EvalNE: A Framework for Evaluating Network Embeddings on Link Prediction[*]

Alexandru Mara[†] Jefrey Lijffijt[†] Tijl De Bie[†]

Abstract

Network embedding (NE) methods aim to learn low-dimensional representations of network nodes as vectors, typically in Euclidean space. These representations are then validated on a variety of downstream prediction tasks. Link prediction is one of the most popular choices for assessing the performance of NE methods. However, the complexity of link prediction requires a carefully designed evaluation pipeline in order to provide consistent, reproducible and comparable results. We argue this has not been considered sufficiently in many recent works. The main goal of this paper is to overcome difficulties associated with evaluation pipelines and reproducibility of results. We introduce EvalNE, an evaluation framework to transparently assess and compare the performance of NE methods on link prediction. EvalNE provides automation and abstraction for tasks such as hyper-parameter tuning, model validation, edge sampling, computation of edge embeddings and model validation. The framework integrates efficient procedures for edge and non-edge sampling and can be used to easily evaluate any off-the-shelf embedding method. The framework is freely available as a Python toolbox. Finally, demonstrating the usefulness of EvalNE in practice, we conduct an empirical study in which we try to replicate and analyse experimental sections of several influential papers.

Keywords: network embedding; link prediction; evaluation; edge sampling; negative sampling; graphs

1 Introduction

Link prediction is an important task with applications in a wide range of fields such as computer science, social sciences, biology, and medicine [8, 18, 19, 29]. It amounts to estimating the likelihood for the existence of edges, between pairs of nodes that do not form an edge in the input graph.

Many recent Network Embedding (NE) methods, e.g., [1, 3, 7, 10, 13, 15, 21, 22, 23], have been applied to solving link prediction problems, showing promising results. These methods map nodes in the network to d-dimensional vectors, typically in Euclidean space. The obtained representations are then used as features for a variety of tasks such as visualization, multi-label classification, clustering or link prediction.

The challenges of evaluating NE methods for link prediction Unfortunately, the practical performance of most NE methods is poorly understood and not comparable due to inconsistent evaluation procedures. In this paper, *we focus on a number of difficulties specific to the evaluation of NE methods for link prediction.* Link prediction is particularly challenging as the evaluation involves a number of design choices that can confound the results and are prone to errors.

For example, the implicit assumption is that the input graph is not complete and the purpose is to predict the missing edges as accurately as possible. To evaluate the performance of a NE method for link prediction, one thus needs an (incomplete) training graph along with a (more) complete version of that graph for testing.

Much research has been devoted to determining the best approach to generate these training graphs [8, 18, 29]. Theoretical and empirical evidence suggest that in order to fairly evaluate link prediction methods, snapshots of the networks at different points in time should be used for training and testing. In this way, the methods are tested on natural evolutions of the networks. Nonetheless, the availability of such snapshots is uncommon and raises additional questions, such as how to choose the time intervals for splitting the networks.

For these reasons, in most cases [7, 10, 13, 15] authors resort in their evaluations to sampling sets of edges from the input graphs and using the resulting subgraphs for training. The remaining edges are later used as positive test examples. The process of sampling edges is not standardized and varies in several ways between studies. The relative sizes of the train and test sets, for example, is a user-defined parameter which varies significantly between experimental setups. In [10, 13] the authors use a 50-50 train-test split, in [7] a 60-40, in [15] an 80-20 and in [26] values ranging from 30-70 up to 80-20. Moreover, certain methods have requirements such as connectedness of the training graph [10], while others do not [15].

[*]The research leading to these results has received funding from the ERC under the EU's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement no. 615517, from the FWO (project no. G091017N, G0F9816N), and from the EU's Horizon 2020 research and innovation programme and the FWO under the Marie Sklodowska-Curie Grant Agreement no. 665501.

[†]Ghent University. Email: firstname.lastname(at)ugent.be

A related problem is the fact that, in addition to the 'positive' train and test edges, in most cases also 'negative' train and test edges (also called non-edges) are required. Sometimes these are used also to derive the embedding, while in other cases they are used only to train a classifier that predicts links. These sets of non-edges can be selected according to different strategies [14] and can be of various sizes.

Furthermore, many NE methods only provide node embeddings. From these, edge embeddings need to be derived prior to performing predictions. There are several approaches for deriving edge embeddings [4], and the selection of the edge embedding approach has a strong impact on the performance [10].

Also the metrics used to report the accuracy of different methods vary wildly, from, e.g., AUC-ROC [13] to precision-recall [27] to precision@k [26].

Finally, it appears to be common practice in recent literature to use recommended default settings for existing methods, while tuning the hyper-parameters for the method being introduced. Especially when the recommended default settings were informed by experiments on other graphs than those used in the study at hand, this can paint an unduly unfavorable picture.

Contributions In this paper we propose EvalNE, a framework that simplifies the complex and time consuming process of evaluating NE methods for link prediction. EvalNE automates many parts of the evaluation process: hyper-parameter tuning, selection of train and test edges, negative sampling, and more. The framework: (1) Implements guidelines from research in the area of link prediction evaluation and sampling [8, 18, 29]. (2) Includes (novel) efficient edge and non-edge sampling algorithms. (3) Provides the most widely used edge embedding methods. (4) Evaluates the scalability and accuracy of methods, through wall clock time and a wide range of fixed-threshold metrics and threshold curves. (5) Integrates in a single operation the evaluation of any set of NE methods coded in any language on any number of networks. (6) Finally, EvalNE ensures reproducibility and comparability of evaluations and results through shareable configuration files.

The remainder of this paper is organized as follows. In Section 2 we give a brief overview of related work. Section 3 presents the proposed evaluation framework. Our experiments reproducing the experimental setups of several papers and empirical analysis of the proposed edge set selection strategy are reported in Section 4. Finally Section 5 concludes this paper.

The open-source toolbox is freely available at `https://github.com/Dru-Mara/EvalNE` and the documentation and examples at ReadTheDocs `https://evalne.readthedocs.io/en/latest/index.html`.

2 Related Work

Our work is related to the evaluation of link prediction which has been studied in several recent works, i.e., [8, 18, 20, 29]. These studies focus on the impact of different train set sampling strategies, negative sampling and fair evaluations criteria.

Link prediction as evaluation for the representations learned by a NE algorithm was first introduced in the pioneering work of Grover et al. [10]. The survey work of Zhang et al. [31] points out the importance of link prediction as an application of network representation learning. The authors also signal the inconsistencies in the evaluation of different NE approaches and conduct an empirical analysis of many of these methods on a wide range of datasets. Their empirical study, however, only focuses on vertex classification and clustering. The importance of standard evaluation frameworks as tools to bridge the gap between research and application of NEs is discussed in Hamilton et al. [12].

To the best of our knowledge only two frameworks for the evaluation of NE methods currently exist. `OpenNE` is a recently proposed toolbox for evaluating NE methods on multi-label classification. The toolbox also includes implementations of several state-of-the-art embedding methods. `GEM` [9] is a similar framework which also implements a variety of embedding methods and includes basic evaluation on multi-label classification, vizualization and link prediction tasks. These frameworks, however, are focused on the implementations of embedding methods rather than the evaluation pipeline. Furthermore, these libraries are limited to the NE methods provided by the authors or require new implementations that comply with pre-defined interfaces.

3 EvalNE

In this section we discuss the key aspects of EvalNE. The framework has been designed as a pipeline of interconnected and interchangeable building blocks, as illustrated in Figure 1. These blocks have specific functions such as providing sets of train and test edges or computing edge embeddings from given node embeddings. The modular structure of our framework simplifies code maintenance and the addition of new features, and allows for flexible model evaluation. The framework can be used to evaluate methods providing node embeddings, edge embeddings, or similarity scores (we include in this category the link prediction heuristics). Next, we describe the core functions as well as other building blocks that constitute EvalNE and provide some details regarding the software design.

3.1 Evaluation framework The core building blocks of our framework are the data split and model

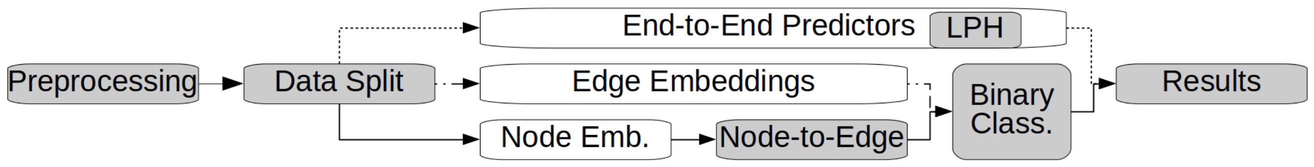

Figure 1: Diagram showing the types of methods which can be evaluated using EvalNE. Gray blocks represent modules provided by the library while white blocks are the user-specified methods to be evaluated. The library allows for the evaluation of end-to-end prediction methods (several LP heuristics are included here as baselines), edge embedding methods, and node embedding methods.

evaluation. These blocks constitute the most basic pipeline for assessing the quality of link predictions. Before describing these, we introduce some notation.

We represent a directed weighted network as $G = (V, E, W)$ with vertex set $V = \{1, \ldots, N\}$, edge set $E \subseteq V \times V$ and weight matrix $W \in \mathbb{R}^{\mathrm{N \times N}}$. Edges are represented as ordered pairs $e = (u, v) \in E$ with weights $w_e \in [0, \infty)$. E_{train} and E_{test} denote the training and testing edge sets. We represent the embedding of network nodes into a d-dimensional space as $\mathbf{X} = (x_1, x_2, \ldots, x_N)$ where $\mathbf{X} \in \mathbb{R}^{\mathrm{N \times d}}$.

3.1.1 Data split As pointed out in Section 1, in order to perform link prediction on a given input graph G, sets of train and test edges are required. The set of train edges is generally required to span all nodes in G and induce a training graph G_{train} with a single connected component, because embeddings of independent components will be far away from and unrelated to each other.

Most studies resort to a naive algorithm to derive a connected G_{train}. The procedure removes edges from an input graph iteratively until the required number of train edges remain. The removed edges are used as test samples. In each iteration, the connectedness of the graph needs to be checked and only remove an edge if it does not cause the graph to become disconnected. This requirement is generally satisfied by running a Breadth First Search (BFS) on the graph after each edge removal, which is a costly operation ($\mathcal{O}(|V| + |E|)$).

Integrated in our evaluation framework, we include a novel algorithm to perform the train-test splits which, as we will show in Section 4, is orders of magnitude faster yet equally simple. Our algorithm, also accounts for the fact that the training graph G_{train} must span all nodes in G and contain a single connected component.

Given as input an undirected graph $G = (V, E, W)$ and a target number of train edges $|E_{train}|$, the proposed algorithm to split the train and test edges proceeds as follows:

1. Obtain a uniform spanning tree ST of G
2. Initialize the set of training edges E_{train} to all edges in ST
3. Select edges uniformly at random without replacement from the remaining edges $E \setminus E_{train}$.

We select a spanning tree uniformly at random from the set of all possible ones using Broder's algorithm [2]:

1. Select a random vertex s of G and start a random walk on the graph until every vertex is visited. For each vertex $i \in V \setminus \{s\}$ collect the edge $e = (j, i)$ that corresponds to the first entrance to vertex i. Let T be this collection of edges.
2. Output the set T.

The complexity of the uniform spanning tree generation methods is $\mathcal{O}(n \log n)$ on expectation [2]. The same bound applies for the proposed train-test split algorithm as the addition of random edges to the train set can be efficiently done in $\mathcal{O}(|E_{train}|)$ time. A more efficient algorithm for constructing a uniform spanning tree of a given graph exists. This method, named Wilson's algorithm [28], will be included in future versions of EvalNE.

For directed graphs G we first construct an equivalent undirected version G^* by adding reciprocals for every edge in E. We then run the same algorithm described above on G^* and include in the train set only those edges present in the initial directed graph G. This method results in a weakly connected training graph spanning the same nodes as the original G.

In addition to train and test edges, sets of train and test *non-edges* (also referred to as *negative samples*) are required in order to evaluate link prediction. These are edges between pairs of vertices u, v such that $e = (u, v) \notin E$. The proposed toolbox can compute these non-edges according to either the *open world* or the *closed world* assumption. The two strategies only differ in the selection of the train non-edges. Under the open world assumption, train non-edges are selected such that they are not in E_{train}. Thus, this strategy allows overlapping between train non-edges and test real edges.

Under the closed world assumption, we consider the non-edges to be known *a priori* and therefore select them so that they are neither in E_{train} nor in E_{test}.

The number of train and test edges and non-edges are user-controlled parameters. The minimum number of train edges is determined by the size of the spanning tree and the combined sets of non-edges have a trivial upper limit of $(N \cdot N) - |E|$. However, for the train set size, fractions between 50% and 90% of total edges E are recommended. For values below 50%, the resulting training graph will often not preserve the properties of the original graph G as shown in [16].

3.1.2 Evaluation

The proposed framework can evaluate the scalability, parameter sensitivity and accuracy of embedding methods. We assess the scalability directly by measuring the wall clock time. The performance of the methods can be easily reported for different values of embedding dimensionality and hyper-parameter values. Finally, the link prediction accuracy is reported using two types of metrics: fixed-threshold metrics and threshold curves. Fixed-threshold metrics summarize the performance of the methods to single values. The framework provides the following measures: confusion matrix (TP, FN, FP, TN), precision, recall, fallout, miss, accuracy, F-score and AUC-ROC. Threshold curves present the performance of the methods for a range of threshold values between 0 and 1. EvalNE includes precision-recall curves [18] and receiver operating curves [6]. The framework also recommends the most suitable metrics based on the evaluation setup.

3.2 Baselines and building blocks

In addition to the core building blocks, and in order to extend the evaluation process to different types of embedding methods, our framework also provides data preprocessing, data manipulation, and node to edge embedding functions. Moreover, EvalNE integrates a standard binary classification technique which can be used, for example, to obtain link predictions from edge embeddings. Furthermore, the toolbox contains a series of built-in link prediction heuristics which can be used as baselines.

3.2.1 Preprocessing

The toolbox offers a variety of functions to load, store, and manipulate networks. These include methods to prune nodes based on degree, remove self-loops, relabel nodes, obtain sets of specific types of edges, restrict networks to their main connected components and obtain common network statistics. The preprocessing functions of EvalNE build on top of and can be used in combination with those provided by other mainstream software packages (e.g. NetworkX [11]).

3.2.2 Link prediction heuristics

A training graph $G_{train} = (V, E_{train}, W)$ spanning the same set of vertices as the initial graph G but containing only the edges in the train set is provided as input to the link prediction heuristics. Depending on the input graph type we use one of the following definitions. For undirected graphs we use the ones presented below, where $\Gamma(u)$ denotes the neighbourhood of a node u. For directed graphs we restrict our analysis to either the in-neighbourhood ($\Gamma_i(u)$) or the out-neighbourhood ($\Gamma_o(u)$) of the nodes.

Common neighbours (CN):

$$\mathrm{CN}(u,v) = |\Gamma(u) \cap \Gamma(v)|$$

Jaccard coefficient (JC):

$$\mathrm{JC}(u,v) = \frac{|\Gamma(u) \cap \Gamma(v)|}{|\Gamma(u) \cup \Gamma(v)|}$$

Adamic-Adar index (AA):

$$\mathrm{AA}(u,v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(w)|}$$

Resource allocation index (RAI):

$$\mathrm{RAI}(u,v) = \sum_{w \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(w)|}$$

Preferential attachment (PA):

$$\mathrm{PA}(u,v) = |\Gamma(u)| \cdot |\Gamma(v)|$$

Katz:

$$\mathrm{Katz}(u,v) = \sum_{l=1}^{\infty} \beta^l \cdot |\rho(u,v|l)|$$

For Katz, the parameter $\beta \in [0,1]$ is a damping factor and $|\rho(u,v|l)|$ represents the number of paths of length l from node u to node v.

In addition to these methods, a random prediction model is provided for reference. This method simply outputs a uniform random value in $[0,1]$ as the likelihood of a link between any pair of given vertices (u,v).

3.2.3 Node to edge embeddings

Unlike for the abovementioned LP heuristics where the output can be directly used for link prediction, for NE methods this is generally not the case. Most implementations of NE methods produce only node embeddings. Thus, an additional step of learning edge embeddings is required in order to predict links via binary classification.

The edge representations are typically derived in an unsupervised fashion. For example, to derive the edge embedding $y_{(u,v)}$ for edge $e = (u, v)$, we apply a binary operator $\circ$ over the embeddings of nodes u and v, i.e. x_u and x_v respectively:

$$y_{(u,v)} = x_u \circ x_v.$$

In [10] the authors propose the following alternatives for the operator $\circ$ which we include in our evaluation framework. Any other user-defined operators can be also easily integrated.

Average (Avg.):

$$x_u \oplus x_v \equiv \frac{x_{u,i} + x_{v,i}}{2}$$

Hadamard (Had.):

$$x_u \odot x_v \equiv x_{u,i} * x_{v,i}$$

Weighted L_1:

$$||x_u \cdot x_v||_{\bar{1}} \equiv |x_{u,i} - x_{v,i}|$$

Weighted L_2:

$$||x_u \cdot x_v||_{\bar{2}} \equiv |x_{u,i} - x_{v,i}|^2$$

3.2.4 Binary classification Most NE methods (e.g., [7, 10, 13, 15]) rely on a logistic regression classifier to predict link probabilities from edge embeddings. In EvalNE we include logistic regression with 10-fold cross validation. The framework, however, is flexible and allows for any binary classifier to be used.

3.3 Software design The EvalNE framework is provided as a Python toolbox compatible with Python2 and Python3 and which can be easily installed and used on Linux, MacOS, and Microsoft Windows. The toolbox depends only on a small number of popular open-source Python packages and the coding style and code documentation comply with the PEP8 and numpy docstring standards, respectively. The documentation provided includes instructions on the installation and use as well as examples of high-level and low-level use and integration with existing code.

EvalNE can be used both as a command line tool and an API. As a command line tool, the framework requires as input a configuration file defining a complete experimental setup, from methods to evaluate to networks, edge splits and parameters to be tuned. For convenience, several pre-filled configuration files which reproduce the experimental sections of influential papers on NE are provided as examples. Users can create new files defining their own evaluations (e.g. by adding new methods or different networks) and share them in order to ensure the reproducibility of results.

When used as an API, the framework exposes a modular design with blocks providing independent and self contained functionalities which can be easily exploited. The user interacts with the library trough an evaluator object which integrates all the building blocks and orchestrates the evaluation pipelines. As an API EvalNE provides users with access to more advanced capabilities and more flexible evaluation workflows.

4 Experiments

In this section we seek to prove the simplicity and usefulness of our evaluation framework. To this end we have selected four papers from the NE literature, i.e. Node2vec, PRUNE, CNE and SDNE and replicated their experimental setups. We have also conducted a series of experiments in which we compare our edge set selection algorithm to the naive approach. We performed these comparisons in terms of both scalability and link prediction accuracy. All experiments were run on a single machine equiped with a Intel® Core$^{\text{TM}}$ i5 processor and 16GB of RAM.

4.1 Experimental setups Next, we present the main characteristics of the experimental settings we have replicated and some important remarks:

Node2vec [10]: In this work the authors evaluate the following embedding methods and link prediction heuristics: Node2vec, DeepWalk [22], LINE [23], Spectral Clustering [24], CN, JC, AA, and PA. Experiments are performed on the Facebook [17], PPI [10], and AstroPh[17] datasets with 50-50 train-test splits. The same number of edges and non-edges are used throughout the experiment. The results are reported in terms of AUC-ROC and the edge embedding methods used are average, Hadamard, weighted L_1 and weighted L_2.

PRUNE [15]: This experimental setup includes Node2vec, DeepWalk, LINE, SDNE, PRUNE and NRCL [27]. The networks used are HepPH [17], FB-wallpost [25] and Webspam and the train-test fraction is 0.8 with similar sizes of the non-edge sets. The node to edge embedding operator used is not reported by the authors and the results are presented in terms of AUC-ROC. In this setting, the network used are directed.

CNE [13]: The evaluation setup in this case is similar to that of Node2vec with the following differences: CNE and Metapath2vec [5] methods are also evaluated; BlogCatalog [30], Wikipedia [17] and StudentDB are included in the list of evaluated networks and the results are only reported for the Hadamard operator. It should be noted that two authors of this paper are also authors

Dataset	Category	$\|V\|$	$\|E\|$
Facebook	Social	4039	88234
FB-wallpost	Social	43953	262631
BlogCatalog	Social	10312	333983
StudentDB	Social	395	3423
HepPh	Citation	34546	421578
AstroPh	Collaboration	17903	196972
GR-QC	Collaboration	4158	26844
PPI	Biological	3852	37841
Wikipedia	Language	4777	92295

Table 1: All the datasets used for evaluation.

of CNE, and to some degree the development of CNE inspired this work. However, the code used for evaluation in that paper was completely separate.

SDNE [26]: The authors report in this case experiments using the following methods: DeepWalk, LINE, SDNE, GraRep [3], LapEig [1] and CN. The link prediction experiments are performed on the GR-QC dataset [17] with a 0.85 train-test fraction. The results are reported only for the Hadamard operator and in terms of precision@k for a collection of values between 2 and 10000. In this setup, the number of train non-edges is the same as that of train edges while all the remaining non-endges in the graph are used for testing.

In our experiments we have replicated these setting with the following exceptions (1) for node2vec we did not include spectral clustering and (2) for PRUNE we could not obtain NRCL and lacked the computational resources to evaluate the NE methods on the Webspam dataset. In all cases we report the same metrics as the original authors and average the results over five independent executions. We used the open-world assumption for the non-edges, logistic regression for binary classification and tuned the same hyper-parameters. All experiments were run using specific configuration files created for each setting which will be made public.

Regarding the implementations, we evaluated the LP heuristics included in EvalNE; original code by the authors for Deepwalk[1] , Node2vec[2], LINE[3], PRUNE[4], CNE[5], and Metapath2vec[6], and for the remaining ones, the implementations in the `OpenNE`[7] library. Table 1 contains the main characteristics of all the networks used. Network sizes are those corresponding to the main (weakly) connected components of each graph.

[1]`https://github.com/phanein/deepwalk`

[2]`https://github.com/aditya-grover/node2vec`

[3]`https://github.com/tangjianpku/LINE`

[4]`https://github.com/ntumslab/PRUNE`

[5]`https://bitbucket.org/ghentdatascience/cne/`

[6]`https://ericdongyx.github.io/metapath2vec/m2v.html`

[7]`https://github.com/thunlp/OpenNE`

	Facebook	PPI	AstroPh
CN	0.0226	0.0031	0.0189
JC	0.0134	0.0066	0.0178
AA	0.0245	0.0031	0.0189
PA	0.0242	0.0321	0.0393

Table 2: Absolute difference in recall between results obtained using the naive and proposed edge split.

4.2 Evaluation results For each of the experimental settings reproduced we present a table containing the original values reported in the paper, and in parentheses the difference between our result and these values (Tables 3–6). Positive values in parentheses, thus, indicate that our results are higher than the ones reported in the original papers by that margin, while negative values indicate the opposite.

The results obtained show that our experiments align fairly well with those reported in the CNE and PRUNE papers. The only exception is the result of Metapath2vec on the Facebook dataset, which substantially differs from the CNE paper. For Node2vec and SDNE, the differences are larger and occasionally severe (differences over 0.15 are marked in bold in all tables). Possible explanations for these inconsistencies are the use of different implementations of NE methods, different not reported default parameters or the use of parallelization. In addition, we have also observed important differences when computing the AUC-ROC directly from class labels or from class probabilities. In order to reproduce the CNE experiments we used class probabilities, while for Node2vec we used class labels as seem to have been done by the original authors.

These results illustrate the need to: (a) create reproducible pipelines for experiments, and (b) report specifics about the parameter settings and precise implementations used in the evaluation.

4.3 Edge sampling We compared the proposed edge and non-edge sampling strategy in terms of accuracy and scalability to the naive approach. For the accuracy experiment we selected the Facebook, PPI, and AstroPh datasets, and performed link predictions with the CN, JC, AA, and PA heuristics. We collected all the metrics available in EvalNE and computed for each the absolute difference between the naive and proposed approach. The recall presents the highest deviation, so we show these results in Table 2. Although recall differs up to 0.0393, in this case it is traded off with precision. The accuracy in this case differs only by 0.0154. For the AUC-ROC, one of the most widely used metrics, the maximum deviation is of 0.015 reached on the AstroPh dataset by the PA heuristic. Across methods and

		Facebook	PPI	AstroPh
	CN	0.81 (+0.14)	0.71 (+0.06)	0.82 (+0.13)
	JC	0.88 (+0.04)	0.70 (+0.04)	0.81 (+0.12)
	AA	0.83 (+0.13)	0.71 (+0.06)	0.83 (+0.12)
	PA	0.71 (+0.04)	0.67 (+0.13)	0.70 (+0.08)
Avg.	DeepWalk	0.72 (−0.01)	0.69 (+0.08)	0.71 (+0.01)
	LINE	0.70 (−0.03)	0.63 (+0.12)	0.65 (+0.14)
	node2vec	0.73 (−0.01)	0.75 (−0.01)	0.72 (−0.02)
Had.	DeepWalk	0.97 (−0.03)	**0.74 (−0.2)**	0.93 (−0.12)
	LINE	0.95 (−0.06)	0.72 (−0.01)	0.89 (+0.05)
	node2vec	0.97 (0.0)	**0.77 (−0.17)**	0.94 (−0.05)
W L_1	DeepWalk	0.96 (−0.01)	0.60 (+0.14)	0.83 (+0.09)
	LINE	**0.95 (−0.33)**	0.70 (−0.01)	**0.88 (−0.28)**
	node2vec	0.96 (−0.01)	0.63 (−0.03)	0.85 (+0.03)
W L_2	DeepWalk	0.96 (−0.01)	0.61 (+0.14)	0.83 (+0.09)
	LINE	**0.95 (−0.33)**	0.71 (−0.02)	**0.89 (−0.27)**
	node2vec	0.96 (0.0)	0.62 (−0.02)	0.85 (+0.03)

Table 3: Reported AUC-ROC values in the node2vec paper and difference to our reproduced values.

	DeepWalk	LINE	Node2vec	SDNE	PRUNE
HepPh	0.80 (−0.05)	0.80 (−0.09)	0.81 (−0.05)	0.75 (−0.04)	0.86 (−0.03)
FB-wallpost	0.83 (+0.01)	0.78 (+0.09)	0.85 (−0.09)	0.86 (−0.02)	0.88 (−0.01)

Table 4: Reported AUC-ROC values in the PRUNE paper and difference to our reproduced values.

	Facebook	PPI	AstroPh	BlogCatalog	Wikipedia	StudentBD
CN	0.97 (+0.01)	0.77 (0.0)	0.94 (+0.01)	0.92 (+0.01)	0.84 (0.0)	0.42 (−0.01)
JS	0.97 (+0.01)	0.76 (0.0)	0.94 (+0.01)	0.78 (0.0)	0.50 (−0.01)	0.42 (−0.01)
AA	0.98 (0.0)	0.77 (+0.01)	0.94 (+0.01)	0.93 (0.0)	0.86 (+0.01)	0.42 (−0.01)
PA	0.83 (+0.01)	0.89 (+0.01)	0.86 (+0.01)	0.95 (0.0)	0.91 (+0.01)	0.91 (+0.01)
DeepWalk	0.98 (0.0)	0.64 (+0.01)	0.92 (+0.01)	0.61 (−0.01)	0.56 (0.0)	0.76 (+0.03)
LINE	0.95 (0.0)	0.75 (+0.01)	0.98 (0.0)	0.76 (+0.01)	0.71 (0.0)	0.86 (−0.02)
Node2vec	0.99 (0.0)	0.68 (+0.02)	0.97 (0.0)	0.73 (−0.05)	0.67 (−0.08)	0.83 (0.0)
Metapath	**0.74 (+0.17)**	0.85 (0.0)	0.83 (+0.02)	0.91 (0.0)	0.83 (+0.02)	0.92 (−0.01)
CNE(unif.)	0.99 (0.0)	0.89 (+0.01)	0.99 (0.0)	0.92 (+0.01)	0.84 (0.0)	0.93 (0.0)
CNE(deg.)	0.99 (0.0)	0.91 (0.0)	0.99 (0.0)	0.96 (0.0)	0.92 (−0.01)	0.94 (0.0)

Table 5: Reported AUC-ROC values in the CNE paper and difference to our reproduced values.

	prec@100	prec@200	prec@300	prec@500	prec@800	prec@1000	prec@10000
SDNE	1 (0.0)	1 (0.0)	1 (+0.01)	0.99 (0.0)	0.97 (+0.03)	0.91 (+0.09)	0.25 (−0.12)
LINE	1 (0.0)	1 (0.0)	0.99 (+0.01)	0.93 (+0.06)	0.74 (+0.26)	0.79 (+0.11)	0.21 (−0.13)
DeepWalk	**0.6 (+0.40)**	**0.55 (+0.45)**	**0.44 (+0.56)**	**0.34 (+0.65)**	**0.29 (+0.70)**	**0.29 (+0.71)**	0.15 (+0.01)
GraRep	**0.04 (+0.96)**	**0.03 (+0.97)**	**0.03 (+0.97)**	**0.04 (+0.96)**	**0.03 (+0.97)**	**0.03 (+0.97)**	**0.19 (+0.16)**
CN	1 (0.0)	0.96 (+0.03)	0.96 (+0.03)	0.98 (0.0)	0.87 (+0.05)	0.79 (+0.09)	0.19 (0.0)
LapEig	0.93 (+0.07)	0.85 (+0.15)	**0.82 (+0.17)**	**0.66 (+0.34)**	**0.46 (+0.53)**	**0.39 (+0.48)**	0.05 (+0.15)

Table 6: Reported precision@k values in the SDNE paper and difference to our reproduced values.

datasets, the results obtained with the naive approach were slightly higher than those using our method.

Regarding the scalability experiments, in Figure 2 we summarize the execution time in seconds of both methods on four different datasets. This experiment shows that the proposed method is several order of magnitude faster than the naive approach and independent on the number of train and test edges required. The execution time of the naive approach decreases, as expected, as the number of test edges requested becomes

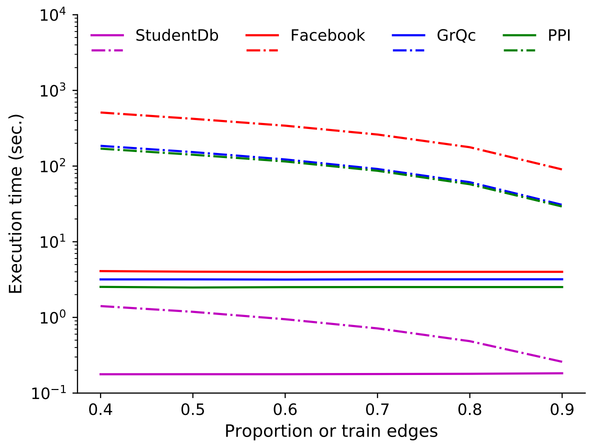

Figure 2: Execution times of proposed and naive edge split methods w.r.t. proportion of train edges. Solid lines correspond to the proposed sampling method, while dashed lines correspond to the typically used (naive) sampling method.

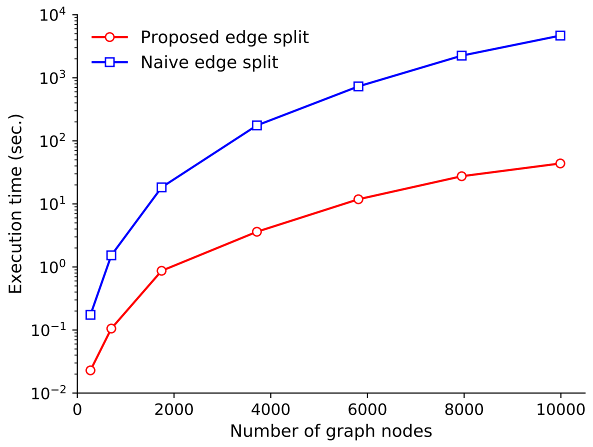

Figure 3: Execution times of proposed and naive edge split methods on sub-graphs of BlogCatalog of increasing sizes. The results are reported for a 50-50 train-test split.

smaller. In Figure 3 we select the BlogCatalog dataset and restrict it to sub-graphs of different number of nodes ranging from 400 up to 10000. The execution times using both sampling strategies are reported in Figure 3.

For our proposed strategy we have also evaluated the variance observed in the performance metrics for different number of repetitions of the experiments (or equivalently, the effect of averaging the results over different number of edge splits). In this case we used the same datasets, NE methods and LP heuristics as in the node2vec experiment and 50-50 and 90-10 train-test splits. We compared the results obtained in a single run with those averaged over five independent runs for both split fractions. For the 50-50 split the average difference observed over all methods, datasets and metrics is $3.4 \cdot 10^{-3}$ with a variance of $1.8 \cdot 10^{-5}$ and a maximum difference of 0.0293. For the 90-10 split, the average difference is $3.0 \cdot 10^{-3}$ the variance of $1.0 \cdot 10^{-5}$ and maximum difference 0.0186. These results indicate that a single train and test split already gives a sufficiently good estimate of the generalization error of the models. Thus, experiment repeats are not necessary for networks of similar sizes. These observations seem to hold for different train-test split fractions.

Finally, we study the effect of the train-test split sizes on the method results. We use an indentical setting as described above and compare the absolute difference of the results obtained using a 50-50 split with those obtained using a 90-10. The average difference across all methods, datasets and metrics is 0.038, the variance 0.0016 and maximum difference 0.2429. This indicate that, in general, the train-test split size has indeed a strong impact on the method results.

5 Conclusions

The recent surge of research in the area of network embeddings has resulted in a wide variety of data sets, metrics, and setups for evaluating and comparing the utility of embedding methods. Comparability across studies is lacking and not all evaluations are equally sound. This highlights the need for specific tools and pipelines to ensure the correct evaluation of these methods. Particularly, the use of representation learning for link prediction tasks requires train and test sampling, non-edge sampling, and in many cases selection of edge embedding methods and binary classifiers. The evaluation procedure, thus, becomes an ensemble of tasks which allow for many errors or inconsistencies.

In this work we have proposed EvalNE, a novel framework that can be used to evaluate any network embedding method for link prediction. Our pipeline automates the selection of train and test edge sets, simplifies the process of tuning model parameters and reports the accuracy of the methods according to many criteria. Our experiments highlight the importance of the edge sampling strategy and parameter tuning for evaluating NE methods. We have also introduced a scalable procedure to sample edge sets from given networks and showed empirically that is orders or magnitude faster than the naive approach that has been used in recent literature.

References

[1] M. Belkin and P. Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, in Proc. of NIPS, 2002, pp. 585–591.

[2] A. Broder, *Generating random spanning trees*, in Proc. of FOCS, 1989, pp. 442–447.

[3] S. Cao, W. Lu, and Q. Xu, *GraRep: Learning graph representations with global structural information*, in Proc. of CIKM, 2015, pp. 891–900.

[4] H. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena, *A Tutorial on Network Embeddings.* arXiv:1808.02590, 2018.

[5] Y. Dong, N. V. Chawla, and A. Swami, *Metapath2vec: Scalable representation learning for heterogeneous networks*, in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, New York, NY, USA, 2017, ACM, pp. 135–144.

[6] T. Fawcett, *ROC graphs: Notes and practical considerations for researchers*, tech. rep., 2004.

[7] M. Gao, L. Chen, X. He, and A. Zhou, *Bine: Bipartite network embedding*, in Proc. of SIGIR, 2018, pp. 715–724.

[8] D. Garcia-Gasulla, C. U. Cortes, E. Ayguade, and J. J. Labarta, *Evaluating link prediction on large graphs*, in Proc. of CAAI, 2015, pp. 90–99.

[9] P. Goyal and E. Ferrara, *Gem: A python package for graph embedding methods*, JOSS, 3 (2018), p. 876.

[10] A. Grover and J. Leskovec, *node2vec: Scalable feature learning for networks*, in Proc. of KDD, 2016, pp. 855–864.

[11] A. A. Hagberg, D. A. Schult, and P. J. Swart, *Exploring network structure, dynamics, and function using networkx*, in Proceedings of the 7th Python in Science Conference, G. Varoquaux, T. Vaught, and J. Millman, eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[12] W. L. Hamilton, R. Ying, and J. Leskovec, *Representation learning on graphs: Methods and applications.* arXiv:1709.05584, 2017.

[13] B. Kang, J. Lijffijt, and T. De Bie, *Conditional Network Embeddings.* arXiv:1805.07544, 2018.

[14] B. Kotnis and V. Nastase, *Analysis of the impact of negative sampling on link prediction in knowledge graphs*, CoRR, abs/1708.06816 (2017).

[15] Y.-A. Lai, C.-C. Hsu, W. H. Chen, M.-Y. Yeh, and S.-D. Lin, *Prune: Preserving proximity and global ranking for network embedding*, in Proc. of NIPS, 2017, pp. 5257–5266.

[16] J. Leskovec and C. Faloutsos, *Sampling from large graphs*, in Proc. of KDD, 2006, pp. 631–636.

[17] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection.* `http://snap.stanford.edu/data`, June 2014.

[18] R. N. Lichtenwalter and N. V. Chawla, *Link prediction: Fair and effective evaluation*, in Proc. of ASONAM, 2012, pp. 376–383.

[19] L. Lü and T. Zhou, *Link prediction in complex networks: A survey*, Physica A, 390 (2011), pp. 1150–1170.

[20] V. Martínez, F. Berzal, and J.-C. Cubero, *A survey of link prediction in complex networks*, ACM Comp. Surv., 49 (2016).

[21] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, *Asymmetric transitivity preserving graph embedding*, in Proc. of KDD, 2016, pp. 1105–1114.

[22] B. Perozzi, R. Al-Rfou, and S. Skiena, *Deepwalk: Online learning of social representations*, in Proc. of KDD, 2014, pp. 701–710.

[23] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, *LINE: Large-scale information network embedding*, in Proc. of WWW, 2015, pp. 1067–1077.

[24] L. Tang and H. Liu, *Leveraging social media networks for classification*, Data Mining and Knowledge Discovery, 23 (2011), pp. 447–478.

[25] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, *On the evolution of user interaction in facebook*, in Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09), August 2009.

[26] D. Wang, P. Cui, and W. Zhu, *Structural deep network embedding*, in Proc. of KDD, 2016, pp. 1225–1234.

[27] X. Wei, L. Xu, B. Cao, and P. S. Yu, *Cross view link prediction by learning noise-resilient representation consensus*, in Proc. of WWW, 2017, pp. 1611–1619.

[28] D. B. Wilson, *Generating random spanning trees more quickly than the cover time*, in Proc. of STOC, 1996, pp. 296–303.

[29] Y. Yang, R. N. Lichtenwalter, and N. V. Chawla, *Evaluating link prediction methods*, KAIS, 45 (2015), pp. 751–782.

[30] R. Zafarani and H. Liu, *Social computing data repository at ASU*, 2009.

[31] D. Zhang, J. Yin, X. Zhu, and C. Zhang, *Network representation learning: A survey.* arXiv:1801.05852, 2018.