SoftwareX 17 (2022) 100997

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

EvalNE: A framework for network embedding evaluation

Alexandru Mara *, Jefrey Lijffijt, Tijl De Bie

Department of Electronics and Information Systems, Ghent University, Technologiepark 122, 9052 Ghent, Belgium

ARTICLE INFO

Article history:
Received 23 March 2021
Received in revised form 12 January 2022
Accepted 12 January 2022

Keywords:
Representation learning
Evaluation
Reproducibility
Open-source tools

ABSTRACT

In this paper we introduce EvalNE, a Python toolbox for network embedding evaluation. The main goal of EvalNE is to aid researchers and practitioners in performing consistent and reproducible evaluations of new embedding methods, replicating existing evaluations, and conducting benchmark studies. The toolbox can evaluate models independently of their programming language and assess the quality of learned representations through data visualization and downstream tasks such as sign and link prediction, network reconstruction, and node multi-label classification. EvalNE streamlines evaluation by providing automation and abstraction for tasks such as hyperparameter tuning and model validation, node and edge sampling, node-pair embedding computation, and performance reporting. As a command line tool, configuration files describe the evaluation setup and guarantee consistency and reproducibility. As an API, EvalNE provides the building blocks to design any evaluation setup while minimizing the risk of evaluation errors.

Code metadata

Current code version	v0.3.3
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00057
Code Ocean compute capsule	N/A
Legal Code License	MIT
Code versioning system used	Git
Software code languages, tools, and services used	Python 2.7.x & 3.6.x
Compilation requirements, operating environments & dependencies	Numpy, Scipy, NetworkX, Scikit-learn, Pandas, Matplotlib, Tqdm
If available Link to developer documentation/manual	https://evalne.readthedocs.io/en/latest/
Support email for questions	alexandru.mara@ugent.be

Software metadata

Current software version	v0.3.3
Permanent link to executables of this version	https://github.com/Dru-Mara/EvalNE/releases/tag/v0.3.3
Legal Software License	MIT
Computing platforms/Operating Systems	Linux, macOS, Microsoft Windows
Installation requirements & dependencies	Numpy, Scipy, NetworkX, Scikit-learn, Pandas, Matplotlib, Tqdm
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://evalne.readthedocs.io/en/latest/
Support email for questions	alexandru.mara@ugent.be

1. Motivation and significance

Network embedding (NE) or network representation learning methods aim to learn low-dimensional representations of network nodes as vectors, typically in the Euclidean space. These representations can then be directly used for network visualization or to efficiently perform a variety of downstream prediction

* Corresponding author.
E-mail addresses: alexandru.mara@ugent.be (Alexandru Mara), jefrey.lijffijt@ugent.be (Jefrey Lijffijt), tijl.debie@ugent.be (Tijl De Bie).

https://doi.org/10.1016/j.softx.2022.100997

tasks. These tasks include sign prediction [1,2], link prediction [3,4], network reconstruction [5,6], and node classification [7,8]. A higher performance on these downstream tasks is then generally associated with a better representation of the network.

Evaluating NE methods based on downstream task performance, however, is challenging and requires a number of additional steps and design choices which can confound the results, harm reproducibility, and are prone to errors. Firstly, for each downstream task one must ensure that the input data is preprocessed consistently, such that all methods are correctly evaluated. For instance, many NE methods require the input network to contain a single connected component (e.g. [4]) while some do not (e.g. [6]). Additionally, the necessary sets of train and test data (nodes or edges) can be selected according to a variety of approaches and be of different sizes [9]. Two popular choices for edge sampling, for example, are pseudo-random train-test split with certain constraints or timestamp-based sampling (i.e., use the most recent edges for testing and the remaining ones for training). Another source of evaluation inconsistencies is negative sampling [10]. This technique is commonly used in link prediction evaluation to generate the *negative class* for the binary classification of edges and non-edges. The negative sampling approach and the relative sizes of the train and test non-edge sets vary between scientific studies. Yet another challenge in NE evaluation is that embedding methods provide different types of outputs. While some approaches only output node embeddings [11], others can provide node-pair embeddings [12] or even node similarities [4]. As such, specific pipelines for different output types are required. Particularly important, in this case, is the computation of node-pair embeddings from node embeddings which has been shown to strongly impact embedding quality [3]. Finally, additional complexity stems from hyperparameter tuning, not only for the evaluated embedding methods, but also for the downstream task itself.

To address the above-mentioned challenges and simplify the evaluation of NE methods on downstream tasks, we introduce EvalNE. The toolbox can be used both as a standalone application or integrated in existing code.

EvalNE as a command line tool. EvalNE leverages configuration files that allow the user to describe complete experimental setups, from the tasks, datasets and methods to use, to preprocessing, sampling, hyperparameter tuning and metrics to optimize. These configuration files provide flexibility to define or replicate different experimental setups and are sufficient for complete reproducibility. After evaluation, detailed performance reports and graphics are generated as well as log files detailing the issues encountered.

EvalNE as an API. The toolbox provides access to a series of classes and functions implementing various components required for method evaluation on different downstream tasks (see Fig. 1). Users can combine and modify these building blocks to produce new evaluation pipelines. The library does not implement any NE method but provides the necessary accessories to conveniently evaluate any off-the-shelf approach. The framework does, however, implement a number of link prediction heuristics (see Section 3.2).

2. Related software

To the best of our knowledge, no other libraries provide similar levels of automation and flexibility as EvalNE for network embedding evaluation. A recently proposed framework [13], which draws inspiration from EvalNE, is geared towards evaluating embeddings on semantic tasks and is limited to node embedding approaches only. Other libraries such as `OpenNE` [14] and `GEM` [15] focus mainly on providing implementations of NE methods and present limited evaluation capabilities. Their evaluation pipelines are rigid and tailored specifically to the approaches they implement, particular datasets and small sets of downstream tasks (node classification in both cases and additionally link prediction for `GEM`). The `Karateclub` [16] library is designed to aid users in coding their own evaluations. Finally, other initiatives such as the Open Graph Benchmark [17] collect challenging datasets for NE evaluation. The associated `OGB` Python package also offer support for data preprocessing and a set of evaluators specifically tailored to the benchmark datasets.

Unlike these frameworks, EvalNE can evaluate methods with different types of output, from node to node-pair embeddings and node-pair similarities. Our framework is not limited to a predefined set of networks or methods available in a particular library. Instead, it can evaluate any available approach (note that this includes all implementations in the above-mentioned libraries). EvalNE also does not require users to write their own evaluation pipelines nor significantly modify their code to fit a particular API.

3. Software description

EvalNE is available on GitHub and is released under the MIT free software license. The library's code style complies with PEP 8 and the docstring documentation follows the standard Numpy format. The toolbox is compatible with Python 2 and Python 3 and can be easily installed using `pip`. Supported platforms include Linux, macOS, and Microsoft Windows. EvalNE only depends on a small number of popular open-source packages and others such as `OpenNE` and `GEM` are optional and can provide implementations of different NE methods. The toolbox documentation is automatically managed and hosted online by Read the Docs. It provides detailed instructions on the installation and main functionalities as well as a range of examples for both command line and API use. Finally, the library contains a set of pre-filled configuration files which allow one to reproduce the experimental sections of several influential papers on NE.

3.1. Software architecture

Conceptually, the design of EvalNE can be seen as a set of interconnected building blocks which provide all the necessary components for evaluation. This modular structure, shown in Fig. 1, simplifies code maintenance and addition of new features and allows one to evaluate methods with different output types (node embeddings, node-pair embeddings, node-pair similarities, etc.) on different downstream tasks.

3.2. Software functionalities

The building blocks introduced in Section 3.1 are composed of a variety of classes and methods offering the following functionalities.

Data preprocessing. Building on the popular `NetworkX` [18] framework, EvalNE offers additional functions for loading and storing networks, pruning and relabeling nodes, removing self-loops, sampling edges, restricting networks to the main connected component and obtaining common statistics.

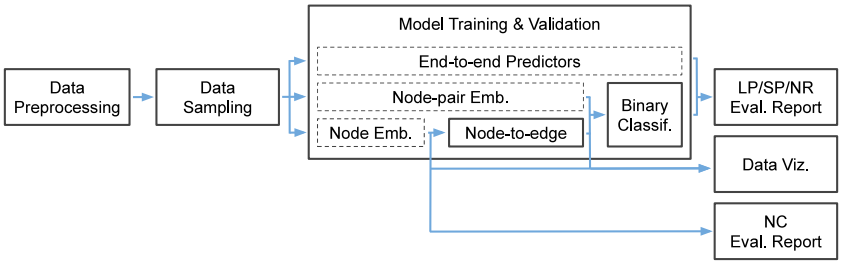

Fig. 1. Block diagram of EvalNE showing the methods and tasks that can be evaluated. Downstream link prediction, sign prediction, network reconstruction and node classification tasks are abbreviated as LP, SP, NR, and NC, respectively. Dashed blocks correspond to user-specified methods with different output types while the remaining blocks are provided by EvalNE.

Data sampling. For evaluation, sets of train, validation, and test data (nodes or edges) must be obtained from the input networks. The particular sampling strategy varies for each downstream task. In node classification, for instance, this is relatively straightforward as test/validation nodes can be sampled randomly. For the remaining tasks, which require sets of edges, we provide a variety of sampling methods. From timestamp-based to random sampling that ensures the train edges continue to span a connected subgraph (a key requirement for many NE methods). For negative sampling, EvalNE provides two alternatives: the *open world* and the *closed world* assumptions. The former considers the case where non-edges are *not known* a priori and, thus, must be sampled from the train graph spanned by the train edges. In this case, the sampled non-edges may overlap with the test edges. The latter considers the case where non-edges are known a priori and, thus, the train non-edges do not overlap with the test edges. The *EvalSplit* class encapsulates the sampling and negative sampling functionalities.

Model training & validation. EvalNE provides specific classes called *Evaluators* for each downstream task. These classes manage model training with appropriate input data, hyperparameter tuning via grid search and collection of results. For sign and link prediction and network reconstruction, methods providing node similarities can be directly evaluated. Those that output node or node-pair embeddings are evaluated through binary classification. EvalNE supports any Scikit-learn binary classifier and implements Logistic Regression with cross-validation as a default. Methods that only output node embeddings additionally require a binary operator to compute node-pair embeddings. For this, EvalNE implements the following operators: average, Hadamard, weighted L_1 and weighted L_2 (see [3]).

Evaluation report & visualization. EvalNE can evaluate method scalability through wall clock time, and performance through fixed-threshold metrics and threshold curves. The library implements over 10 different fixed-threshold metrics including AUC, precision, recall, and F-score. Integrated threshold curves, which present method performance for a range of threshold values, include precision–recall [19] and ROC [20] curves. Methods to visualize embeddings and graphs are also provided as well as dimensionality reduction techniques to map higher-dimensional embeddings to 2D. Method specific performance is recorded in *Results* objects while global evaluation summaries are provided through *Scoresheet* objects.

Baseline heuristics. Specifically for the link prediction task, the toolbox provides a set of heuristic methods for both directed and undirected networks. These heuristics are based on: (i) node-pair similarities, i.e., common neighbors, Jaccard coefficient, Adamic–Adar index, preferential attachment, resource allocation, cosine similarity, Leicht–Holme–Newman index and topological overlap, (ii) paths, i.e., Katz similarity, and (iii) embeddings, i.e., all_ baselines (the concatenation of five heuristics as an embedding).

4. Illustrative examples

In this section, we present two examples that showcase the use of EvalNE as a command line application and as a Python API.

4.1. Command line example

When using EvalNE as a command line application a configuration file describing the evaluation setup is required. An example of one such file is presented in Appendix A of the supplementary material. The evaluation parameters in the configuration file are grouped into 8 categories or sections also described in Appendix A. These sections cover general task-related parameters, data ingestion and preprocessing, edge sampling, methods to evaluate and results reporting.

Once the configuration file is filled, the evaluation can be started using the following command: `python -m evalne <config_file>.ini`. The toolbox will automatically log any issues such as failed method executions, provide a tabular output and two pickle files containing the train and test evaluation scoresheets.

4.2. API example

A code snippet showing the use of EvalNE as an API to evaluate five different methods on downstream link prediction is presented in Appendix B of the supplementary file. In this example, we first read and preprocess a new dataset, crate an *LPEvalSplit* object that will contain the train and test edges and non-edges and initialize a link prediction evaluator. Then, we create a scoresheet object to store the evaluation results and select three baseline heuristics and two NE methods to be evaluated. Finally, we launch the evaluation and present the results.

5. Impact

Recent research has found that progress is stalling in many areas within AI [21–23], and that at the core of this phenomenon is the so called reproducibility crisis. Tools such as EvalNE and others inspired by it, e.g. [13], have the potential to minimize this crisis by enabling reproducible and consistent evaluations and large-scale benchmarks. One such large-scale evaluation using EvalNE has already been conducted, specifically for the link prediction task [9]. Moreover, the ability to use configuration files to describe rich experimental setups presents two particular benefits. On the one hand, evaluations are simplified as many repetitive tasks such as data preprocessing or hyperparameter tuning are automated. On the other hand, ensuring that other practitioners can replicate an evaluation, reduces to making the configuration files accessible. The authors of [2,24] have already explored these benefits when conducting a link prediction evaluation on large-scale networks and a sign prediction evaluation,

respectively. We also note that the software has attracted some attention in other research areas with BSc and MSc thesis in the fields of Mathematics [25], Biology [26] and Computer Science [27] making use of its capabilities. Finally, the user base of EvalNE has been continuously growing for the past months as shown by several popularity indicators of our GitHub repository.

6. Conclusions

In this paper we have introduced EvalNE, a Python toolbox for consistent and reproducible evaluation of network embedding methods. The toolbox allows users to compare NE methods and perform benchmarks on a variety of downstream tasks using automated yet highly flexible evaluation pipelines. In the near future, we expect to broaden the impact of our framework with support for additional tasks such as clustering and node classification through link prediction, direct integration with libraries providing NE method implementations, extended visualization capabilities and a GUI.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007–2013) / ERC Grant Agreement no. 615517, from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme, and from the FWO (project no. G091017N, G0F9816N, 3G042220).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2022.100997.

References

[1] Wang S, Tang J, Aggarwal C, Chang Y, Liu H. Signed network embedding in social media. In: Proceedings of the 17th SIAM international conference on data mining. 2017, p. 327–35. http://dx.doi.org/10.1137/1.9781611974973.37.

[2] Mara A, Mashayekhi Y, Lijffijt J, de Bie T. CSNE: Conditional signed network embedding. In: Proceedings of the 29th ACM international conference on information & knowledge management. 2020, p. 1105–14. http://dx.doi.org/10.1145/3340531.3411959.

[3] Grover A, Leskovec J. Node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016, p. 855–64. http://dx.doi.org/10.1145/2939672.2939754.

[4] Kang B, Lijffijt J, De Bie T. Conditional network embeddings. In: Proceedings of the 7th international conference on learning representations. 2019, URL https://openreview.net/forum?id=ryepUj0qtX.

[5] Wang D, Cui P, Zhu W. Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016, p. 1225–34. http://dx.doi.org/10.1145/2939672.2939753.

[6] Zhang Z, Cui P, Wang X, Pei J, Yao X, Zhu W. Arbitrary-order proximity preserved network embedding. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. 2018, p. 2778–86. http://dx.doi.org/10.1145/3219819.3219969.

[7] Perozzi B, Al-Rfou R, Skiena S. DeepWalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. 2014, p. 701–10. http://dx.doi.org/10.1145/2623330.2623732.

[8] Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q. LINE: Large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web. 2015, p. 1067–77. http://dx.doi.org/10.1145/2736277.2741093.

[9] Mara A, Lijffijt J, De Bie T. Benchmarking network embedding models for link prediction : are we making progress? In: Proceedings of the 7th IEEE international conference on data science and advanced analytics. 2020, p. 138–47. http://dx.doi.org/10.1109/DSAA49011.2020.00026.

[10] Kotnis B, Nastase V. Analysis of the impact of negative sampling on link prediction in knowledge graphs. 2017, CoRR abs/1708.06816. URL https://dblp.org/rec/journals/corr/abs-1708-06816.

[11] Tsitsulin A, Mottin D, Karras P, Müller E. VERSE: Versatile graph embeddings from similarity measures. In: Proceedings of the 27th international conference on world wide web. 2018, p. 539–48. http://dx.doi.org/10.1145/3178876.3186120.

[12] Song W, Wang S, Yang B, Lu Y, Zhao X, Liu X. Learning node and edge embeddings for signed networks. Neurocomputing 2018;319:42–54. http://dx.doi.org/10.1016/j.neucom.2018.08.072.

[13] Pellegrino MA, Cochez M, Garofalo M, Ristoski P. A configurable evaluation framework for node embedding techniques. In: ESWC satellite events. 2019, p. 156–60. http://dx.doi.org/10.1007/978-3-030-32327-1_31.

[14] Tu C, Yang C, Liu Z, Sun M. Network representation learning: an overview. Sci Sinica Inf 2017;47(8):980–96. http://dx.doi.org/10.1360/N112017-00145.

[15] Goyal P, Ferrara E. GEM: A python package for graph embedding methods. J Open Sour Softw 2018;3(29):876. http://dx.doi.org/10.21105/joss.00876.

[16] Rozemberczki B, Kiss O, Sarkar R. Karate club: An API oriented open-source Python framework for unsupervised learning on graphs. In: Proceedings of the 29th ACM international conference on information & knowledge management. 2020, p. 3125–32. http://dx.doi.org/10.1145/3340531.3412757.

[17] Hu W, Fey M, Zitnik M, Dong Y, Ren H, Liu B, et al. Open graph benchmark: Datasets for machine learning on graphs. 2020, CoRR abs/2005.00687 URL https://dblp.org/rec/journals/corr/abs-2005-00687.

[18] Hagberg AA, Schult DA, Swart PJ. Exploring network structure, dynamics, and function using networkx. In: Proceedings of the 7th Python in science conference. 2008, p. 11–5, URL https://www.osti.gov/biblio/960616.

[19] Lichtenwalter RN, Chawla NV. Link prediction: Fair and effective evaluation. In: Proceedings of the 2012 IEEE/ACM International conference on advances in social networks analysis and mining. 2012, p. 376–83. http://dx.doi.org/10.1109/ASONAM.2012.68.

[20] Fawcett T. ROC graphs: Notes and practical considerations for researchers. Tech. rep., Personal communication; 2004.

[21] Hutson M. Core progress in AI has stalled in some fields. Science 2020;368(6494):927. http://dx.doi.org/10.1126/science.368.6494.927.

[22] Dacrema MF, Cremonesi P, Jannach D. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In: Proceedings of the 13th ACM conference on recommender systems. 2019, p. 101–9. http://dx.doi.org/10.1145/3298689.3347058.

[23] Blalock D, Ortiz JJG, Frankle J, Guttag J. What is the state of neural network pruning? In: Proceedings of the 2nd machine learning and systems conference. 2020, p. 129–46, URL https://proceedings.mlsys.org/paper/2020/hash/d2ddea18f00665ce8623e36bd4e3c7c5-Abstract.html.

[24] Adriaens F, Mara A, Lijffijt J, De Bie T. Block-approximated exponential random graphs. In: Proceedings of the 7th IEEE international conference on data science and advanced analytics. (DSAA), 2020, p. 70–80. http://dx.doi.org/10.1109/DSAA49011.2020.00019.

[25] Petro-de Chalendar KN, Benczúr A. Node embedding algorithms and their evaluation through link prediction. Tech. rep., Eotvos Lorand University; 2020, URL https://web.cs.elte.hu/blobs/diplomamunkak/bsc_matelem/2020/petro_de_chalendar_katalin_nicole.pdf.

[26] Strybol P-P. Using deep learning approaches to unveil key biological processes in human tumours. Tech. rep., Ghent University; 2019, URL https://lib.ugent.be/catalog/rug01:002782844.

[27] Mbungu S, Saerens M. Graph embedding with application to semi-supervised classification, visualization, reconstruction and neighborhood preservation tasks: an experimental comparison. Tech. rep., Ecole polytechnique de Louvain; 2019, URL http://hdl.handle.net/2078.1/thesis:19457.