*, ***kwargs*)
bayesian global optimization with Gaussian Process

Bayesian optimization is a module to perform hyper-paramter tuning. It can be utilized to find the best hyper-parameter with fewer iterations.

Examples

```
>>> from bayes_opt import BayesianOptimization
>>> def black_box_function(x, y):
                return x+y
>>> pbounds = {'x': (2, 4), 'y': (-3, 3)}
>>> optimizer = BayesianOptimization(f=black_box_function,
                                                                                ␣
↪pbounds=pbounds,
                                                                                verbose=2,
↪ # verbose = 1 prints only when a maximum is observed, verbose = 0 is silent
                                                                                random_
↪state=1,)
>>> optimizer.maximize(init_points=2,  n_iter=3,)
>>> print(optimizer.max)
```

Methods

optimization_func(self, **hyp_space)	The main method to be optimized
optimize(self[, random_state, verbose, ...])	Function to perform the actual optimization.
search_space(self, hyp_range)	Function to define the search space

optimization_func (*self*, ***hyp_space*)
The main method to be optimized

optimize (*self*, *random_state=5*, *verbose=2*, *init_points=10*, *n_iter=5*, *acq='poi'*)
Function to perform the actual optimization.

search_space (*self*, *hyp_range*)
Function to define the search space

12.6 Graph Generation Functions

gemben.utils.graph_gens.**barabasi_albert_graph** (*N*, *deg*, *dia*, *dim*, *domain*)
Return random graph using Barabási-Albert preferential attachment model.

Parameters

- **n** (*int*) – Number of Nodes
- **deg** (*int*) – Degree of the graphs
- **dia** (*float*) – diameter of the graph
- **dim** (*int*) –
- **m** – Number of edges to attach from a new node to existing nodes
- **for m** (*Formula*) – (m^2)- (Nm)/2 + avg_deg * (N/2) = 0 => From this equation we need to find m :

:param : return: Graph Object

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**barbell_graph**(*m1*, *m2*)
Function to generate barbell graph.

A n-barbell graph is the simple graph obtained by connecting two copies of a complete graph K_n by a bridge.

Return the Barbell Graph: two complete graphs connected by a path.

For m1 > 1 and m2 >= 0.

Two identical complete graphs K_{m1} form the left and right bells, and are connected by a path P_{m2}.

The 2*m1+m2 nodes are numbered 0,...,m1-1 for the left barbell, m1,...,m1+m2-1 for the path, and m1+m2,...,2*m1+m2-1 for the right barbell.

gemben.utils.graph_gens.**binary_community_graph**(*N*, *k*, *maxk*, *mu*)
Retruns a binary community graph.

gemben.utils.graph_gens.**duplication_divergence_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Returns an undirected graph using the duplication-divergence model.

A graph of n nodes is created by duplicating the initial nodes and retaining edges incident to the original nodes with a retention probability p.

Parameters of the graph: n (int) – The desired number of nodes in the graph. p (float) – The probability for retaining the edge of the replicated node.

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**hyperbolic_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
The Hyperbolic Generator distributes points in hyperbolic space and adds edges between points with a probability depending on their distance. The resulting graphs have a power-law degree distribution, small diameter and high clustering coefficient. For a temperature of 0, the model resembles a unit-disk model in hyperbolic space.

Parameters of the graph: N = Num of nodes k = Average degree gamma = Target exponent in Power Law Distribution

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**lfr_benchmark_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Returns the LFR benchmark graph for testing community-finding algorithms.

Parameters of the graph: n (int) – Number of nodes in the created graph. tau1 (float) – Power law exponent for the degree distribution of the created graph. This value must be strictly greater than one. tau2 (float) – Power

law exponent for the community size distribution in the created graph. This value must be strictly greater than one. mu (float) – Fraction of intra-community edges incident to each node. This value must be in the interval [0, 1]. average_degree (float) – Desired average degree of nodes in the created graph. This value must be in the interval [0, n]. Exactly one of this and min_degree must be specified, otherwise a NetworkXError is raised.

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**plot_hist**(*title*, *data*)
Function to truncate the given floating point values.

gemben.utils.graph_gens.**powerlaw_cluster_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Holme and Kim algorithm for growing graphs with powerlaw degree distribution and approximate average clustering.

The average clustering has a hard time getting above a certain cutoff that depends on `m`. This cutoff is often quite low. The transitivity (fraction of triangles to possible triangles) seems to decrease with network size.

It is essentially the Barabási–Albert (BA) growth model with an extra step that each random edge is followed by a chance of making an edge to one of its neighbors too (and thus a triangle).

This algorithm improves on BA in the sense that it enables a higher average clustering to be attained if desired.

It seems possible to have a disconnected graph with this algorithm since the initial `m` nodes may not be all linked to a new node on the first iteration like the BA model.

Parameters of the graph: n (int) – the number of nodes m (int) – the number of random edges to add for each new node p (float,) – Probability of adding a triangle after adding a random edge Formula for m: (m^2)- (Nm)/2 + avg_deg * (N/2) = 0 => From this equation we need to find m : p : Does not vary the average degree or diameter so much. : Higher value of p may cause average degree to overshoot intended average_deg so we give the control of average degree to parameter m: by setting a lower value of p: 0.1

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**r_mat_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Generates static R-MAT graphs.

R-MAT (recursive matrix) graphs are random graphs with n=2^scale nodes and m=n*edgeFactor edges. More details at http://www.graph500.org or in the original paper: Deepayan Chakrabarti, Yiping Zhan, Christos Faloutsos: R-MAT: A Recursive Model for Graph Mining.

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**random_geometric_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Return the random geometric graph in the unit cube.

The random geometric graph model places n nodes uniformly at random in the unit cube Two nodes *u,v* are connected with an edge if *d(u,v)<=r* where *d* is the Euclidean distance and *r* is a radius threshold.

Average Degree is given by formula: Avg_Deg = (pi*(r^2)*num_nodes)/(l^2) Formula for r: avg_deg * l where l can be considered a constant where its square can be approximated to 1.04 [ength of square] Empirically Found

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**stochastic_block_model**(*N*, *deg*, *dia*, *dim*, *domain*)
Returns a stochastic block model graph.

This model partitions the nodes in blocks of arbitrary sizes, and places edges between pairs of nodes independently, with a probability that depends on the blocks.

Parameters

- **N** – Number of Nodes
- **p** – Element (r,s) gives the density of edges going from the nodes of group r to nodes of group s. p must match the number of groups (len(sizes) == len(p)), and it must be symmetric if the graph is undirected.

Formula for p: Through Empirical Studies - p = 0.001 * Deg gives perfect result for Num_of_Nodes = 1024
But if N >1024: scaler = N/1024 : then p = (0.001*deg)/scaler And if N < 1024 : Scaler = 1024/N : then p = (0.001*deg)*scaler and if N == 1024: p = (0.001*deg)

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**stochastic_kronecker_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Generates stochastic kronecker graph.

The stochastic Kronecker graph model introduced by Leskovec etal. is a random graph with vertex setZn2, where two verticesuandvare connected with probability *u·v(1u)·(1v)nu·v(1u)·(1v)* in-dependently of the presence or absence of any other edge, for fixedparameters *0< ,, <1*. They have shown empirically that the de-gree sequence resembles a power law degree distribution. In this paperwe show that the stochastic Kronecker graph a.a.s. does not feature apower law degree distribution for any parameters *0< ,, <1*.

Parameters of the graph: degree_seq

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**truncate**(*f*, *n*)
Function to truncate the given floating point values.

gemben.utils.graph_gens.**watts_strogatz_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Return a Watts-Strogatz small-world graph.

First create a ring over n nodes. Then each node in the ring is connected with its k nearest neighbors (k-1 neighbors if k is odd). Then shortcuts are created by replacing some edges as follows: for each edge u-v in the underlying "n-ring with k nearest neighbors" with probability p replace it with a new edge u-w with uniformly random choice of existing node w.

Parameters of the graph: n (int) – The number of nodes k (int) – Each node is joined with its k nearest neighbors in a ring topology. p (float) – The probability of rewiring each edge

Average Degree is solely decided by k Diameter depends on the value of p

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

gemben.utils.graph_gens.**waxman_graph**(*N*, *deg*, *dia*, *dim*, *domain*)
Return a Waxman random graph.

The Waxman random graph models place n nodes uniformly at random in a rectangular domain. Two nodes u,v are connected with an edge with probability

Parameters of the graph: n (int or iterable) – Number of nodes or iterable of nodes

beta (float) – Model parameter

alpha (float) – Model parameter

Average Degree is given by formula: k where P = beta * exp(-d/alpha*L) alpha = (gamma((k/2)+1) * (beta^k))/((n-1)*(pi^(k/2))*gamma(k)) where beta is chosen randomly to satisfy the average degree criterion So we fix the parameter beta = 0.1, and we know the default value of d/L is in range: 0.25 to 0.3 (Empiricially calculated) so we only tweak alpha to get the required avg deg.

Parameters

- **n** (*int or iterable*) –
- **dia** (*float*) –
- **dim** (*int, optional*) – Dimension of the graph
- **domain** (*str, optional*) – Domain of the graph

Returns Best graph, beast average degree and best diameter.

Return type Object

12.7 Graph Utility Functions

gemben.utils.graph_util.**addChaos**(*di_graphs*, *k*)
Function to add randomness in the graph.

gemben.utils.graph_util.**addNodeAnomalies**(*di_graphs*, *p*, *k*)
Function to add anomalous edges in the graph.

gemben.utils.graph_util.**print_graph_stats**(*G*)
Function to print the graph statistics.

gemben.utils.graph_util.**randwalk_DiGraph_to_adj**(*di_graph*, *node_frac=0.1*, *n_walks_per_node=5*, *len_rw=2*)
Function to perform a randomwalk on directed graph and return the adjacency list.

gemben.utils.graph_util.**sample_graph**(*di_graph*, *n_sampled_nodes=None*)
Function to sample the graph.

gemben.utils.graph_util.**sample_graph_rw**(*di_graph*, *n_sampled_nodes=None*, *random_res_p=0.01*, *verbose=False*)
Function to return the sampled graph.

gemben.utils.graph_util.**transform_DiGraph_to_adj**(*di_graph*)
Function to convert the directed graph to adjacency matrix.

gemben.utils.graph_util.**transform_adj_to_DiGraph**(*adj*)
Function to convert the adjacency matrix into directed graph.

12.8 Kronecker Graph Generator

gemben.utils.kronecker_generator.**convert**(*something*)
Function to convert the numpy array to networkx graph.

gemben.utils.kronecker_generator.**deleteSelfLoops**(*graph*, *nNodes*)
Function to take away self loops in final graph for stat purposes.

gemben.utils.kronecker_generator.**generateStochasticKron**(*initMat*, *k*, *deleteSelfLoopsForStats=False*, *directed=False*, *customEdges=False*, *edges=0*)

Function to generate stochastic kronecker graph.

12.9 Kronecker Matrix Initializer

class gemben.utils.kronecker_init_matrix.**InitMatrix**(*numNodes*, *W=None*)

Class module to initialize the kronechker graph.

Methods

addEdge	
addSelfEdges	
getMtxSum	
getNumNodes	
getValue	
make	
makeFromNetworkxGraph	
makeStochasticAB	
makeStochasticABFromNetworkxGraph	
makeStochasticCustom	
setNumNodes	
setValue	

12.10 lrf graph generator

gemben.utils.lrf_gen.**average_degree**(*dmax*, *dmin*, *gamma*)

Function to calculate average degree.

gemben.utils.lrf_gen.**benchmark**(*excess*, *defect*, *num_nodes*, *average_k*, *max_degree*, *tau*, *tau2*, *mixing_parameter*, *overlapping_nodes*, *overlap_membership*, *nmin*, *nmax*, *fixed_range*, *clustering_coeff*)

Function to solve for minimum degree for the given benchmark.

gemben.utils.lrf_gen.**integral**(*a*, *b*)

Function to evaulate the integral.

gemben.utils.lrf_gen.**solve_dmin**(*dmax*, *dmed*, *gamma*)

Function to solve for minimum degree.

12.11 Plotting Utility Function

gemben.utils.plot_util.**get_node_color**(*node_community*)

Function to get the node colors for the communities.

gemben.utils.plot_util.**plot**(*x_s*, *y_s*, *fig_n*, *x_lab*, *y_lab*, *file_save_path*, *title*, *legendLabels=None*, *show=False*)
Function to plot the graph with respective embeddings.

gemben.utils.plot_util.**plotExpRes**(*res_pre*, *methods*, *exp*, *d_arr*, *save_fig_pre*, *n_rounds*, *plot_d*, *plot_ratio=0.8*, *samp_scheme='u_rand'*, *K=1024*)
Function to plot experiment results for maps.

gemben.utils.plot_util.**plot_F1**(*res_pre*, *res_suffix*, *exp_type*, *m_names_f*, *m_names*, *d_arr*, *n_rounds*, *save_fig_name*, *K=1024*, *plot_d=False*)
Function to plot the F1-score.

gemben.utils.plot_util.**plot_hyp**(*hyp_keys*, *exp_param*, *meth*, *data*, *s_sch='u_rand'*)
Function to explore the hyperparameters.

gemben.utils.plot_util.**plot_hyp_all**(*hyp_keys*, *exp_param*, *meth*, *data_sets*, *s_sch='u_rand'*)
Function to plot all the hyper-parameter results.

gemben.utils.plot_util.**plot_hyp_data**(*hyp_keys*, *exp_param*, *meths*, *data*, *s_sch='u_rand'*, *dim=2*)
Function to plot the result of hyperparameter exploration.

gemben.utils.plot_util.**plot_hyp_data2**(*hyp_keys*, *exp_param*, *meths*, *data*, *s_sch='u_rand'*, *dim=2*)
Function to plot the result of hyper-parameter exploration.

gemben.utils.plot_util.**plot_p_at_k**(*res_pre*, *res_suffix*, *exp_type*, *m_names_f*, *m_names*, *d_arr*, *n_rounds*, *save_fig_name*, *K=1024*, *plot_d=False*, *plot_ratio=0.8*, *s_sch='u_rand'*)
Function to plot precision at k.

gemben.utils.plot_util.**plot_ts**(*ts_df*, *plot_title*, *eventDates*, *eventLabels=None*, *save_file_name=None*, *xLabel=None*, *yLabel=None*, *show=False*)
Function to plot the time series data.

gemben.utils.plot_util.**turn_latex**(*key_str*)
Function to convert special words to latex comaptible ones.

CHAPTER

THIRTEEN

EVALUATION MODULE

13.1 Graph Reconstruction

gemben.evaluation.evaluate_graph_reconstruction.**evaluateStaticGraphReconstruction**(*digraph*, *graph_embe*
X_stat,
node_l=Non
file_suffix=N
sam-
ple_ratio_e=
is_undirecte
is_weighted=

This function evaluates the graph reconstruction accuracy of the embedding algorithms.

Parameters

- **digraph** (*Object*) – directed networkx graph object.
- **graph_embedding** (*object*) – Object of the embedding algorithm class defined in gemben/embedding.
- **X_stat** (*Vector*) – Embedding of the the nodes of the graph.
- **node_l** (*Int*) – Number of nodes in the graph.
- **file_suffix** (*Str*) – The name of the algorithm and dataset used to save the embedding.
- **sample_ratio_e** (*Float*) – The ratio used to sample the original graph for evaluation purpose.
- **is_undirected** (*bool*) – Boolean flag to denote whether the graph is directed or not.
- **is_weighted** (*bool*) – Boolean flag to denote whether the edges of the graph is weighted.

Returns Consiting of Mean average precision precision curve, errors and error baselines.

Return type Numpy Array

gemben.evaluation.evaluate_graph_reconstruction.**expGR**(*digraph*, *graph_embedding*, *X*, *n_sampled_nodes_l*, *rounds*, *res_pre*, *m_summ*, *K=10000*, *is_undirected=True*, *sampling_scheme='u_rand'*)

This function is used to experiment graph reconstruction.

Parameters

- **digraph** (*Object*) – directed networkx graph object.
- **graph_embedding** (*object*) – Object of the embedding algorithm class defined in gemben/embedding.
- **X** (*Vector*) – Embedding of the the nodes of the graph.
- **n_sampled_node_l** (*Int*) – Number of nodes in the graph.
- **rounds** (*Int*) – The number of times the graph reconstruction is performed.
- **res_pre** (*Str*) – Prefix to be used to save the result.
- **m_summ** (*Str*) – String to denote the name of the summary file.
- **K** (*Int*) – The maximum value to be use to get the precision curves.
- **sampling_scheme** (*Str*) – Sampling schme used to sample nodes to be reconstructed.
- **is_undirected** (*bool*) – Boolean flag to denote whether the graph is directed or not.

Returns Consisting of Mean average precision.

Return type Numpy Array

13.2 Link Prediction

gemben.evaluation.evaluate_link_prediction.**evaluateStaticLinkPrediction**(*train_digraph*, *test_digraph*, *graph_embedding*, *X*, *node_l=None*, *sample_ratio_e=None*, *is_undirected=True*, *store_predictions=1*)

This function evaluates the static link prediction accuracy of the embedding algorithms.

Parameters

- **train_digraph** (*Object*) – directed networkx graph object used for training the algorithm.
- **test_digraph** (*Object*) – directed networkx graph object to be used for testing the algorithm.
- **graph_embedding** (*object*) – Object of the embedding algorithm class defined in gemben/embedding.
- **X** (*Vector*) – Embedding of the the nodes of the graph.
- **node_l** (*Int*) – Number of nodes in the graph.
- **sample_ratio_e** (*Float*) – The ratio used to sample the original graph for evaluation purpose.
- **is_undirected** (*bool*) – Boolean flag to denote whether the graph is directed or not.
- **store_prediction** (*Int*) – Stores the predicted values.

Returns Consiting of Mean average precision and the precision curve values.

Return type Numpy Array

gemben.evaluation.evaluate_link_prediction.**expLP**(*digraph*, *graph_embedding*, *n_sample_nodes_l*, *rounds*, *res_pre*, *m_summ*, *train_ratio=0.8*, *no_python=True*, *K=32768*, *is_undirected=True*, *sampling_scheme='u_rand'*)

This function is used to experiment link prediction.

Parameters

- **digraph** (*Object*) – directed networkx graph object.
- **graph_embedding** (*object*) – Object of the embedding algorithm class defined in gemben/embedding.
- **n_sampled_node_l** (*Int*) – Number of nodes in the graph.
- **rounds** (*Int*) – The number of times the graph reconstruction is performed.
- **res_pre** (*Str*) – Prefix to be used to save the result.
- **train_ratio** (*Float*) – The split used for dividing the traing and testing data.
- **no_python** (*Bool*) – Flag to denote if python is used.
- **m_summ** (*Str*) – String to denote the name of the summary file.
- **K** (*Int*) – The maximum value to be use to get the precision curves.
- **sampling_scheme** (*Str*) – Sampling schme used to sample nodes to be reconstructed.
- **is_undirected** (*bool*) – Boolean flag to denote whether the graph is directed or not.

Returns Consisting of Mean average precision.

Return type Numpy Array

gemben.evaluation.evaluate_link_prediction.**expLPT**(*digraph*, *graph_embedding*, *res_pre*, *m_summ*, *K=100000*, *is_undirected=True*)

This function is used to experiment graph reconstruction for temporally varying graphs.

Parameters

- **digraph** (*Object*) – directed networkx graph object.
- **graph_embedding** (*object*) – Object of the embedding algorithm class defined in gemben/embedding.
- **res_pre** (*Str*) – Prefix to be used to save the result.
- **m_summ** (*Str*) – String to denote the name of the summary file.
- **K** (*Int*) – The maximum value to be use to get the precision curves.
- **is_undirected** (*bool*) – Boolean flag to denote whether the graph is directed or not.

13.3 Node Classification

class gemben.evaluation.evaluate_node_classification.**TopKRanker**(*estimator*, *n_jobs=None*)

Class to get top K ranks.

Attributes

coef_

intercept_

multilabel_ Whether this is a multilabel classifier

n_classes_

Methods

decision_function(self, X)	Returns the distance of each sample from the decision boundary for each class.
fit(self, X, y)	Fit underlying estimators.
get_params(self[, deep])	Get parameters for this estimator.
partial_fit(self, X, y[, classes])	Partially fit underlying estimators
predict(self, X, top_k_list)	This function returns the prediction for top k node labels.
predict_proba(self, X)	Probability estimates.
score(self, X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.
set_params(self, **params)	Set the parameters of this estimator.

predict (*self*, *X*, *top_k_list*)

This function returns the prediction for top k node labels.

Parameters

- **X** (*Vector*) – Embedding of the nodes.
- **top_k_list** (*List*) – list consisting of value to denote top k.

Returns Predicted node labels.

Return type Numpy Array

gemben.evaluation.evaluate_node_classification.**evaluateNodeClassification**(*X*, *Y*, *test_ratio*)

This function is used to evaluate node classification.

Parameters

- **X** (*Vector*) – Embedding values of the nodes.
- **Y** (*Int*) – Labels of the nodes.
- **test_ratio** (*Float*) – Ratio to split the training and testing nodes.

Returns Micro and macro accuracy scores.

Return type Numpy Array

gemben.evaluation.evaluate_node_classification.**expNC**(*X*, *Y*, *test_ratio_arr*, *rounds*, *res_pre*, *m_summ*)

This function is used to experiment node classification.

Parameters

- **X** (*vector*) – Embedding values of the nodes.
- **Y** (*Int*) – Labels of the nodes.

- **rounds** (*Int*) – The number of times the graph reconstruction is performed.
- **res_pre** (*Str*) – Prefix to be used to save the result.
- **test_ratio_arr** (*Float*) – The split used for dividing the traing and testing data.
- **m_summ** (*Str*) – String to denote the name of the summary file.

Returns Average accuracy.

Return type Numpy Array

13.4 Evaluation Metrics

gemben.evaluation.metrics.**computeMAP**(*predicted_edge_list*, *true_digraph*, *max_k=-1*)
This function computers the Mean average precision.

Parameters

- **predicted_edge_list** (*Array*) – Consists of predicted edge list for each node.
- **true_digraph** (*object*) – True network graph object consists of the original nodes and edges.
- **max_k** (*Int*) – Maximum number of edges to be considered for computing the precsion.

Returns MAP values.

Return type Array

gemben.evaluation.metrics.**computePrecisionCurve**(*predicted_edge_list*, *true_digraph*, *max_k=-1*)
This function computers the precision curves.

Parameters

- **predicted_edge_list** (*Array*) – Consists of predicted edge list for each node.
- **true_digraph** (*object*) – True network graph object consists of the original nodes and edges.
- **max_k** (*Int*) – Maximum number of edges to be considered for computing the precsion.

Returns Precision scores and delta factors.

Return type Array

gemben.evaluation.metrics.**getMetricsHeader**()
This function gets the header for the calculated predcison@k.

Returns Header

Return type String

gemben.evaluation.metrics.**getPrecisionReport**(*prec_curv*, *edge_num*)
Function to generate the precision report..

Parameters

- **prec_curv** (*Array*) – Calculated precision curve.
- **edge_num** (*Int*) – Total number of the edges.

Returns Precision report.

Return type String

CHAPTER

FOURTEEN

EXPERIMENT MODULE

14.1 Simple Experiments

A simple experiment to run the gemben library.

14.2 Bayesian Hyperparameter Optimizer

Experiment to utilize the bayesian hyperparameter tuning for the algorithms.

14.3 Full Experiments with Benchmark

Example to run the benchmark across all the baseline embedding algorithms.

CHAPTER

FIFTEEN

GENERAL EXAMPLES

General-purpose and introductory examples for the *GEM-Benchmark* library.

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

15.1 Experiment with Benchmark

Example to run the benchmark across all the baseline embedding algorithms.

```
from subprocess import call
import itertools
try: import cPickle as pickle
except: import pickle
import json
from argparse import ArgumentParser
import networkx as nx
import pandas as pd
import pdb
import os
import sys
from time import time
# sys.path.insert(0, './')
from gemben.utils import graph_gens

methClassMap = {"gf": "GraphFactorization",
                "hope": "HOPE",
                "lap": "LaplacianEigenmaps",
                "node2vec": "node2vec",
                "sdne": "SDNE",
                "pa": "PreferentialAttachment",
                "rand": "RandomEmb",
                "cn": "CommonNeighbors",
                "aa": "AdamicAdar",
                "jc": "JaccardCoefficient"}

if __name__ == "__main__":
    ''' Sample usage
    python experiments/exp_synthetic.py -syn_names all -plot_hyp_data 1 -meths all
    '''
    t1 = time()
    parser = ArgumentParser(description='Graph Embedding Benchmark Experiments')
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('-data', '--data_sets',
                        help='dataset names (default: barabasi_albert_graph)')
    parser.add_argument('-dims', '--dimensions',
                        help='embedding dimensions list(default: 128)')
    parser.add_argument('-meth', '--methods',
                        help='method list (default: all methods)')
    parser.add_argument('-plot_hyp_data', '--plot_hyp_data',
                        help='plot the hyperparameter results (default: False)')
    parser.add_argument('-rounds', '--rounds',
                        help='number of rounds (default: 20)')
    parser.add_argument('-s_sch', '--samp_scheme',
                        help='sampling scheme (default: rw)')
    parser.add_argument('-lexp', '--lexp',
                        help='load experiment (default: False)')
    params = json.load(
        open('gemben/experiments/config/params_benchmark.conf', 'r')
    )
    args = vars(parser.parse_args())
    print (args)
    syn_hyps = json.load(
        open('gemben/experiments/config/syn_hypRange.conf', 'r')
    )
    for k, v in args.items():
        if v is not None:
            params[k] = v
    params["rounds"] = int(params["rounds"])
    if params["data_sets"] == "all":
        params["data_sets"] = syn_hyps.keys()
    else:
        params["data_sets"] = params["data_sets"].split(',')
    params["lexp"] = bool(int(params["lexp"]))
    params["plot_hyp_data"] = bool(int(params["plot_hyp_data"]))
    if params["methods"] == "all":
        params["methods"] = methClassMap.keys()
    else:
        params["methods"] = params["methods"].split(',')
    params["dimensions"] = params["dimensions"].split(',')
    samp_scheme = params["samp_scheme"]
    for syn_data in params["data_sets"]:
        syn_hyp_range = syn_hyps[syn_data]
        hyp_keys = list(syn_hyp_range.keys())
        if syn_data == "binary_community_graph":
            graphClass = getattr(graph_gens, syn_data)
        else:
            graphClass = getattr(nx, syn_data)
        ev_cols = ["GR MAP", "LP MAP", "LP P@100", "NC F1 score"]
        for dim in params["dimensions"]:
            dim = int(dim)
            for meth in params["methods"]:
                if not params["lexp"]:
                    hyp_df = pd.DataFrame(
                        columns=hyp_keys + ev_cols + ["Round Id"]
                    )
                    hyp_r_idx = 0
                    for hyp in itertools.product(*syn_hyp_range.values()):
                        hyp_dict = dict(zip(hyp_keys, hyp))
                        hyp_str = '_'.join(
```

(continues on next page)

(continued from previous page)

```
                            "%s=%r" % (key, val) for (key, val) in hyp_dict.items()
                        )
                        syn_data_folder = 'benchmark_%s_%s' % (syn_data, hyp_str)
                        hyp_df_row = dict(zip(hyp_keys, hyp))
                        for r_id in range(params["rounds"]):
                            G = graphClass(**hyp_dict)
                            if not os.path.exists("gemben/data/%s" % syn_data_folder):
                                os.makedirs("gemben/data/%s" % syn_data_folder)
                            nx.write_gpickle(
                                G, 'gemben/data/%s/graph.gpickle' % syn_data_folder
                            )
                            os.system(
                                "python gem/experiments/exp.py -data %s -meth %s -dim
↪%d -rounds 1 -s_sch %s -exp lp" % (syn_data_folder, meth, dim, samp_scheme)
                            )
                            MAP, prec, n_samps = pickle.load(
                                open('gemben/results/%s_%s_%d_%s.lp' % (syn_data_
↪folder, meth, dim, samp_scheme), 'rb')
                            )
                            hyp_df.loc[hyp_r_idx, hyp_keys] = \
                                pd.Series(hyp_df_row)
                            prec_100 = prec[int(n_samps[0])][0][100]
                            hyp_df.loc[hyp_r_idx, ev_cols + ["Round Id"]] = \
                                [0, MAP[int(n_samps[0])][0], prec_100, 0, r_id]
                            hyp_r_idx += 1
                    hyp_df.to_hdf(
                        "gemben/intermediate/%s_%s_lp_%s_dim_%d_data_hyp.h5" % (syn_
↪data, meth, samp_scheme, dim),
                        "df"
                    )
            if params["plot_hyp_data"]:
                from gem.utils import plot_util
                plot_util.plot_hyp_data2(
                    hyp_keys, ["lp"], params["methods"], syn_data, samp_scheme, dim
                )
    print('Total time taken: %f sec' % (time() - t1))
```

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

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

15.2 Module to run a single or multiple examples

Module to run the benchmark across all the baseline embedding algorithms.

```
from subprocess import call
import itertools
try: import cPickle as pickle
except: import pickle
import json
import networkx as nx
import pandas as pd
import pdb
```

(continues on next page)

(continued from previous page)

```
import os
import sys
from time import time
from gemben.utils import graph_gens

methClassMap = {"gf": "GraphFactorization",
                "hope": "HOPE",
                "lap": "LaplacianEigenmaps",
                "node2vec": "node2vec",
                "sdne": "SDNE",
                "pa": "PreferentialAttachment",
                "rand": "RandomEmb",
                "cn": "CommonNeighbors",
                "aa": "AdamicAdar",
                "jc": "JaccardCoefficient"}

class exp:
    def __init__(self, domain="social", method="sdne", rounds=1,lexp=False, samp_
↪scheme='rw', plot_hyp_data=False):

        t1 = time()
        self.params = json.load(
            open('gemben/experiments/config/params_benchmark.conf', 'r')
        )
        self.domain_graph_map = json.load(
            open('gemben/experiments/config/domain_graph_map.conf', 'r')
        )
        # graph_hyp_range: {N: [128, 256, 512, 1024], deg: [4, 6,8, 10, 12]}
        self.graph_hyp_range = json.load(
            open('gemben/experiments/config/graph_hyp_range.conf', 'r')
        )
        # def_graph_hyps: {N: 1024, deg: 8, dia: None, dim: 128}
        self.def_graph_hyps = json.load(
            open('gemben/experiments/config/def_graph_hyps.conf', 'r')
        )

        self.params["rounds"] = rounds
        self.params["graphs"] = self.domain_graph_map[domain]
        self.params["lexp"] = lexp
        self.params["plot_hyp_data"] = plot_hyp_data
        if method == "all":
            self.params["methods"] = methClassMap.keys()
        elif len(method)>1:
            self.params["methods"] = method.split(',')
        else:
            self.params["methods"] = self.method
        self.samp_scheme = samp_scheme


    def run(self):

        try:
          os.makedirs("gemben/intermediate")
        except:
          pass
        try:
          os.makedirs("gemben/results")
```

(continues on next page)

(continued from previous page)

```
        except:
          pass
        try:
          os.makedirs("gemben/temp")
        except:
          pass

        graph_hyp_keys = list(self.graph_hyp_range.keys())
        ev_cols = ["LP MAP", "LP P@100"]
        for meth , graph in itertools.product(*[self.params["methods"],self.params[
↪"graphs"]]):
            hyp_df = pd.DataFrame(
                    columns=graph_hyp_keys + ev_cols + ["Round Id"]
                )
            hyp_r_idx = 0
            for hyp_key in graph_hyp_keys:

                for curr_hyp_key_range, r_id in itertools.product(
                    *[graph_hyp_range[hyp_key], range(self.params["rounds"])]
                ):

                    if r_id == 0:
                        f_hyp = 1
                    else:
                        f_hyp = 0

                    curr_hyps = self.def_graph_hyps.copy()

                    curr_hyps[hyp_key] = curr_hyp_key_range
                    curr_hyps["domain"] = self.params["domain_name"]
                    hyp_str = '_'.join(
                                "%s=%s" % (key, str(val).strip("'")) for (key, val)␣
↪in curr_hyps.items()
                            )

                    hyp_str_graph_name = '_'.join(
                                "%s=%s" % (key, str(val).strip("'")) for (key, val)␣
↪in curr_hyps.items() if key != 'dim'
                            )

                    syn_data_folder = 'benchmark_%s_%s_%s' % (graph, hyp_str_graph_
↪name, r_id)

                    graphClass = getattr(graph_gens, graph)

                    try:
                        nx.read_gpickle(
                                'gemben/data/%s/graph.gpickle' % syn_data_folder
                      )
                    except:
                        flag =  1
                        ##### flag = 0 means the labels are continous on lcc
                        while flag:
                            print("Graph is generating...")
                            G = graphClass(**curr_hyps)[0]
                            if len(set(G.nodes())) == G.number_of_nodes() and list(G.
↪nodes())[-1] == G.number_of_nodes() -1:
```

(continues on next page)

(continued from previous page)

```
                                flag = 0
                        if G:
                            if not os.path.exists("gemben/data/%s" % syn_data_folder):
                                os.makedirs("gemben/data/%s" % syn_data_folder)
                            nx.write_gpickle(
                                    G, 'gemben/data/%s/graph.gpickle' % syn_data_
↪folder
                        )
                    perf_exp = not self.params["lexp"]
                    if self.params["lexp"]:
                      try:
                        MAP, prec, n_samps = pickle.load(
                          open('gemben/results/%s_%s_%d_%s.lp' % (
                              syn_data_folder, meth,
                              curr_hyps["dim"], self.samp_scheme), 'rb'))
                      except:
                          perf_exp = 1
                           ##### only find the best hyp for first round
                    if perf_exp:
                        os.system(
                          "python3 gemben/experiments/exp.py -data %s -meth %s -dim
↪%d -rounds 1 -find_hyp %d -s_sch %s -exp lp" % (
                              syn_data_folder,
                              meth,
                              curr_hyps["dim"],
                              f_hyp,
                              self.samp_scheme
                          )
                        )
                    MAP, prec, n_samps = pickle.load(
                    open('gemben/results/%s_%s_%d_%s.lp' % (
                        syn_data_folder, meth,
                        curr_hyps["dim"], self.samp_scheme), 'rb'))
                    hyp_df.loc[hyp_r_idx, graph_hyp_keys] = \
                    pd.Series(curr_hyps)
                    #prec_100 = prec[int(n_samps[0])][0][100]
                    try:
                      prec_100 = list(prec.values())[0][0][100]
                    except:
                      pdb.set_trace()
                    f_temp = open("gemben/temp/%s_%s_%s_lp_%s_data_hyp.txt" % (
                      self.params["domain_name"], graph, meth, self.samp_scheme), 'a')
                    f_temp.write('%s: round: %d, MAP: %f, prec_100: %f' % (hyp_str, r_
↪id, list(MAP.values())[0][0], prec_100))
                    f_temp.close()
                    hyp_df.loc[hyp_r_idx, ev_cols + ["Round Id"]] = \
                    [list(MAP.values())[0][0], prec_100, r_id]
                    #[MAP[int(n_samps[0])][0], prec_100, r_id]
                    hyp_r_idx += 1

            hyp_df.to_hdf(
                "gemben/intermediate/%s_%s_%s_lp_%s_data_hyp.h5" % (
                    self.params["domain_name"], graph, meth, self.samp_scheme),
                "df"
            )
            print('Experiments done for %s, %s' % (graph, meth))
```

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

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

15.3 Experiment with Benchmark

Example to run the benchmark across all the baseline embedding algorithms.

```
from subprocess import call
import itertools
try: import cPickle as pickle
except: import pickle
import json
from argparse import ArgumentParser
import networkx as nx
import pandas as pd
import pdb
import os
import sys
from time import time
# sys.path.insert(0, './')
from gemben.utils import graph_gens

methClassMap = {"gf": "GraphFactorization",
                "hope": "HOPE",
                "lap": "LaplacianEigenmaps",
                "node2vec": "node2vec",
                "sdne": "SDNE",
                "pa": "PreferentialAttachment",
                "rand": "RandomEmb",
                "cn": "CommonNeighbors",
                "aa": "AdamicAdar",
                "jc": "JaccardCoefficient"}


if __name__ == "__main__":
    ''' Sample usage
    python experiments/exp_synthetic.py -syn_names all -plot_hyp_data 1 -meths all
    '''
    t1 = time()
    parser = ArgumentParser(description='Graph Embedding Benchmark Experiments')
    parser.add_argument('-domain', '--domain_name',
                        help='domain name (default: social)')
    parser.add_argument('-graph', '--graphs',
                        help='graph name (default: all)')
    parser.add_argument('-meth', '--methods',
                        help='method list (default: all)')
    parser.add_argument('-plot_hyp_data', '--plot_hyp_data',
                        help='plot the hyperparameter results (default: False)')
    parser.add_argument('-rounds', '--rounds',
                        help='number of rounds (default: 20)')
    parser.add_argument('-s_sch', '--samp_scheme',
                        help='sampling scheme (default: rw)')
    parser.add_argument('-lexp', '--lexp',
```

(continues on next page)

(continued from previous page)

```
                        help='load experiment (default: False)')
    params = json.load(
        open('gemben/experiments/config/params_benchmark.conf', 'r')
    )
    args = vars(parser.parse_args())
    print (args)
    domain_graph_map = json.load(
        open('gemben/experiments/config/domain_graph_map.conf', 'r')
    )
    # graph_hyp_range: {N: [128, 256, 512, 1024], deg: [4, 6,8, 10, 12]}
    graph_hyp_range = json.load(
        open('gemben/experiments/config/graph_hyp_range.conf', 'r')
    )
    # def_graph_hyps: {N: 1024, deg: 8, dia: None, dim: 128}
    def_graph_hyps = json.load(
        open('gemben/experiments/config/def_graph_hyps.conf', 'r')
    )
    for k, v in args.items():
        if v is not None:
            params[k] = v
    params["rounds"] = int(params["rounds"])
    #params["domain_name"] = params["domain_name"].split(',')
    if params["graphs"] == "all":
        params["graphs"] = domain_graph_map[params["domain_name"]]
    else:
        params["graphs"] = params["graphs"].split(',')
    params["lexp"] = bool(int(params["lexp"]))
    params["plot_hyp_data"] = bool(int(params["plot_hyp_data"]))
    if params["methods"] == "all":
        params["methods"] = methClassMap.keys()
    else:
        params["methods"] = params["methods"].split(',')
    samp_scheme = params["samp_scheme"]









    try:
      os.makedirs("gemben/intermediate")
    except:
      pass
    try:
      os.makedirs("gemben/results")
    except:
      pass
    try:
      os.makedirs("gemben/temp")
    except:
      pass
#     if not os.path.exists("gem/intermediate"):
#         os.makedirs("gem/intermediate")
#     if not os.path.exists("gem/results"):
#         os.makedirs("gem/results")

```

(continues on next page)

(continued from previous page)

```
    graph_hyp_keys = list(graph_hyp_range.keys())
    ev_cols = ["LP MAP", "LP P@100"]
    for meth , graph in itertools.product(*[params["methods"],params["graphs"]]):
        hyp_df = pd.DataFrame(
                columns=graph_hyp_keys + ev_cols + ["Round Id"]
            )
        hyp_r_idx = 0
        for hyp_key in graph_hyp_keys:

            for curr_hyp_key_range, r_id in itertools.product(
                *[graph_hyp_range[hyp_key], range(params["rounds"])]
            ):


                ##### first round to find the best parameter for each methods
                if r_id == 0:
                    f_hyp = 1
                else:
                    f_hyp = 0



                curr_hyps = def_graph_hyps.copy()

                curr_hyps[hyp_key] = curr_hyp_key_range
                curr_hyps["domain"] = params["domain_name"]
                hyp_str = '_'.join(
                            "%s=%s" % (key, str(val).strip("'")) for (key, val) in␣
↪curr_hyps.items()
                        )

                hyp_str_graph_name = '_'.join(
                            "%s=%s" % (key, str(val).strip("'")) for (key, val) in␣
↪curr_hyps.items() if key != 'dim'
                        )

                syn_data_folder = 'benchmark_%s_%s_%s' % (graph, hyp_str_graph_name,␣
↪r_id)

                graphClass = getattr(graph_gens, graph)

                try:
                    nx.read_gpickle(
                            'gemben/data/%s/graph.gpickle' % syn_data_folder
                  )
                except:
                    flag =  1
                    ##### flag = 0 means the labels are continous on lcc
                    while flag:
                        print("Graph is generating...")
                        G = graphClass(**curr_hyps)[0]
                        if len(set(G.nodes())) == G.number_of_nodes() and list(G.
↪nodes())[-1] == G.number_of_nodes() -1:
                            flag = 0
                    if G:
                        if not os.path.exists("gemben/data/%s" % syn_data_folder):
                            os.makedirs("gemben/data/%s" % syn_data_folder)
```

(continues on next page)

(continued from previous page)

```
                        nx.write_gpickle(
                                G, 'gemben/data/%s/graph.gpickle' % syn_data_folder
                    )
                perf_exp = not params["lexp"]
                if params["lexp"]:
                  try:
                    MAP, prec, n_samps = pickle.load(
                      open('gemben/results/%s_%s_%d_%s.lp' % (
                          syn_data_folder, meth,
                          curr_hyps["dim"], samp_scheme), 'rb'))
                  except:
                      perf_exp = 1
                       ##### only find the best hyp for first round
                if perf_exp:
                    os.system(
                      "python3 gemben/experiments/exp.py -data %s -meth %s -dim %d -
↪rounds 1 -find_hyp %d -s_sch %s -exp lp" % (
                          syn_data_folder,
                          meth,
                          curr_hyps["dim"],
                          f_hyp,
                          samp_scheme
                      )
                    )
                MAP, prec, n_samps = pickle.load(
                open('gemben/results/%s_%s_%d_%s.lp' % (
                    syn_data_folder, meth,
                    curr_hyps["dim"], samp_scheme), 'rb'))
                hyp_df.loc[hyp_r_idx, graph_hyp_keys] = \
                pd.Series(curr_hyps)
                #prec_100 = prec[int(n_samps[0])][0][100]
                try:
                  prec_100 = list(prec.values())[0][0][100]
                except:
                  pdb.set_trace()
                f_temp = open("gemben/temp/%s_%s_%s_lp_%s_data_hyp.txt" % (
                  params["domain_name"], graph, meth, samp_scheme), 'a')
                f_temp.write('%s: round: %d, MAP: %f, prec_100: %f' % (hyp_str, r_id,␣
↪list(MAP.values())[0][0], prec_100))
                f_temp.close()
                hyp_df.loc[hyp_r_idx, ev_cols + ["Round Id"]] = \
                [list(MAP.values())[0][0], prec_100, r_id]
                #[MAP[int(n_samps[0])][0], prec_100, r_id]
                hyp_r_idx += 1

        hyp_df.to_hdf(
            "gemben/intermediate/%s_%s_%s_lp_%s_data_hyp.h5" % (
                params["domain_name"], graph, meth, samp_scheme),
            "df"
        )
        print('Experiments done for %s, %s' % (graph, meth))
```

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

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

15.4 Plotting Results

Code example to plot the results after the experiment.

Out:

```
social_watts_strogatz_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
social_hyperbolic_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
social_hyperbolic_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
social_stochastic_kronecker_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
social_hyperbolic_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_block_model_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_block_model_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_block_model_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
biology_duplication_divergence_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_block_model_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_block_model_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_block_model_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_block_model_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_block_model_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
biology_stochastic_block_model_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_watts_strogatz_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
internet_stochastic_kronecker_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_watts_strogatz_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_block_model_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
social_watts_strogatz_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
social_hyperbolic_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
social_powerlaw_cluster_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
social_hyperbolic_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_hyperbolic_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
social_watts_strogatz_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_barabasi_albert_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_powerlaw_cluster_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_random_geometric_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_duplication_divergence_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
social_hyperbolic_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_r_mat_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_waxman_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
social_stochastic_kronecker_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
biology_random_geometric_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
biology_random_geometric_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_hyperbolic_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_watts_strogatz_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_barabasi_albert_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_powerlaw_cluster_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_random_geometric_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_duplication_divergence_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
biology_hyperbolic_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_r_mat_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_waxman_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
biology_stochastic_kronecker_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_gf_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_rand_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_watts_strogatz_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_pa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_lap_lp_rw_data_hyp.h5 not found. Ignoring data set
```

(continues on next page)

(continued from previous page)

```
internet_watts_strogatz_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_hope_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_watts_strogatz_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_cn_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_aa_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_watts_strogatz_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_barabasi_albert_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_powerlaw_cluster_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_random_geometric_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_duplication_divergence_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
internet_hyperbolic_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_r_mat_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_waxman_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data set
internet_stochastic_kronecker_graph_sdne_lp_rw_data_hyp.h5 not found. Ignoring data␣
↪set
```

```
try: import cPickle as pickle
except: import pickle

import matplotlib
import matplotlib.pyplot as plt
import itertools
from matplotlib import rc
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import pandas as pd
import seaborn

font = {'family': 'serif', 'serif': ['computer modern roman']}
rc('text', usetex=False)
rc('font', weight='bold')
rc('font', size=8)
rc('lines', markersize=2.5)
rc('lines', linewidth=0.5)
rc('xtick', labelsize=6)
rc('ytick', labelsize=6)
rc('axes', labelsize='small')
rc('axes', labelweight='bold')
rc('axes', titlesize='small')
rc('axes', linewidth=1)
plt.rc('font', **font)
seaborn.set_style("darkgrid")
import pdb


def plot_benchmark(domains, graph_attrs, attr_defaults, methods, graph_names, s_sch=
↪'rw'):
    df_all = pd.DataFrame()
    for d, m , g in itertools.product(*[domains, methods, graph_names]):
        try:
            df = pd.read_hdf(path+
                "%s_%s_%s_lp_%s_data_hyp.h5" % (d, g, m, s_sch),
                "df"
            )
        except:
            print('%s_%s_%s_lp_%s_data_hyp.h5 not found. Ignoring data set' % (d, g,␣
↪m, s_sch))
            continue
        df["Domain"], df["Method"], df["Graph"] = d, m, g
        df_all = df_all.append(df).reset_index()
        df_all = df_all.drop(['index'], axis=1)
    if df_all.empty:
        return
    df_all = df_all.drop(['dia'], axis=1)
    plot_shape = (len(domains), len(graph_attrs))
    fin1, axarray1 = plt.subplots(len(domains), len(graph_attrs), figsize=(7, 4),␣
↪sharex='col', sharey='row')
    data_idx = 0
    gfs_score = {}
    for dom in domains:
        gfs_score[dom] = {m: 0 for m in methods}
        n_attr = 0
        for attr in graph_attrs:
            plot_idx = np.unravel_index(data_idx, plot_shape)
            data_idx += 1
            try:
                rem_attrs = list(set(graph_attrs) - {attr})
                df_grouped = df_all[df_all["Domain"]==dom]
                for rem_attr in rem_attrs:
                    df_grouped = df_grouped[df_grouped[rem_attr]==attr_defaults[rem_
↪attr]]
                df_grouped = df_grouped[[attr, "Round Id", "LP MAP", "Method", "Graph
↪"]]
```

(continues on next page)

(continued from previous page)

```
                print(df_grouped.head())
                df_grouped['LP MAP'] = df_grouped['LP MAP'].astype('float')
                df_grouped = df_grouped.groupby([attr, "Round Id", "Method", "Graph
↪"]).mean().reset_index()
            except TypeError:
                df_trun[hyp_key_ren + "2"] = \
                    df_trun[hyp_key_ren].apply(lambda x: str(x))
                df_trun[hyp_key_ren] = df_trun[hyp_key_ren + "2"].copy()
                df_trun = df_trun.drop([hyp_key_ren + "2"], axis=1)
                df_grouped = df_trun.groupby([hyp_key_ren, "Round Id", "Data"]).max().
↪reset_index()

            df_grouped['unit']=df_grouped.apply(lambda x:'%s_%s' % (x['Round Id'],x[
↪'Graph']),axis=1)
            df_grouped = df_grouped.drop(['Round Id', "Graph"], axis=1)
            m_scores = dict(df_grouped.groupby(["Method"])["LP MAP"].mean())
            n_attr += 1
            gfs_score[dom] = {m: m_scores[m] + gfs_score[dom][m] for m in methods}
            ax = seaborn.tsplot(time=attr, value="LP MAP",
                                unit="unit", condition="Method", legend=False,
                                data=df_grouped, ax=axarray1[plot_idx[0], plot_
↪idx[1]])
            if not plot_idx[1]:
                ax.set_ylabel(dom)
            if plot_idx[1]:
                ax.set_ylabel('')
            if plot_idx[0] < len(domains) - 1:
                ax.set_xlabel('')
            attr_values = df_grouped[attr].unique()
            l_diff = attr_values[-1] - attr_values[-2]
            f_diff = attr_values[1] - attr_values[0]
            l_f_diff_r = l_diff / f_diff
            if l_f_diff_r > 1:
                log_base = pow(l_f_diff_r, 1.0 / (len(attr_values) - 2))
                ax.set_xscale('log', basex=round(log_base))
            marker = ["o", "s", "D", "^", "v", "8", "*", "p", "1", "h"]
            for line_i in range(len(ax.lines)):
                ax.lines[line_i].set_marker(marker[line_i])
        gfs_score[dom] = {m: gfs_score[dom][m]/n_attr for m in methods}
    print(gfs_score)
    plt.savefig(
       'benchmark.pdf',
       dpi=300, format='pdf', bbox_inches='tight'
    )
    plt.show()

def plot_benchmark_individual(domains, graph_attrs, attr_defaults, methods, graph_
↪names, graph_short, s_sch='rw'):
    df_all = pd.DataFrame()
    for d, m , g in itertools.product(*[domains, methods, graph_names]):
        try:
            df = pd.read_hdf(path+
                "%s_%s_%s_lp_%s_data_hyp.h5" % (d, g, m, s_sch),
                "df"
            )
        except:
            print('%s_%s_%s_lp_%s_data_hyp.h5 not found. Ignoring data set' % (d, g,␣
↪m, s_sch))
```

(continues on next page)

(continued from previous page)

```
            continue
        df["Method"], df["Graph"] = m, g
        df_all = df_all.append(df).reset_index()
        df_all = df_all.drop(['index'], axis=1)
    if df_all.empty:
        return
    df_all = df_all.drop(['dia'], axis=1)
    plot_shape = (len(graph_names), len(graph_attrs))
    fin1, axarray1 = plt.subplots(len(graph_names), len(graph_attrs), figsize=(7, 4),␣
↪sharex='col', sharey='row')
    data_idx = 0
    for graph in graph_names:
        for attr in graph_attrs:
            plot_idx = np.unravel_index(data_idx, plot_shape)
            data_idx += 1
            try:
                rem_attrs = list(set(graph_attrs) - {attr})
                df_grouped = df_all[df_all["Graph"]==graph]
                for rem_attr in rem_attrs:
                    df_grouped = df_grouped[df_grouped[rem_attr]==attr_defaults[rem_
↪attr]]
                df_grouped = df_grouped[[attr, "Round Id", "LP MAP", "Method"]]
                print(df_grouped.head())
                df_grouped['LP MAP'] = df_grouped['LP MAP'].astype('float')
                df_grouped = df_grouped.groupby([attr, "Round Id", "Method"]).mean().
↪reset_index()
            except TypeError:
                df_trun[hyp_key_ren + "2"] = \
                    df_trun[hyp_key_ren].apply(lambda x: str(x))
                df_trun[hyp_key_ren] = df_trun[hyp_key_ren + "2"].copy()
                df_trun = df_trun.drop([hyp_key_ren + "2"], axis=1)
                df_grouped = df_trun.groupby([hyp_key_ren, "Round Id", "Data"]).max().
↪reset_index()

            if data_idx == len(graph_names) * len(graph_attrs):
                legend = True
            else:
                legend = False
            ax = seaborn.tsplot(time=attr, value="LP MAP",
                                unit="Round Id", condition="Method",␣
↪legend=legend,
                                data=df_grouped, ax=axarray1[plot_idx[0], plot_
↪idx[1]])
            if legend:
                ax.legend_.remove()
            if not plot_idx[1]:
                ax.set_ylabel(graph_short[graph])
            if plot_idx[1]:
                ax.set_ylabel('')
            if plot_idx[0] < len(graph_names) - 1:
                ax.set_xlabel('')
            attr_values = df_grouped[attr].unique()
            l_diff = attr_values[-1] - attr_values[-2]
            f_diff = attr_values[1] - attr_values[0]
            l_f_diff_r = l_diff / f_diff
            if l_f_diff_r > 1:
                log_base = pow(l_f_diff_r, 1.0 / (len(attr_values) - 2))
```

(continues on next page)

(continued from previous page)

```
                ax.set_xscale('log', basex=round(log_base))
            marker = ["o", "s", "D", "^", "v", "8", "*", "p", "1", "h"]
            for line_i in range(len(ax.lines)):
                ax.lines[line_i].set_marker(marker[line_i])
    for col_idx in range(axarray1[3].shape[0]):
        box = axarray1[3][col_idx].get_position()
        axarray1[3][col_idx].set_position(
            [box.x0,
             box.y0 + box.height * 0.1,
             box.width,
             box.height * 0.9]
        )
    fin1.legend(loc='lower center', bbox_to_anchor=(0.45, -0.01),
                ncol=len(methods), fancybox=True, shadow=True)
    fin1.savefig(
       'benchmark_individual3.pdf',
       dpi=300, format='pdf', bbox_inches='tight'
    )
    plt.show()


#Path for the stored .h5 result
path = '/Users/Bench_files/new_files/'



plot_benchmark(
    ["social", "biology", "internet"],
    ["N", "dim", "deg"],
    {"N": 4096, "deg": 8, "dia": None, "dim": 128},
    ["gf", "rand", "pa", "lap", "hope", "cn", "aa","sdne"],
    ["watts_strogatz_graph", "barabasi_albert_graph", "powerlaw_cluster_graph",
↪"random_geometric_graph",\
     "duplication_divergence_graph","hyperbolic_graph","r_mat_graph","waxman_graph",
↪"stochastic_block_model",\
     "stochastic_kronecker_graph"],
    s_sch='rw'
)

plot_benchmark_individual(
    ["social", "biology", "internet"],
    ["N", "dim", "deg"],
    {"N": 4096, "deg": 8, "dia": None, "dim": 128},
    ["gf", "rand", "pa", "lap", "hope", "cn", "aa", "sdne"],
    ["watts_strogatz_graph", "barabasi_albert_graph", "powerlaw_cluster_graph",
↪"random_geometric_graph",\
     "duplication_divergence_graph","hyperbolic_graph","r_mat_graph","waxman_graph",
↪"stochastic_kronecker_graph"],
    {"watts_strogatz_graph": "WS", "barabasi_albert_graph": "BA", \
     "powerlaw_cluster_graph": "PC", "random_geometric_graph": "RG",\
     "duplication_divergence_graph": "DD", "hyperbolic_graph": "HB",\
     "r_mat_graph": "RM", "waxman_graph":"WM","stochastic_block_model":"SBM",
↪"stochastic_kronecker_graph": "KG"
    },
    s_sch='rw'
)
```

Total running time of the script: (0 minutes 1.366 seconds)

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

15.5 Simple Experiment

A simple experiment to run the gemben library.

```
try: import cPickle as pickle
except: import pickle
from time import time
from argparse import ArgumentParser
import importlib
import json
# import cPickle
import networkx as nx
import itertools
import pdb
import sys
import numpy as np
import pandas as pd
# sys.path.insert(0, './')
import os
from gemben.utils      import graph_util, plot_util
from gemben.evaluation import visualize_embedding as viz
from gemben.evaluation.evaluate_graph_reconstruction import expGR
from gemben.evaluation.evaluate_link_prediction import expLP, expLPT
from gemben.evaluation.evaluate_node_classification import expNC
from gemben.evaluation.visualize_embedding import expVis

methClassMap = {"gf": "GraphFactorization",
                "hope": "HOPE",
                "lap": "LaplacianEigenmaps",
                "node2vec": "node2vec",
                "sdne": "SDNE",
                "pa": "PreferentialAttachment",
                "rand": "RandomEmb",
                "cn": "CommonNeighbors",
                "aa": "AdamicAdar",
                "jc": "JaccardCoefficient"}
expMap = {"gf": "GF MAP", "lp": "LP MAP",
          "nc": "NC MAP"}


def learn_emb(MethObj, di_graph, params, res_pre, m_summ):
    if params["experiments"] == ["lp"]:
        X = None
    else:
        print('Learning Embedding: %s' % m_summ)
        if not bool(int(params["load_emb"])):
            X, learn_t = MethObj.learn_embedding(graph=di_graph,
                                                 edge_f=None,
                                                 no_python=True)
            print('\tTime to learn embedding: %f sec' % learn_t)
```

(continues on next page)

(continued from previous page)

```
            pickle.dump(X, open('%s_%s.emb' % (res_pre, m_summ), 'wb'))
            pickle.dump(learn_t,
                        open('%s_%s.learnT' % (res_pre, m_summ), 'wb'))
        else:
            X = pickle.load(open('%s_%s.emb' % (res_pre, m_summ),
                                 'rb'))
            try:
                learn_t = pickle.load(open('%s_%s.learnT' % (res_pre, m_summ),
                                           'rb'))
                print('\tTime to learn emb.: %f sec' % learn_t)
            except IOError:
                print('\tTime info not found')
    return X


def run_exps(MethObj, meth, dim, di_graph, data_set, node_labels, params):
    m_summ = '%s_%d' % (meth, dim)
    res_pre = "gemben/results/%s" % data_set
    n_r = params["rounds"]
    X = learn_emb(MethObj, di_graph, params, res_pre, m_summ)
    gr, lp, nc = [0] * n_r, [0] * n_r, [0] * n_r
    if "gr" in params["experiments"]:
        gr = expGR(di_graph, MethObj,
                   X, params["n_sample_nodes"].split(","),
                   n_r, res_pre,
                   m_summ, is_undirected=params["is_undirected"],
                   sampling_scheme=params["samp_scheme"])
    if "lpt" in params["experiments"]:
        expLPT(di_graph, MethObj, res_pre, m_summ,
               is_undirected=params["is_undirected"])
    if "lp" in params["experiments"]:
        lp = expLP(di_graph, MethObj,
                   params["n_sample_nodes"].split(","),
                   n_r, res_pre,
                   m_summ, train_ratio=params["train_ratio_lp"],
                   is_undirected=params["is_undirected"],
                   sampling_scheme=params["samp_scheme"])
    if "nc" in params["experiments"]:
        if "nc_test_ratio_arr" not in params:
            print('NC test ratio not provided')
        else:
            nc = expNC(X, node_labels, params["nc_test_ratio_arr"],
                       n_r, res_pre,
                       m_summ)
    if "viz" in params["experiments"]:
        if MethObj.get_method_name() == 'hope_gsvd':
            d = X.shape[1] / 2
            expVis(X[:, :d], res_pre, m_summ,
                   node_labels=node_labels, di_graph=di_graph)
        else:
            expVis(X, res_pre, m_summ,
                   node_labels=node_labels, di_graph=di_graph)
    return gr, lp, nc


def get_max(val, val_max, idx, idx_max):
    if val > val_max:
```

(continues on next page)

(continued from previous page)

```
        return val, idx
    else:
        return val_max, idx_max


def choose_best_hyp(data_set, di_graph, node_labels, params):
    # Load range of hyper parameters to test on
    try:
        model_hyp_range = json.load(
            open('gemben/experiments/config/%s_hypRange.conf' % data_set, 'r')
        )
    except IOError:
        model_hyp_range = json.load(
            open('gemben/experiments/config/default_hypRange.conf', 'r')
        )
    try:
        os.makedirs("gemben/temp_hyp_res")
    except:
        pass
    # Test each hyperparameter for each method and store the best
    for meth in params["methods"]:
        dim = int(params["dimensions"][0])
        MethClass = getattr(
            importlib.import_module("gemben.embedding.%s" % meth),
            methClassMap[meth]
        )
        meth_hyp_range = model_hyp_range[meth]
        gr_max, lp_max, nc_max = 0, 0, 0
        gr_hyp, lp_hyp, nc_hyp = 0, 0, 0
        gr_hyp, lp_hyp, nc_hyp = {meth: {}}, {meth: {}}, {meth: {}}

        # Test each hyperparameter
        ev_cols = ["GR MAP", "LP MAP", "NC F1 score"]
        hyp_df = pd.DataFrame(
            columns=list(meth_hyp_range.keys()) + ev_cols + ["Round Id"]
        )
        hyp_r_idx = 0
        for hyp in itertools.product(*meth_hyp_range.values()):
            hyp_d = {"d": dim}
            hyp_d.update(dict(zip(meth_hyp_range.keys(), hyp)))
            print(hyp_d)
            if meth == "sdne":
                hyp_d.update({
                    "modelfile": [
                        "gemben/intermediate/enc_mdl_%s_%d.json" % (data_set, dim),
                        "gemben/intermediate/dec_mdl_%s_%d.json" % (data_set, dim)
                    ],
                    "weightfile": [
                        "gemben/intermediate/enc_wts_%s_%d.hdf5" % (data_set, dim),
                        "gemben/intermediate/dec_wts_%s_%d.hdf5" % (data_set, dim)
                    ]
                })
            elif meth == "gf" or meth == "node2vec":
                hyp_d.update({"data_set": data_set})
            MethObj = MethClass(hyp_d)
            gr, lp, nc = run_exps(MethObj, meth, dim, di_graph,
                                  data_set, node_labels, params)
```

(continues on next page)

(continued from previous page)

```
            gr_m, lp_m, nc_m = np.mean(gr), np.mean(lp), np.mean(nc)
            gr_max, gr_hyp[meth] = get_max(gr_m, gr_max, hyp_d, gr_hyp[meth])
            lp_max, lp_hyp[meth] = get_max(lp_m, lp_max, hyp_d, lp_hyp[meth])
            nc_max, nc_hyp[meth] = get_max(nc_m, nc_max, hyp_d, nc_hyp[meth])
            hyp_df_row = dict(zip(meth_hyp_range.keys(), hyp))
            f_hyp_temp = open("gemben/temp_hyp_res/%s_%s.txt" % (data_set, meth), "a")
            hyp_str = '_'.join("%s=%s" % (key, str(val).strip("'")) for (key, val) in␣
↪hyp_d.items())
            f_hyp_temp.write('%s: MAP: %f\n' % (hyp_str, lp_max))
            f_hyp_temp.close()
            for r_id in range(params["rounds"]):
                hyp_df.loc[hyp_r_idx, meth_hyp_range.keys()] = \
                    pd.Series(hyp_df_row)
                hyp_df.loc[hyp_r_idx, ev_cols + ["Round Id"]] = \
                    [gr[min(r_id, len(gr) -1)], lp[r_id], nc[r_id], r_id]
                hyp_r_idx += 1
        exp_param = params["experiments"]
        for exp in exp_param:
            hyp_df.to_hdf(
                "gemben/intermediate/%s_%s_%s_%s_hyp.h5" % (data_set, meth,
                                                            exp,
                                                            params["samp_scheme"]),
                "df"
            )
        ###plot_util.plot_hyp(meth_hyp_range.keys(), exp_param,
           ##:                meth, data_set, s_sch=params["samp_scheme"])

        # Store the best hyperparameter
        ####### put the file into synthetic
        opt_hyp_f_pre = 'gemben/experiments/config/synthetic/%s_%s_%s' % (
            data_set,
            meth,
            params["samp_scheme"]
        )
        if gr_max:
            with open('%s_gr.conf' % opt_hyp_f_pre, 'w') as f:
                f.write(json.dumps(gr_hyp, indent=4))
        if lp_max:
            with open('%s_lp.conf' % opt_hyp_f_pre, 'w') as f:
                f.write(json.dumps(lp_hyp, indent=4))
        if nc_max:
            with open('%s_nc.conf' % opt_hyp_f_pre, 'w') as f:
                f.write(json.dumps(nc_hyp, indent=4))


def call_plot_hyp(data_set, params):
    # Load range of hyper parameters tested on to plot
    try:
        model_hyp_range = json.load(
            open('gemben/experiments/config/%s_hypRange.conf' % data_set, 'r')
        )
    except IOError:
        model_hyp_range = json.load(
            open('gemben/experiments/config/default_hypRange.conf', 'r')
        )
    for meth in params["methods"]:
            meth_hyp_range = model_hyp_range[meth]
```

(continues on next page)

(continued from previous page)

```
            exp_param = params["experiments"]
            plot_util.plot_hyp(meth_hyp_range.keys(), exp_param,
                               meth, data_set,
                               s_sch=params["samp_scheme"])


def call_plot_hyp_all(data_sets, params):
    # Load range of hyper parameters tested on to plot
    try:
        model_hyp_range = json.load(
            open('gemben/experiments/config/%s_hypRange.conf' % data_sets[0], 'r')
        )
    except IOError:
        model_hyp_range = json.load(
            open('gemben/experiments/config/default_hypRange.conf', 'r')
        )
    for meth in params["methods"]:
            meth_hyp_range = model_hyp_range[meth]
            exp_param = params["experiments"]
            plot_util.plot_hyp_all(meth_hyp_range.keys(), exp_param,
                                   meth, data_sets,
                                   s_sch=params["samp_scheme"])


def call_exps(params, data_set):
    # Load Dataset
    print('Dataset: %s' % data_set)



########  for SBM, r_mat, hyperbolic
    #if data_set[10:13] == 'r_m' or data_set[10:13] == 'sto' or data_set[10:13] ==
↪'hyp':
     #   di_graph = nx.read_gpickle('gem/data/%s/graph.gpickle' % data_set)[0]
    #else:

    #di_graph = nx.read_gpickle('gem/data/%s/graph.gpickle' % data_set)[0]
    di_graph = nx.read_gpickle('gemben/data/%s/graph.gpickle' % data_set)

    di_graph, nodeListMap = graph_util.get_lcc(di_graph)
    try:
      os.makedirs('gemben/nodeListMap')
    except:
      pass
    pickle.dump(nodeListMap, open('gemben/nodeListMap/%s.pickle' % data_set, 'wb'))
    graph_util.print_graph_stats(di_graph)


    # Load node labels if given
    if bool(params["node_labels"]):
        node_labels = cPickle.load(
            open('gemben/data/%s/node_labels.pickle' % data_set, 'rb')
        )
        node_labels_gc = np.zeros(
            (di_graph.number_of_nodes(), node_labels.shape[1]))
        for k, v in nodeListMap.iteritems():
```

(continues on next page)

(continued from previous page)

```
            try:
                node_labels_gc[v, :] = node_labels[k, :].toarray()
            # Already a numpy array
            except AttributeError:
                node_labels_gc[v, :] = node_labels[k, :]
        node_labels = node_labels_gc
    else:
        node_labels = None

    # Search through the hyperparameter space
    if params["find_hyp"]:
        choose_best_hyp(data_set, di_graph, node_labels, params)

    # Load best hyperparameter and test it again on new test data
    for d, meth, exp in itertools.product(
        params["dimensions"],
        params["methods"],
        params["experiments"]
    ):
        dim = int(d)
        MethClass = getattr(
            importlib.import_module("gemben.embedding.%s" % meth),
            methClassMap[meth]
        )
        opt_hyp_f_pre = 'gemben/experiments/config/synthetic/%s_%s_%s' % (
            data_set,
            meth,
            params["samp_scheme"]
        )
        try:
            if exp != "viz":
                if exp == 'lpt':
                    model_hyp = json.load(
                        open('%s_lp.conf' % opt_hyp_f_pre, 'r')
                    )
                else:
                    model_hyp = json.load(
                        open('%s_%s.conf' % (opt_hyp_f_pre, exp), 'r')
                    )
            else:
                model_hyp = json.load(
                    open(
                        '%s_%s.conf' % (opt_hyp_f_pre, params["viz_params"]), 'r'
                    )
                )
        except IOError:
            print('Default hyperparameter of the method chosen')
            model_hyp = json.load(
                open('gemben/experiments/config/%s.conf' % meth, 'r')
            )
        hyp = {}
        hyp.update(model_hyp[meth])
        hyp.update({"d": dim})
        if meth == "sdne":
                hyp.update({
                    "modelfile": [
                        "gemben/intermediate/en_mdl_%s_%d.json" % (data_set, dim),
```

(continues on next page)

(continued from previous page)

```
                        "gemben/intermediate/dec_mdl_%s_%d.json" % (data_set, dim)
                    ],
                    "weightfile": [
                        "gemben/intermediate/enc_wts_%s_%d.hdf5" % (data_set, dim),
                        "gemben/intermediate/dec_wts_%s_%d.hdf5" % (data_set, dim)
                    ]
                })
        elif meth == "gf" or meth == "node2vec":
            hyp.update({"data_set": data_set})
        MethObj = MethClass(hyp)
        run_exps(MethObj, meth, dim, di_graph, data_set, node_labels, params)


if __name__ == '__main__':
    ''' Sample usage
    python experiments/exp.py -data sbm -dim 128 -meth sdne -exp gr,lp
    '''
    t1 = time()
    parser = ArgumentParser(description='Graph Embedding Experiments')
    parser.add_argument('-data', '--data_sets',
                        help='dataset names (default: sbm)')
    parser.add_argument('-dim', '--dimensions',
                        help='embedding dimensions list(default: 2^1 to 2^8)')
    parser.add_argument('-meth', '--methods',
                        help='method list (default: all methods)')
    parser.add_argument('-exp', '--experiments',
                        help='exp list (default: gr,lp,viz,nc)')
    parser.add_argument('-lemb', '--load_emb',
                        help='load saved embeddings (default: False)')
    parser.add_argument('-lexp', '--load_exp',
                        help='load saved experiment results (default: False)')
    parser.add_argument('-node_labels', '--node_labels',
                        help='node labels available or not (default: False)')
    parser.add_argument('-rounds', '--rounds',
                        help='number of rounds (default: 5)')
    parser.add_argument('-plot', '--plot',
                        help='plot the results (default: True)')
    parser.add_argument('-plot_d', '--plot_d',
                        help='plot the results wrt dims(default: True)')
    parser.add_argument('-hyp_plot', '--hyp_plot',
                        help='plot the hyperparameter results (default: True)')
    parser.add_argument('-hyp_plot_all', '--hyp_plot_all',
                        help='plot the hyperparameter results (all) (default: True)')
    parser.add_argument('-train_ratio_lp', '--train_ratio_lp',
                        help='fraction of data used for training(default: 0.8)')
    parser.add_argument('-viz_params', '--viz_params',
                        help='which params to use for viz (default: gr)')
    parser.add_argument('-find_hyp', '--find_hyp',
                        help='find best hyperparameters (default: False)')
    parser.add_argument('-saveMAP', '--save_MAP',
                        help='save MAP in a latex table (default: False)')
    parser.add_argument('-n_samples', '--n_sample_nodes',
                        help='number of sampled nodes (default: 1024)')
    parser.add_argument('-s_sch', '--samp_scheme',
                        help='sampling scheme (default: u_rand)')

    params = json.load(open('gem/experiments/config/params.conf', 'r'))
```

(continues on next page)

(continued from previous page)

```
    args = vars(parser.parse_args())

    for k, v in args.items():
        if v is not None:
            params[k] = v

    params["experiments"] = params["experiments"].split(',')
    params["data_sets"] = params["data_sets"].split(',')
    params["experiments"] = list(set(params["experiments"]))
    params["data_sets"] = list(set(params["data_sets"]))
    params["rounds"] = int(params["rounds"])
    params["node_labels"] = int(params["node_labels"])
    params["train_ratio_lp"] = float(params["train_ratio_lp"])
    # params["n_sample_nodes"] = int(params["n_sample_nodes"])
    params["is_undirected"] = bool(int(params["is_undirected"]))



    params["plot_d"] = bool(int(params["plot_d"]))
    params["plot"] = bool(int(params["plot"]))
    params["hyp_plot"] = bool(int(params["hyp_plot"]))
    params["hyp_plot_all"] = bool(int(params["hyp_plot_all"]))
    params["find_hyp"] = bool(int(params["find_hyp"]))

    if params["methods"] == "all":
        params["methods"] = methClassMap.keys()
    else:
        params["methods"] = params["methods"].split(',')
    params["methods"] = list(set(params["methods"]))

    params["dimensions"] = params["dimensions"].split(',')
    params["dimensions"] = list(set(params["dimensions"]))
    if "nc_test_ratio_arr" in params:
        params["nc_test_ratio_arr"] = params["nc_test_ratio_arr"].split(',')
        params["nc_test_ratio_arr"] = \
            [float(ratio) for ratio in params["nc_test_ratio_arr"]]
    try:
      os.makedirs("gemben/intermediate")
    except:
      pass
    try:
      os.makedirs("gemben/results")
    except:
      pass

    for data_set in params["data_sets"]:
        if not int(params["load_exp"]):
            call_exps(params, data_set)
        if int(params["plot"]):
            res_pre = "gemben/results/%s" % data_set
            plot_util.plotExpRes(res_pre, params["methods"],
                                 params["experiments"], params["dimensions"],
                                 'gemben/plots/%s_%s' % (data_set, params["samp_scheme
↪"]),
                                 params["rounds"], params["plot_d"],
                                 params["train_ratio_lp"],
```

(continues on next page)

(continued from previous page)

```
                                    params["samp_scheme"])
        if int(params["hyp_plot"]):
            call_plot_hyp(data_set, params)
    if int(params["hyp_plot_all"]):
            call_plot_hyp_all(params["data_sets"], params)
```

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

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

15.6 Bayesian Hyerparameter tuning

Experiment to utilize the bayesian hyperparameter tuning for the algorithms.

```
try: import cPickle as pickle
except: import pickle
from time import time
from argparse import ArgumentParser
import importlib
import json
# import cPickle
import networkx as nx
import itertools
import pdb
import sys
import numpy as np
import pandas as pd
# sys.path.insert(0, './')
import os
from gemben.utils      import graph_util, plot_util
from gemben.evaluation import visualize_embedding as viz
from gemben.evaluation.evaluate_graph_reconstruction import expGR
from gemben.evaluation.evaluate_link_prediction import expLP, expLPT
from gemben.evaluation.evaluate_node_classification import expNC
from gemben.evaluation.visualize_embedding import expVis

from gemben.utils.bayesian_opt import BayesianOpt


methClassMap = {"gf": "GraphFactorization",
                "hope": "HOPE",
                "lap": "LaplacianEigenmaps",
                "node2vec": "node2vec",
                "sdne": "SDNE",
                "pa": "PreferentialAttachment",
                "rand": "RandomEmb",
                "cn": "CommonNeighbors",
                "aa": "AdamicAdar",
                "jc": "JaccardCoefficient"}
expMap = {"gf": "GF MAP", "lp": "LP MAP",
          "nc": "NC MAP"}
```

(continues on next page)

(continued from previous page)

```
def learn_emb(MethObj, di_graph, params, res_pre, m_summ):
    if params["experiments"] == ["lp"]:
        X = None
    else:
        print('Learning Embedding: %s' % m_summ)
        if not bool(int(params["load_emb"])):
            X, learn_t = MethObj.learn_embedding(graph=di_graph,
                                                 edge_f=None,
                                                 no_python=True)
            print('\tTime to learn embedding: %f sec' % learn_t)
            pickle.dump(X, open('%s_%s.emb' % (res_pre, m_summ), 'wb'))
            pickle.dump(learn_t,
                        open('%s_%s.learnT' % (res_pre, m_summ), 'wb'))
        else:
            X = pickle.load(open('%s_%s.emb' % (res_pre, m_summ),
                                 'rb'))
            try:
                learn_t = pickle.load(open('%s_%s.learnT' % (res_pre, m_summ),
                                           'rb'))
                print('\tTime to learn emb.: %f sec' % learn_t)
            except IOError:
                print('\tTime info not found')
    return X


def run_exps(MethObj, meth, dim, di_graph, data_set, node_labels, params):
    m_summ = '%s_%d' % (meth, dim)
    res_pre = "gemben/results/%s" % data_set
    n_r = params["rounds"]
    X = learn_emb(MethObj, di_graph, params, res_pre, m_summ)
    gr, lp, nc = [0] * n_r, [0] * n_r, [0] * n_r
    if "gr" in params["experiments"]:
        gr = expGR(di_graph, MethObj,
                   X, params["n_sample_nodes"].split(","),
                   n_r, res_pre,
                   m_summ, is_undirected=params["is_undirected"],
                   sampling_scheme=params["samp_scheme"])
    if "lpt" in params["experiments"]:
        expLPT(di_graph, MethObj, res_pre, m_summ,
               is_undirected=params["is_undirected"])
    if "lp" in params["experiments"]:
        lp = expLP(di_graph, MethObj,
                   params["n_sample_nodes"].split(","),
                   n_r, res_pre,
                   m_summ, is_undirected=params["is_undirected"],
                   sampling_scheme=params["samp_scheme"])
    if "nc" in params["experiments"]:
        if "nc_test_ratio_arr" not in params:
            print('NC test ratio not provided')
        else:
            nc = expNC(X, node_labels, params["nc_test_ratio_arr"],
                       n_r, res_pre,
                       m_summ)
    if "viz" in params["experiments"]:
        if MethObj.get_method_name() == 'hope_gsvd':
            d = X.shape[1] / 2
            expVis(X[:, :d], res_pre, m_summ,
```

(continues on next page)

(continued from previous page)

```
                    node_labels=node_labels, di_graph=di_graph)
        else:
            expVis(X, res_pre, m_summ,
                    node_labels=node_labels, di_graph=di_graph)
    return gr, lp, nc


def get_max(val, val_max, idx, idx_max):
    if val > val_max:
        return val, idx
    else:
        return val_max, idx_max


def bayesian_optimization(data_set, di_graph, node_labels, params):
    # Load range of hyper parameters
    try:
        model_hyp_range = json.load(
            open('gemben/experiments/config/%s_hypRange.conf' % data_set, 'r')
        )
    except IOError:
        model_hyp_range = json.load(
            open('gemben/experiments/config/default_hypRange.conf', 'r')
        )

    params['model_hyp_range'] = model_hyp_range
    params['data_set'] = data_set
    params['di_graph'] = di_graph

    ## test for gf
    #pdb.set_trace()
    bayesian_opt = BayesianOpt(**params)
    bayesian_opt.optimize()




def choose_best_hyp(data_set, di_graph, node_labels, params):
    # Load range of hyper parameters to test on
    try:
        model_hyp_range = json.load(
            open('gemben/experiments/config/%s_hypRange.conf' % data_set, 'r')
        )
    except IOError:
        model_hyp_range = json.load(
            open('gemben/experiments/config/default_hypRange.conf', 'r')
        )
    try:
        os.makedirs("gemben/temp_hyp_res")
    except:
        pass
    # Test each hyperparameter for each method and store the best
    for meth in params["methods"]:
        dim = int(params["dimensions"][0])
        MethClass = getattr(
            importlib.import_module("gem.embedding.%s" % meth),
            methClassMap[meth]
```

(continues on next page)

(continued from previous page)

```
        )
        meth_hyp_range = model_hyp_range[meth]
        gr_max, lp_max, nc_max = 0, 0, 0
        gr_hyp, lp_hyp, nc_hyp = 0, 0, 0
        gr_hyp, lp_hyp, nc_hyp = {meth: {}}, {meth: {}}, {meth: {}}

        # Test each hyperparameter
        ev_cols = ["GR MAP", "LP MAP", "NC F1 score"]
        hyp_df = pd.DataFrame(
            columns=list(meth_hyp_range.keys()) + ev_cols + ["Round Id"]
        )
        hyp_r_idx = 0
        for hyp in itertools.product(*meth_hyp_range.values()):
            hyp_d = {"d": dim}
            hyp_d.update(dict(zip(meth_hyp_range.keys(), hyp)))
            print(hyp_d)
            if meth == "sdne":
                hyp_d.update({
                    "modelfile": [
                        "gemben/intermediate/enc_mdl_%s_%d.json" % (data_set, dim),
                        "gemben/intermediate/dec_mdl_%s_%d.json" % (data_set, dim)
                    ],
                    "weightfile": [
                        "gemben/intermediate/enc_wts_%s_%d.hdf5" % (data_set, dim),
                        "gemben/intermediate/dec_wts_%s_%d.hdf5" % (data_set, dim)
                    ]
                })
            elif meth == "gf" or meth == "node2vec":
                hyp_d.update({"data_set": data_set})
            MethObj = MethClass(hyp_d)
            gr, lp, nc = run_exps(MethObj, meth, dim, di_graph,
                                  data_set, node_labels, params)
            gr_m, lp_m, nc_m = np.mean(gr), np.mean(lp), np.mean(nc)
            gr_max, gr_hyp[meth] = get_max(gr_m, gr_max, hyp_d, gr_hyp[meth])
            lp_max, lp_hyp[meth] = get_max(lp_m, lp_max, hyp_d, lp_hyp[meth])
            nc_max, nc_hyp[meth] = get_max(nc_m, nc_max, hyp_d, nc_hyp[meth])
            hyp_df_row = dict(zip(meth_hyp_range.keys(), hyp))
            f_hyp_temp = open("gemben/temp_hyp_res/%s_%s.txt" % (data_set, meth), "a")
            hyp_str = '_'.join("%s=%s" % (key, str(val).strip("'")) for (key, val) in␣
↪hyp_d.items())
            f_hyp_temp.write('%s: MAP: %f\n' % (hyp_str, lp_max))
            f_hyp_temp.close()
            for r_id in range(params["rounds"]):
                hyp_df.loc[hyp_r_idx, meth_hyp_range.keys()] = \
                    pd.Series(hyp_df_row)
                hyp_df.loc[hyp_r_idx, ev_cols + ["Round Id"]] = \
                    [gr[min(r_id, len(gr) -1)], lp[r_id], nc[r_id], r_id]
                hyp_r_idx += 1
        exp_param = params["experiments"]
        for exp in exp_param:
            hyp_df.to_hdf(
                "gemben/intermediate/%s_%s_%s_%s_hyp.h5" % (data_set, meth,
                                                            exp,
                                                            params["samp_scheme"]),
                "df"
            )
        ###plot_util.plot_hyp(meth_hyp_range.keys(), exp_param,
```

(continues on next page)

(continued from previous page)

```
            ##:                 meth, data_set, s_sch=params["samp_scheme"])

        # Store the best hyperparameter
        ####### put the file into synthetic
        opt_hyp_f_pre = 'gemben/experiments/config/synthetic/%s_%s_%s' % (
            data_set,
            meth,
            params["samp_scheme"]
        )
        if gr_max:
            with open('%s_gr.conf' % opt_hyp_f_pre, 'w') as f:
                f.write(json.dumps(gr_hyp, indent=4))
        if lp_max:
            with open('%s_lp.conf' % opt_hyp_f_pre, 'w') as f:
                f.write(json.dumps(lp_hyp, indent=4))
        if nc_max:
            with open('%s_nc.conf' % opt_hyp_f_pre, 'w') as f:
                f.write(json.dumps(nc_hyp, indent=4))


def call_plot_hyp(data_set, params):
    # Load range of hyper parameters tested on to plot
    try:
        model_hyp_range = json.load(
            open('gemben/experiments/config/%s_hypRange.conf' % data_set, 'r')
        )
    except IOError:
        model_hyp_range = json.load(
            open('gemben/experiments/config/default_hypRange.conf', 'r')
        )
    for meth in params["methods"]:
            meth_hyp_range = model_hyp_range[meth]
            exp_param = params["experiments"]
            plot_util.plot_hyp(meth_hyp_range.keys(), exp_param,
                               meth, data_set,
                               s_sch=params["samp_scheme"])


def call_plot_hyp_all(data_sets, params):
    # Load range of hyper parameters tested on to plot
    try:
        model_hyp_range = json.load(
            open('gemben/experiments/config/%s_hypRange.conf' % data_sets[0], 'r')
        )
    except IOError:
        model_hyp_range = json.load(
            open('gemben/experiments/config/default_hypRange.conf', 'r')
        )
    for meth in params["methods"]:
            meth_hyp_range = model_hyp_range[meth]
            exp_param = params["experiments"]
            plot_util.plot_hyp_all(meth_hyp_range.keys(), exp_param,
                                   meth, data_sets,
                                   s_sch=params["samp_scheme"])


def call_exps(params, data_set):
```

(continues on next page)

(continued from previous page)

```
    # Load Dataset
    print('Dataset: %s' % data_set)



########  for SBM, r_mat, hyperbolic
    #if data_set[10:13] == 'r_m' or data_set[10:13] == 'sto' or data_set[10:13] ==
↪'hyp':
     #   di_graph = nx.read_gpickle('gem/data/%s/graph.gpickle' % data_set)[0]
    #else:

    #di_graph = nx.read_gpickle('gem/data/%s/graph.gpickle' % data_set)[0]
    di_graph = nx.read_gpickle('gemben/data/%s/graph.gpickle' % data_set)

    di_graph, nodeListMap = graph_util.get_lcc(di_graph)
    graph_util.print_graph_stats(di_graph)


    # Load node labels if given
    if bool(params["node_labels"]):
        node_labels = cPickle.load(
            open('gemben/data/%s/node_labels.pickle' % data_set, 'rb')
        )
        node_labels_gc = np.zeros(
            (di_graph.number_of_nodes(), node_labels.shape[1]))
        for k, v in nodeListMap.iteritems():
            try:
                node_labels_gc[v, :] = node_labels[k, :].toarray()
            # Already a numpy array
            except AttributeError:
                node_labels_gc[v, :] = node_labels[k, :]
        node_labels = node_labels_gc
    else:
        node_labels = None

    # Search through the hyperparameter space
    if params["find_hyp"]:
        #choose_best_hyp(data_set, di_graph, node_labels, params)
        bayesian_optimization(data_set, di_graph, node_labels, params)


    # Load best hyperparameter and test it again on new test data
    for d, meth, exp in itertools.product(
        params["dimensions"],
        params["methods"],
        params["experiments"]
    ):
        dim = int(d)
        MethClass = getattr(
            importlib.import_module("gemben.embedding.%s" % meth),
            methClassMap[meth]
        )
        opt_hyp_f_pre = 'gemben/experiments/config/synthetic/%s_%s_%s' % (
            data_set,
            meth,
            params["samp_scheme"]
```

(continues on next page)

(continued from previous page)

```
        )
        try:
            if exp != "viz":
                if exp == 'lpt':
                    model_hyp = json.load(
                        open('%s_lp.conf' % opt_hyp_f_pre, 'r')
                    )
                else:
                    model_hyp = json.load(
                        open('%s_%s.conf' % (opt_hyp_f_pre, exp), 'r')
                    )
            else:
                model_hyp = json.load(
                    open(
                        '%s_%s.conf' % (opt_hyp_f_pre, params["viz_params"]), 'r'
                    )
                )
        except IOError:
            print('Default hyperparameter of the method chosen')
            model_hyp = json.load(
                open('gemben/experiments/config/%s.conf' % meth, 'r')
            )
        hyp = {}
        hyp.update(model_hyp[meth])
        hyp.update({"d": dim})
        if meth == "sdne":
                hyp.update({
                    "modelfile": [
                        "gemben/intermediate/en_mdl_%s_%d.json" % (data_set, dim),
                        "gemben/intermediate/dec_mdl_%s_%d.json" % (data_set, dim)
                    ],
                    "weightfile": [
                        "gemben/intermediate/enc_wts_%s_%d.hdf5" % (data_set, dim),
                        "gemben/intermediate/dec_wts_%s_%d.hdf5" % (data_set, dim)
                    ]
                })
        elif meth == "gf" or meth == "node2vec":
            hyp.update({"data_set": data_set})
        MethObj = MethClass(hyp)
        run_exps(MethObj, meth, dim, di_graph, data_set, node_labels, params)


if __name__ == '__main__':
    ''' Sample usage
    python experiments/exp.py -data sbm -dim 128 -meth sdne -exp gr,lp
    '''
    t1 = time()
    parser = ArgumentParser(description='Graph Embedding Experiments')
    parser.add_argument('-data', '--data_sets',
                        help='dataset names (default: sbm)')
    parser.add_argument('-dim', '--dimensions',
                        help='embedding dimensions list(default: 2^1 to 2^8)')
    parser.add_argument('-meth', '--methods',
                        help='method list (default: all methods)')
    parser.add_argument('-exp', '--experiments',
                        help='exp list (default: gr,lp,viz,nc)')
    parser.add_argument('-lemb', '--load_emb',
```

(continues on next page)

(continued from previous page)

```
                        help='load saved embeddings (default: False)')
    parser.add_argument('-lexp', '--load_exp',
                        help='load saved experiment results (default: False)')
    parser.add_argument('-node_labels', '--node_labels',
                        help='node labels available or not (default: False)')
    parser.add_argument('-rounds', '--rounds',
                        help='number of rounds (default: 5)')
    parser.add_argument('-plot', '--plot',
                        help='plot the results (default: True)')
    parser.add_argument('-plot_d', '--plot_d',
                        help='plot the results wrt dims(default: True)')
    parser.add_argument('-hyp_plot', '--hyp_plot',
                        help='plot the hyperparameter results (default: True)')
    parser.add_argument('-hyp_plot_all', '--hyp_plot_all',
                        help='plot the hyperparameter results (all) (default: True)')
    parser.add_argument('-viz_params', '--viz_params',
                        help='which params to use for viz (default: gr)')
    parser.add_argument('-find_hyp', '--find_hyp',
                        help='find best hyperparameters (default: False)')
    parser.add_argument('-saveMAP', '--save_MAP',
                        help='save MAP in a latex table (default: False)')
    parser.add_argument('-n_samples', '--n_sample_nodes',
                        help='number of sampled nodes (default: 1024)')
    parser.add_argument('-s_sch', '--samp_scheme',
                        help='sampling scheme (default: u_rand)')

    params = json.load(open('gemben/experiments/config/params.conf', 'r'))

    args = vars(parser.parse_args())

    for k, v in args.items():
        if v is not None:
            params[k] = v

    params["experiments"] = params["experiments"].split(',')
    params["data_sets"] = params["data_sets"].split(',')
    params["experiments"] = list(set(params["experiments"]))
    params["data_sets"] = list(set(params["data_sets"]))
    params["rounds"] = int(params["rounds"])
    params["node_labels"] = int(params["node_labels"])
    # params["n_sample_nodes"] = int(params["n_sample_nodes"])
    params["is_undirected"] = bool(int(params["is_undirected"]))



    params["plot_d"] = bool(int(params["plot_d"]))
    params["plot"] = bool(int(params["plot"]))
    params["hyp_plot"] = bool(int(params["hyp_plot"]))
    params["hyp_plot_all"] = bool(int(params["hyp_plot_all"]))
    params["find_hyp"] = bool(int(params["find_hyp"]))

    if params["methods"] == "all":
        params["methods"] = methClassMap.keys()
    else:
        params["methods"] = params["methods"].split(',')
    params["methods"] = list(set(params["methods"]))
```

(continues on next page)

(continued from previous page)

```
    params["dimensions"] = params["dimensions"].split(',')
    params["dimensions"] = list(set(params["dimensions"]))
    if "nc_test_ratio_arr" in params:
        params["nc_test_ratio_arr"] = params["nc_test_ratio_arr"].split(',')
        params["nc_test_ratio_arr"] = \
            [float(ratio) for ratio in params["nc_test_ratio_arr"]]
    try:
      os.makedirs("gemben/intermediate")
    except:
      pass
    try:
      os.makedirs("gemben/results")
    except:
      pass

    for data_set in params["data_sets"]:
        if not int(params["load_exp"]):
            call_exps(params, data_set)
        if int(params["plot"]):
            res_pre = "gemben/results/%s" % data_set
            plot_util.plotExpRes(res_pre, params["methods"],
                                 params["experiments"], params["dimensions"],
                                 'gemben/plots/%s_%s' % (data_set, params["samp_scheme
↪"]),
                                 params["rounds"], params["plot_d"],
                                 params["samp_scheme"])
        if int(params["hyp_plot"]):
            call_plot_hyp(data_set, params)
    if int(params["hyp_plot_all"]):
            call_plot_hyp_all(params["data_sets"], params)
```

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

CHAPTER

SIXTEEN

AUTHORS

16.1 Core Development

Palash~Goyal, Di Huang, Ankita Goswami
University of Southern California
Information Sciences Institute

Sujit Rokka Chhetri
University of California, Irvine
Email: schhetri@uci.edu

Arquimedes Canedo
Siemens Corporate Technology

Emilio Ferrara
University of Southern California
Information Sciences Institute

CHAPTER

SEVENTEEN

CITING GEMBEN

If you found this open source library useful, please kindly cite us:

```
@online{gem-ben,
  author = {Palash Goyal, Di Huang, Ankita Goswami, Sujit Rokka Chhetri, Arquimedes␣
↪Canedo and Emilio Ferrara},
  title = {{Gem-Benchmark},
  year = 2019,
  url = {https://gem.readthedocs.io}
  }
```

CHAPTER

EIGHTEEN

LICENSE

The MIT License

Copyright (c) 2018 The Python Packaging Authority

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PYTHON MODULE INDEX

g

INDEX

H

I

J

L