CS768: Learning with Graphs

GELib: A Graph Embedding Library for Common Graphs

BY

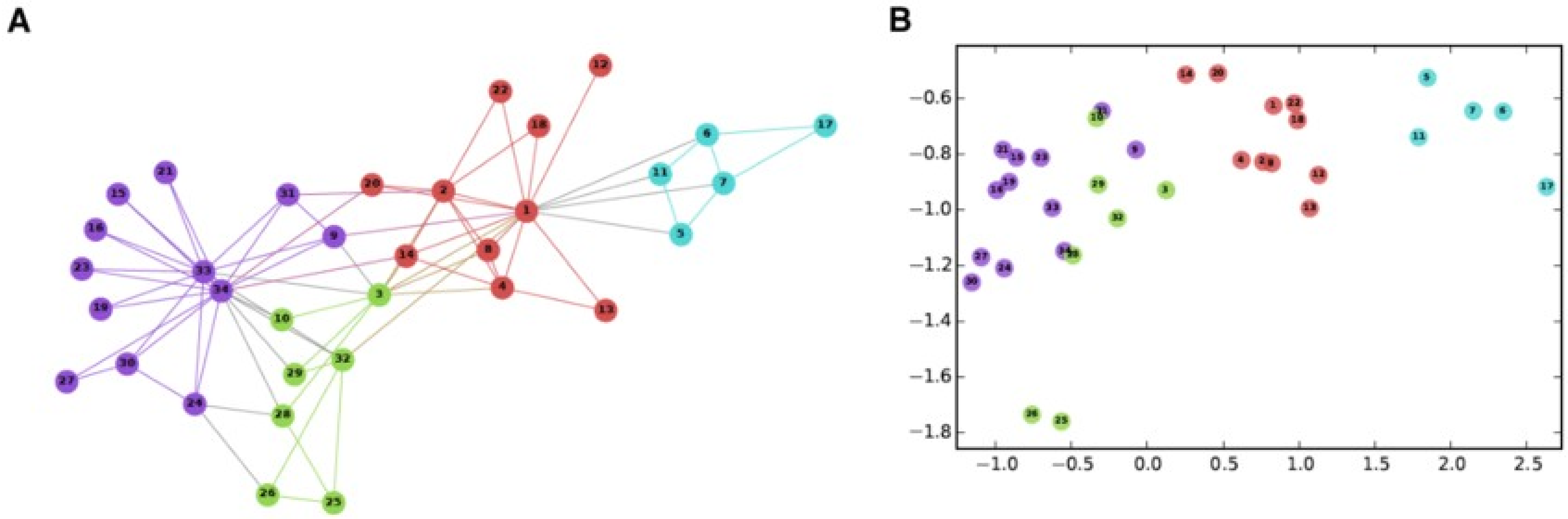
Kaveri Kale 194057002

Arjun Kashettiwar 180050012

Anish MM 203050066

Introduction to Graph Embedding

- Graph embedding provides an effective yet efficient way to solve the graph analytics problem. Specifically, graph embedding converts a graph into a low dimensional space in which the graph information is preserved.

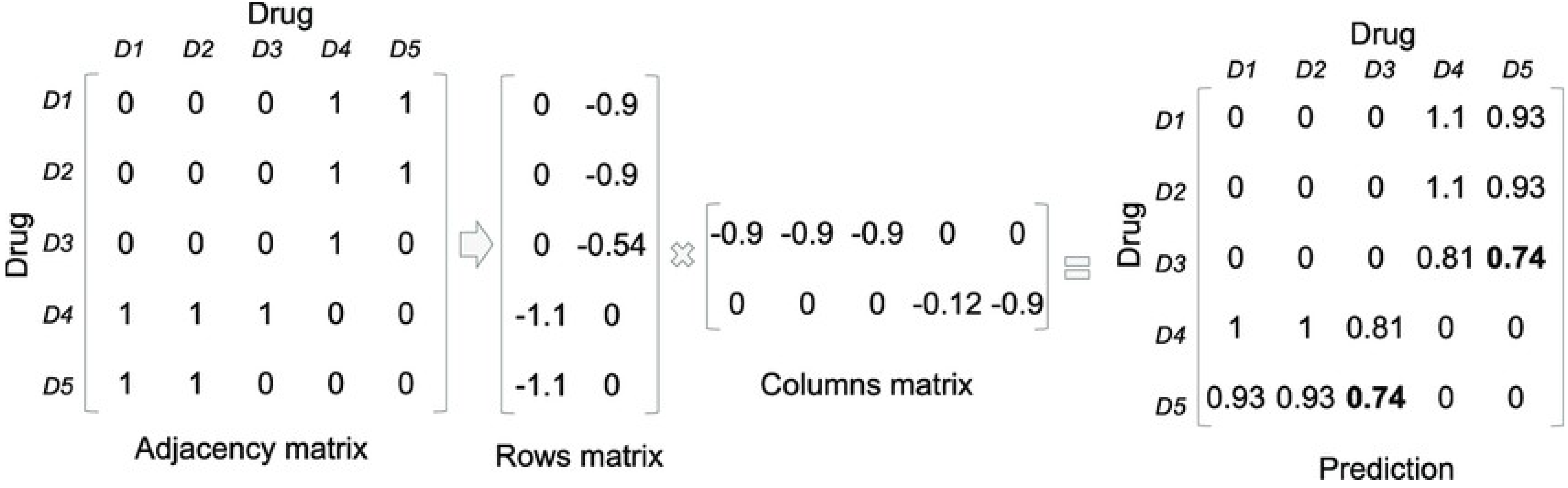


Problem Statement

- Generate graph embedding for given graph.
- Input : Graph edge-list file.
- Output : Graph embedding R^d , where is d is the embedding dimention $d<<|V|$

Matrix Factorization Based Embedding

- Matrix factorization based graph embedding represent graph property (e.g., node pairwise similarity) in the form of a matrix and factorize this matrix to obtain node embedding.
- Connection between two nodes are represented in matrix form.
- Matrices used for representation are node adjacency matrix, Laplacian matrix, node transition probability matrix, and Katz similarity matrix etc.



Matrix Factorization Based Methods

- Laplacian Eigenmaps
- Graph Factorization
- HOPE
- GraRep
- LINE
- Singular Value Decomposition (SVD)

Laplacian Eigenmaps

- If W_{ij} is high then it keeps the embedding of two nodes close.
- Objective function :
- $$\Phi(Y)=\frac{1}{2}\sum_{i,j}|Y_i-Y_j|^2 W_{ij}=tr(Y^T LY)$$
- where L is the Laplacian of graph G. The objective function is subjected to the constraint $Y^T DY = I$ to eliminate trivial solution.
- The solution to this can be obtained by taking the eigenvectors corresponding to the d smallest eigenvalues of the normalized Laplacian,
- $$L_{norm}=D^{-1/2}L\,D^{-1/2}$$

GF

- To obtain the embedding, GF factorizes the adjacency matrix of the graph, minimizing the following loss function
- $\varphi(Y,\lambda)=\frac{1}{2}\sum_{(i,j)\in E}\left(W_{ij}-\langle Y_i,Y_j\rangle\right)^2+\frac{\lambda}{2}\sum_i \|Y_i\|^2$
- where λ is a regularization coefficient.

HOPE

- HOPE preserves higher order proximity by

  minimizing $\|S - Y_s Y_t^T\|^2_F$

  where S is some similarity matrix.

- Similarity can be Katz Index, Common Neighbors, Adamic-Adar score, etc.
- Each similarity measure is represented as $S = M_g^{-1} M_l$, where both M_g and M_l are sparse. Then generalized SingularValue Decomposition (SVD) is used to obtain the embeddings efficiently.

Random Walk Based Methods

- Deep learning based graph embedding, a graph is represented as a set of random walk paths sampled from it. The deep learning methods are then applied to the sampled paths for graph embedding which preserves graph properties carried by the paths.
- **Methods:**
- node2vec
- DeepWalk

DeepWalk

- DeepWalk preserves higher-order proximity between nodes by maximizing the probability of observing the last k nodes and the next k nodes in the random walk centered at V_i , i.e.

$$\text{maximizing} \;\; \log \Pr(v_{i-k}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+k} | Y_i),$$

  where 2k + 1 is the length of the random walk.

- The model generates multiple random walks each of length 2k + 1 and performs the optimization over sum of log-likelihoods for each random walk.

node2vec

- The crucial difference from DeepWalk is that node2vec employs biased-random walks that provide a trade-off between breadth-first (BFS) and depth-first (DFS) graph searches, and hence produces higher-quality and more informative embeddings than DeepWalk.
- Choosing the right balance enables node2vec to preserve community structure as well as structural equivalence between nodes.

Deep Learning Based Methods

- Structural Deep Network Embedding (SDNE)
- Graph Auto-Encoder (GCN based)
- Variational Graph Auto-Encoder (GCN based)

SDNE

- After obtaining $y^{(K)}_i$, we can obtain the output $\hat{x}_i$ by reversing the calculation process of encoder. The goal of the autoencoder is to minimize the reconstruction error of the output and the input.
- The loss function is shown as follows:

$$\mathcal{L} = \sum_{i=1}^{n} \|\hat{\mathbf{x}}_i - \mathbf{x}_i\|_2^2$$

$$\begin{aligned}\mathcal{L}_{1st} &= \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i^{(K)} - \mathbf{y}_j^{(K)}\|_2^2 \\ &= \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{2nd} &= \sum_{i=1}^{n} \|(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{b_i}\|_2^2 \\ &= \|(\hat{X} - X) \odot B\|_F^2\end{aligned}$$

$$\begin{aligned}\mathcal{L}_{mix} &= \mathcal{L}_{2nd} + \alpha \mathcal{L}_{1st} + \nu \mathcal{L}_{reg} \\ &= \|(\hat{X} - X) \odot B\|_F^2 + \alpha \sum_{i,j=1}^{n} s_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 + \nu \mathcal{L}_{reg}\end{aligned}$$

GCN-AE and GCN-VAE

- These models use a graph convolutional network (GCN) encoder and an inner product decoder.
- The input is adjacency matrix and they rely on GCN to learn the higher order dependencies betweennodes.
- It has been empirically shown that using variational autoencoders can improve performance compared to non-probabilistic autoencoders.

GELib(Graph Embedding Library)

- GELib is a Python package which offers a general framework for graph embedding methods.
- **Features :**
- The main features of this libarary are:
  - A variety of graph embedding models have been incorporated.
  - Two evaluation tasks are provided.
  - Results can be easily visualized and stored.
  - A unified interface/pipeline has been provided to use all these facilities
- It implements many state-of-the-art embedding techniques including **Laplacian Eigenmaps, Graph Factorization, SVD , GraRep, LINE, Higher-Order Proximity preserved Embedding (HOPE), Structural Deep Network Embedding (SDNE) node2vec, DeepWalk, GCN-Autoencoder, GCN-Variational Autoencoder** .
- The framework implements functions to evaluate the quality of obtained embedding including link prediction and node classification.

Graph Format

- We store all graphs using the DiGraph as directed weighted graph in python package networkx. The weight of an edge is stored as attribute "weight". We save each edge in undirected graph as two directed edges.

Python Libraries and Packages Used

- Networkx – Graph
- Scipy, numpy – Numerical computation
- Tensorflow – Neural Network
- Matplotlib – Visualization

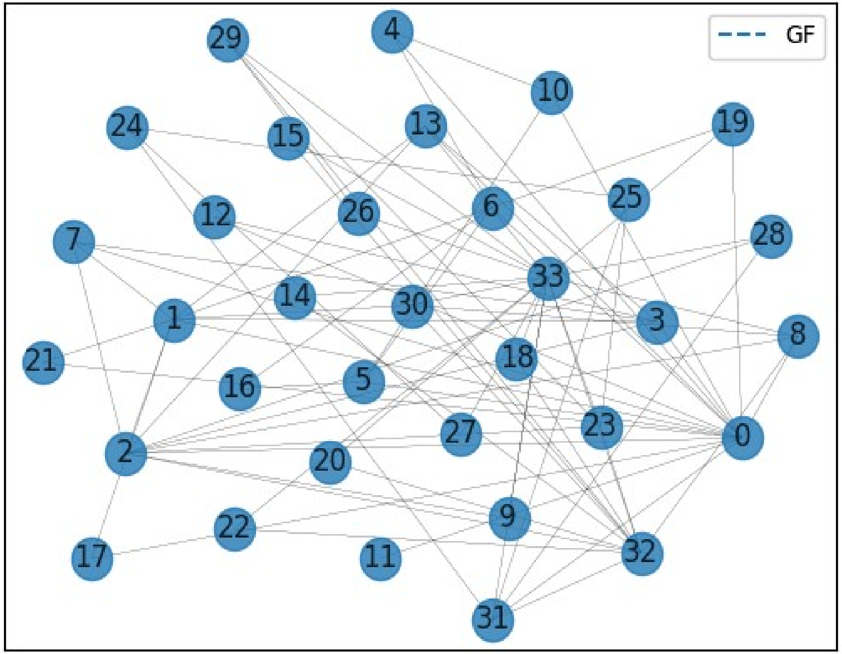
Repository Structure

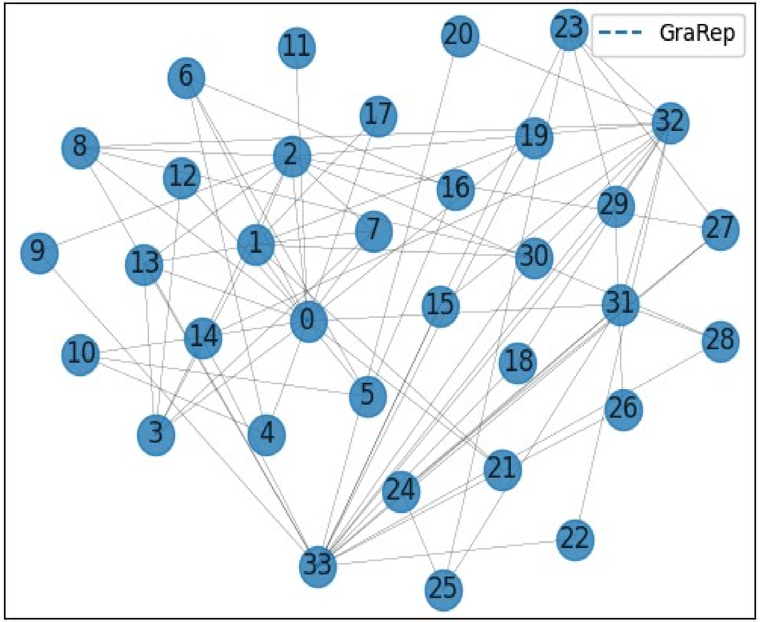
```
graph_embedding
├── eval_metrics
│   ├── __init__.py
│   └── metrics.py
├── evaluation
│   ├── evaluation.py
│   └── __init__.py
├── __init__.py
├── __main__.py
├── main.py
├── MF_RW
│   ├── classify.py
│   ├── gf.py
│   ├── graph.py
│   ├── grarep.py
│   ├── hope.py
│   ├── __init__.py
│   ├── lap.py
│   ├── line.py
│   ├── node2vec.py
│   ├── sdne.py
│   └── walker.py
├── NN
│   ├── __init__.py
│   ├── layers.py
│   ├── model.py
│   ├── optimizer.py
│   ├── preprocessing.py
│   └── train_model.py
├── pipeline.py
├── SVD
│   ├── __init__.py
│   └── model.py
├── training
│   ├── embed_train.py
│   ├── __init__.py
│   └── utils.py
└── visualization
    ├── graph_util.py
    ├── __init__.py
    ├── plot_util.py
    └── visualize_embedding.py
```

Evaluation Metrics

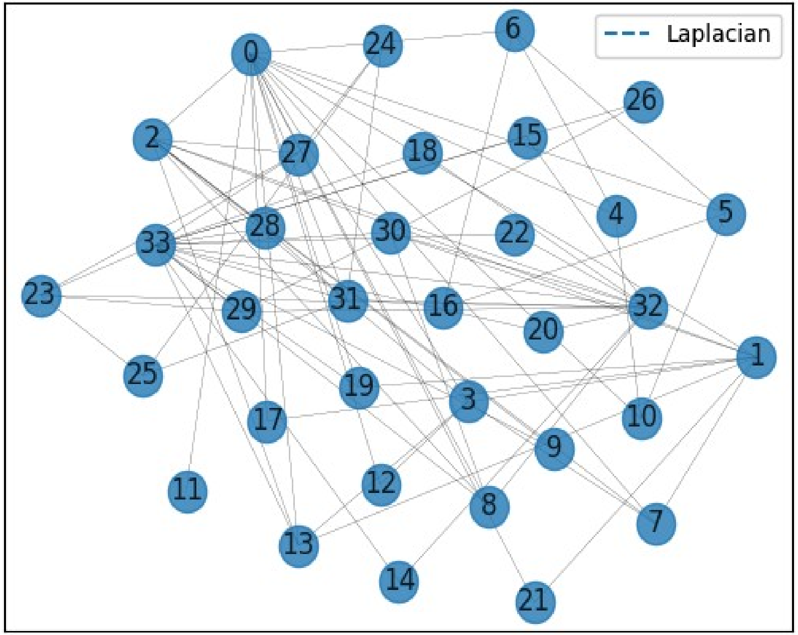
- Mean Reciprocal Rank (MRR)
- Mean Average Precision (MAP)
- F1 Score
- AUC-ROC
- AUC-PR

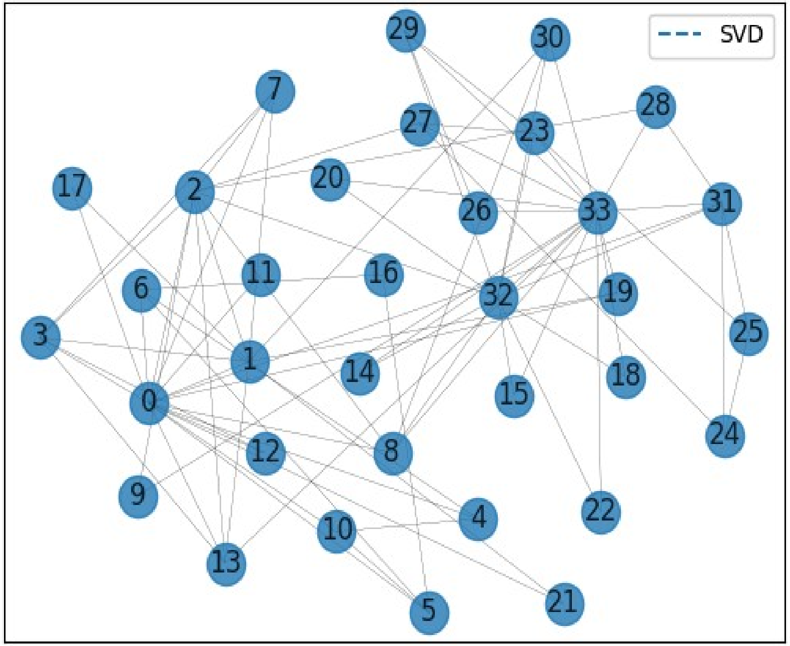
GF and GraRep



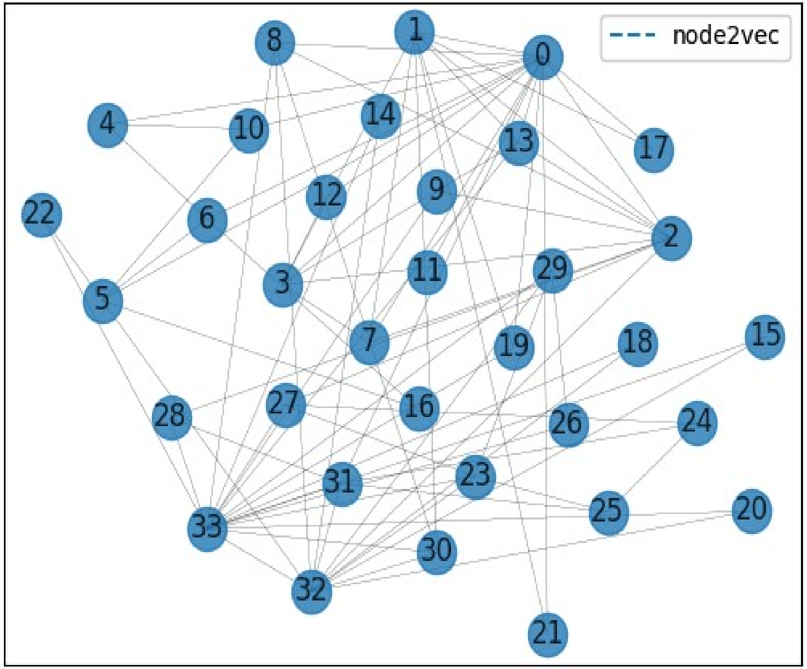


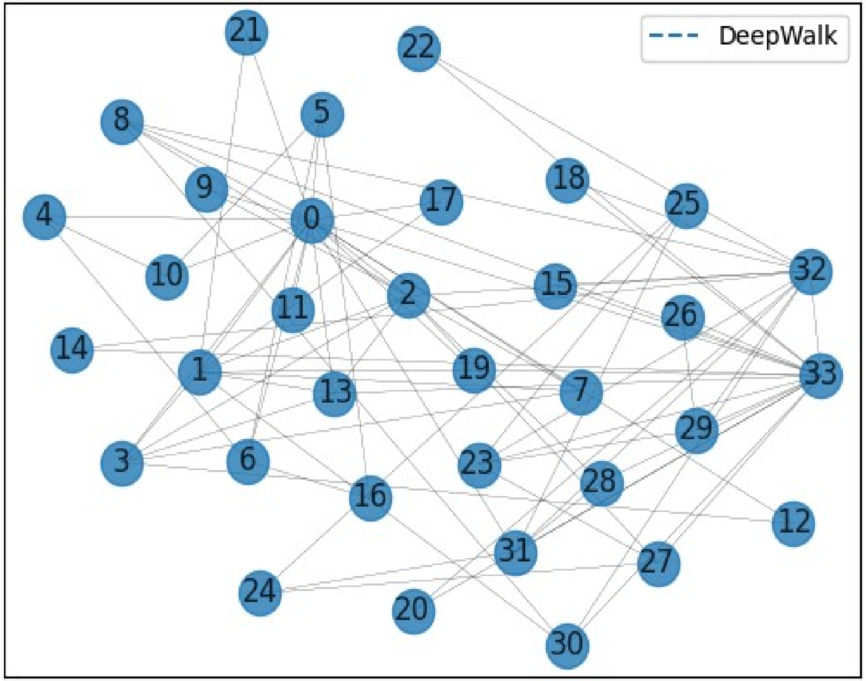
Laplacian and SVD



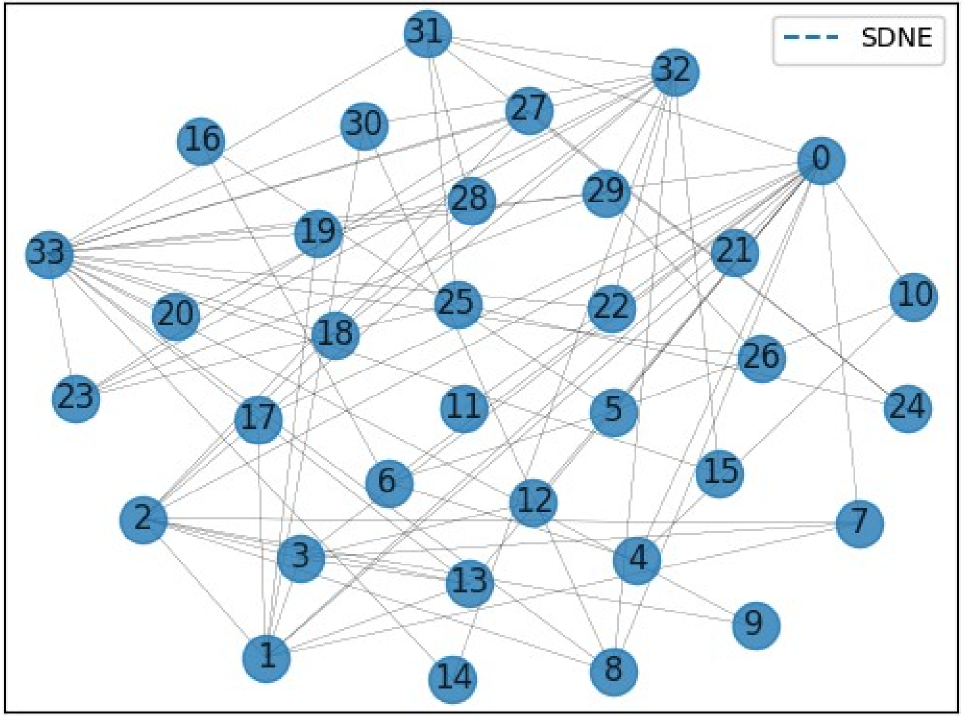


node2vec and DeepWalk



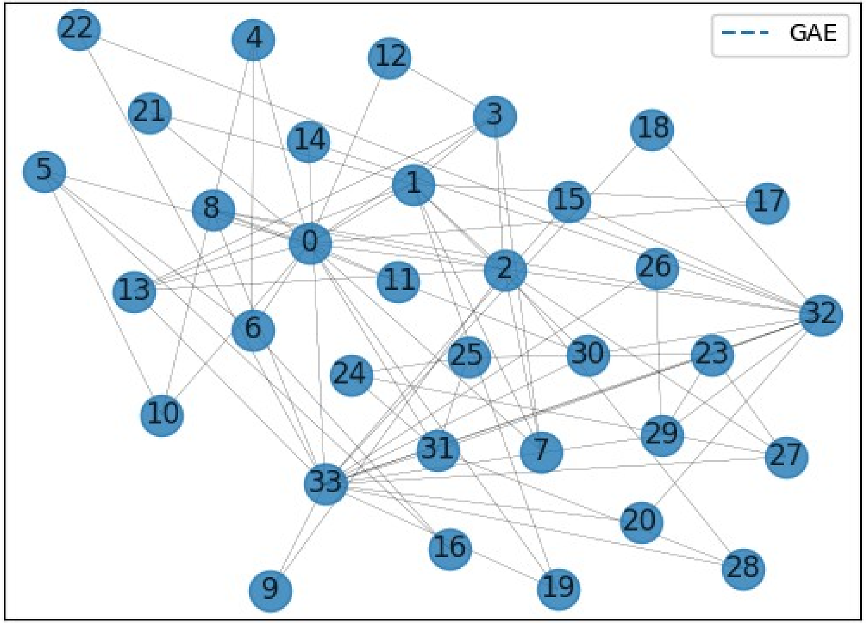


SDNE

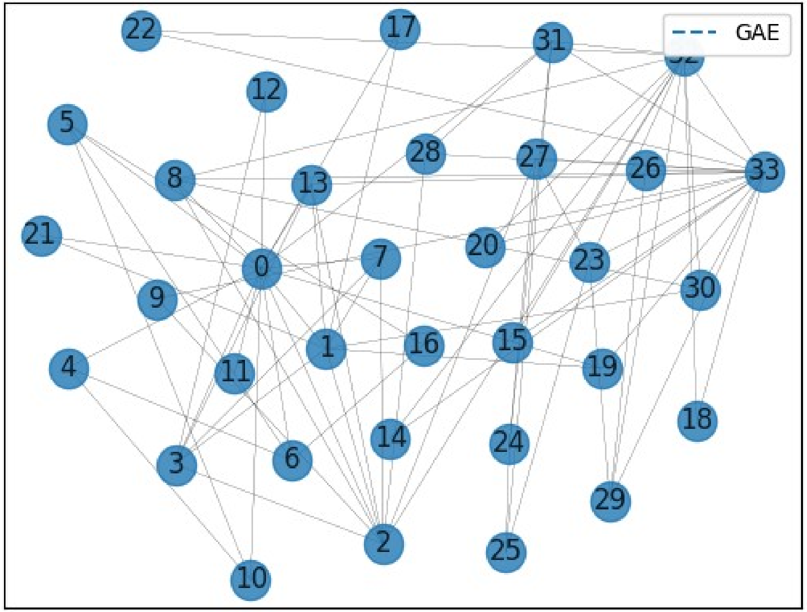


SDNE
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

GCN-AE and GCN-VAE



GAE-AE



GAE-VAE

Link Prediction task evaluation:Karate Graph

Methods	MAP	MRR	AUC-ROC	AUC-PR	Accuracy	F1-score
Laplacian	0.867	0.867	0.916	0.925	0.800	0.786
GF	0.719	0.719	0.591	0.677	0.600	0.538
SVD	0.609	0.609	0.622	0.667	0.533	0.533
HOPE	0.000	0.000	0.500	0.500	0.500	0.000
GraRep	0.778	0.778	0.884	0.907	0.800	0.800
node2vec	0.630	0.630	0.689	0.701	0.633	0.645
DeepWalk	0.652	0.652	0.689	0.695	0.667	0.706
LINE	0.611	0.611	0.689	0.697	0.667	0.688
SDNE	0.867	0.867	0.920	0.930	0.800	0.786
GCN-AE	0.639	0.639	0.631	0.561	0.667	0.643
GCN-VAE	0.620	0.620	0.644	0.562	0.700	0.727

Link Prediction task evaluation: Facebook Graph

Methods	MAP	MRR	AUC-ROC	AUC-PR	Accuracy	F1-score
Laplacian	0.592	0.627	0.672	0.646	0.619	0.626
GF	0.642	0.655	0.847	0.828	0.770	0.773
SVD	0.744	0.752	0.919	0.891	0.854	0.858
HOPE	0.325,	0.248	0.604	0.618	0.579	0.298
GraRep	0.782	0.777	0.941	0.891	0.893	0.898
node2vec	0.733	0.731	0.919	0.877	0.859	0.864
DeepWalk	0.766	0.763	0.935	0.892	0.892	0.886
LINE	0.790	0.802	0.927	0.887	0.873	0.878
SDNE	0.727	0.733	0.898	0.866	0.828	0.834
GCN-AE	0.666	0.613	0.878	0.786	0.840	0.849
GCN-VAE	0.597	0.547	0.852	0.768	0.798	0.808

Demo

- Demo to create pipeline

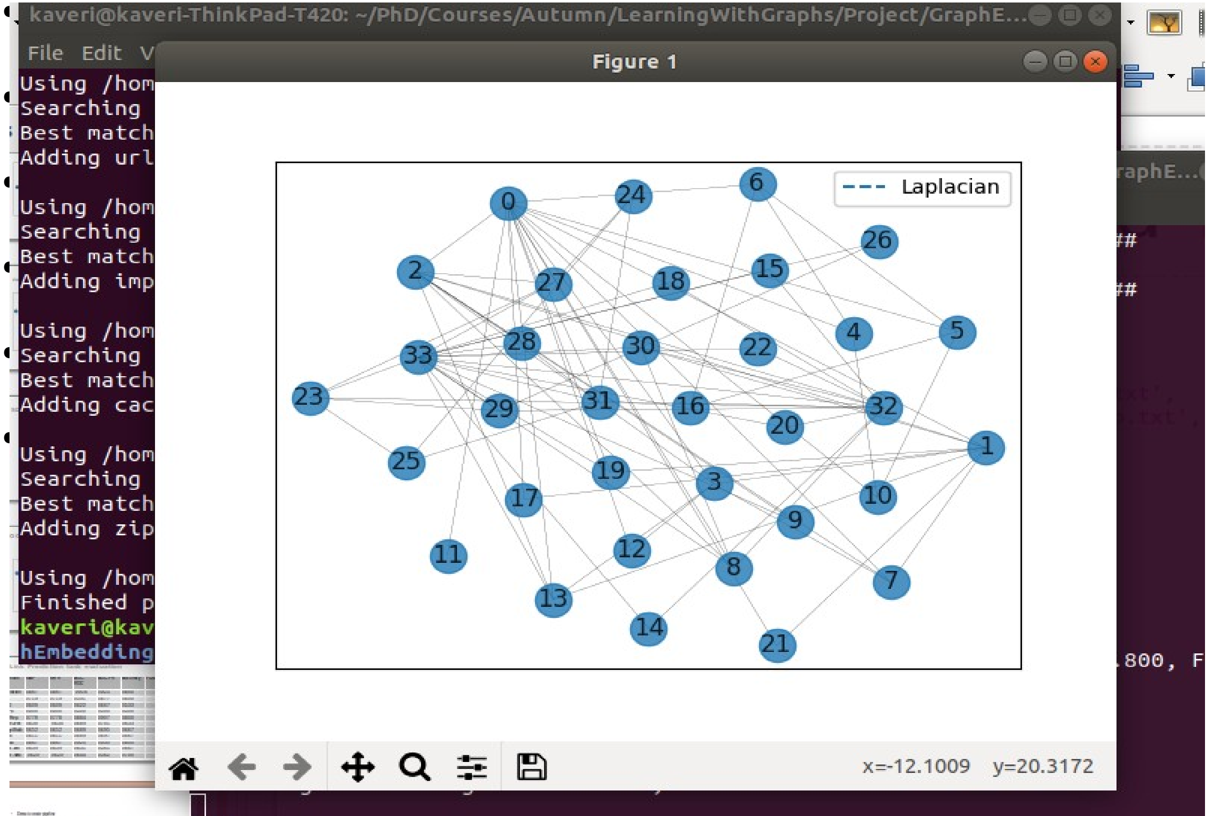
```
pipeline = Pipeline()
pipeline.execute_pipeline(dataset = '../../data/karate.txt',
                          output = '../../embeddings/emb.txt',
                          method = 'Laplacian',
                          task = 'link-prediction',
                          dimensions = 10)
```

Demo : python interpreter

```
Python 3.7.9 (default, Aug 18 2020, 02:07:21)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from graph_embedding.pipeline import Pipeline
>>> p = Pipeline()
>>> p.execute_pipeline(dataset="./data/karate.txt",
...                    output="./embeddings/sample_out.txt",
...                    method="Laplacian",
...                    task="link-prediction",
...                    dimensions=10)
######################################################################
Embedding Method: Laplacian, Evaluation Task: link-prediction
######################################################################
Original Graph: nodes: 34 edges: 78
Training Graph: nodes: 34 edges: 64
Loading training graph for learning embedding...
Graph Loaded...
begin norm_lap_mat
finish norm_lap_mat
finish getLap...
finish eigh(lap_mat)...
Saving embeddings...
Embedding Learning Time: 0.01 s
Nodes with embedding: 34
Begin evaluation...
######### Link Prediction Evaluation #########
MAP : 0.867, MRR : 0.867, AUC-ROC: 0.916, AUC-PR: 0.925, Accuracy: 0.800, F1: 0.786
##################################################
Prediction Task Time: 0.01 s
```

Output : Graph Embedding Visualization

- It will give visualization for embedding.



Output : Evaluation Result

- It will print evaluation result

```
######################################################################
Embedding Method: Laplacian, Evaluation Task: link-prediction
######################################################################
Original Graph: nodes: 34 edges: 78
Training Graph: nodes: 34 edges: 64
Loading training graph for learning embedding...
graph_train.edgelist
Graph Loaded...
begin norm_lap_mat
finish norm_lap_mat
finish getLap...
finish eigh(lap_mat)...
Saving embeddings...
Embedding Learning Time: 0.01 s
Nodes with embedding: 34
Begin evaluation...
######### Link Prediction Evaluation #########
MAP : 0.867, MRR : 0.867, AUC-ROC: 0.916, AUC-PR: 0.925, Accuracy: 0.800, F1: 0.786
##################################################
Prediction Task Time: 0.01 s
```

Conclusion

- In GELib we have tried to incorporate various types of algorithms for graph embeddings.
- Our intention is to make this repository contain implementations of all the major algorithms for this task to make model comparison easier.
- We have also attempted to provide a single API that allows the user to run all the incorporated models.
- We have also provided functionalities to load data, visualize created embeddings, store the embeddings and various evaluation metrics.

Future Work

- More models need to be added.
- Want to add more analysis tasks like Graph Reconstruction, Visualization, Clustering etc.
- Want to extend this library for Knowledge Graph Embedding methods.
- Restructure the code to mimic the structure of thePyKEEN library which implements various algorithms for Knowledge Graph embeddings in a modular and elegant manner.

Thank You