EvalNE: A Framework for Evaluating Network Embeddings on Link Prediction

Alexandru Mara, Jefrey Lijffijt, and Tijl De Bie
Ghent University, Belgium
Outline

1. Motivation
2. Objectives
3. Network Embedding (NE)
4. Link Prediction (LP)
5. Evaluating NE on LP
6. EvalNE frameworks
7. Experiments
Motivation

Difficulty of comparing new network embedding methods against the sota.

- Non-standard evaluations
  - Networks
  - Methods
  - Method implementations
  - Hyperparameter tuning
  - Evaluation metrics
  
  *incomparable results*

- LP a complex task
  
  *evaluation prone to errors, many evaluation choices*

- Current NE frameworks
  - OpenNE
  - GEM
  
  *limited LP evaluation capabilities, very restricted*
Objectives

- Address the reproducibility crisis in the field of Network Embedding (NE) for Link Prediction (LP)
- Simplify evaluation of NE methods and comparison with sota
- Provide a unified benchmarking framework
  - Flexible enough to adapt to existing evaluation settings
  - Flexible to incorporate any method and data
  - Minimize the likelihood of evaluation errors
  - With justified recommendations of the most adequate evaluation pipelines
Network Embedding (NE)

- A mapping of network nodes to d-dimensional vector representations
- The representation learned can be used as features for a variety of standard ML tasks (e.g. clustering, classification, etc.)
- Constitute a way of bringing all the power of standard ML to graphs
- Node embeddings and/or edge embeddings
Network Embedding (NE)

- Formally, a network embedding is a mapping $\Phi : V \rightarrow \mathbb{R}^{|V| \times d}$ where $d << |V|$. This mapping $\Phi$ defines the latent representation (or embedding) of each node $v \in V$.
- Categories of NE methods
  - Matrix factorization (e.g. LapEig, MatFact)
  - Random walks (e.g. DeepWalk, Node2vec)
  - Deep learning (e.g. SDNE, BINE)
- Learning embeddings:
  1. Proximity measure defined on the graph
  2. Similarity in the embedding space
  3. Cost function
Network Embedding Evaluation

The quality of the embeddings provided by NE methods is generally assessed through the following tasks:

- Multi-label classification
- Clustering
- Visualization
- Link prediction
Network Embedding Evaluation

The quality of the embeddings provided by NE methods is generally assessed through the following tasks:

- Multi-label classification
- Clustering
- Visualization
- Link prediction

Only node embedding, embed complete network and evaluate

Edge embedding, evaluation requires embedding of a subgraph of the original network or snapshot of the network in time
Link Prediction (LP)

- Estimate the likelihood of the existence of edges between pairs of nodes
- Binary classification with positive and negative examples (both true edges and non-edges required for evaluation)
  - Split the network edges in a set of train edges and a set of test edges (snapshots of the network in time can be used for train/test)
  - Generate sets of false edges or non-edges
  - Train the binary classifier with a set of train edges and train non-edges
  - Evaluate performance on the test edges
Evaluating NE methods on LP
Evaluating NE methods on LP
Evaluating NE methods on LP
Evaluating NE methods on LP

Node Embeddings → Edge Embeddings → Binary classification

Original network → Train edges → Test edges → Train non-edges → Test non-edges
Evaluating NE methods on LP

Node Embeddings → Edge Embeddings → Binary classification → Link predictions
Evaluating NE methods on LP

Evaluation choices:

- Network preprocessing
  - Restrict graph to main cc
  - Relabel nodes
- Train/test fraction
  - Common values 30-90
- Non-edge sampling
  - Open-world
  - Closed-world
- Train/Test edge selection
  - Naive slow approaches
Evaluating NE methods on LP

Evaluation choices:

- Node to edge embedding
- LP heuristics
- Binary classifiers
- Evaluation metrics
  - Commonly AUROC, prec@k, prec-recall

Average (Avg.):
\[ x_u \oplus x_v \equiv \frac{x_{u,i} + x_{v,i}}{2} \]

Hadamard (Had.):
\[ x_u \odot x_v \equiv x_{u,i} \times x_{v,i} \]

Weighted \( L_1 \):
\[ ||x_u \cdot x_v||_1 \equiv |x_{u,i} - x_{v,i}| \]

Weighted \( L_2 \):
\[ ||x_u \cdot x_v||_2 \equiv |x_{u,i} - x_{v,i}|^2 \]
EvalNE

- CLI tool and API
- Open-source (https://github.com/Dru-Mara/EvalNE)
- Cross-platform
- Complete documentation (https://evalne.readthedocs.io/en/latest/)
- Easy to use (no coding required)
Main Features

- Highly **flexible evaluation** pipelines (described in conf. files)
- Automated method evaluation
- Automated **hyper-parameter tuning**
- Simple addition of new methods
- Language-independent evaluation
- **Efficient train/test edge split algorithm**
- Many evaluation criteria
- Main node-to-edge embedding methods

---

**Alg. 1: Train/Test edge selection**

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 \( T \).
Toolbox Use

Through CLI:

- Fill configuration file
- Run:
  - `foo@bar:$ python evalne conf.ini`

```
[GENERAL]
EDGE_EMBEDDING_METHODS = average hadamard
LP_MODEL = LogisticRegression
EXP_REPEATS = 10
EMBED_DIM = 128
VERBOSE = True

[NETWORKS]
NAME = Facebook PPI ArXiv
INPATHS = ../data/Facebook/facebook_combined.txt
        ../data/ArXiv/CA-AstroPh.txt
OUTPATHS = ../output/Facebook/
          ../output/ArXiv/
DIRECTED = False False False
SEPARATORS = \s\t\
COMMENTS = #

[PREPROCESSING]
RELABEL = True
DEL_SELFLOOPS = True
PREP_NN_NAME = prep_graph.edgelist
WRITE_STATS = True
DELIMITER = ','

[TRAINTEST]
TRAIN_FRAC = 0.5
FAST_SPLIT = True
OWA = True
NUM_FE_TRAIN = None
NUM_FE_TEST = None
TRAINTEST_PATH = train_test_splits/

[REPORT]
MAXIMIZE = aucroc
SCORES = % (maximize)
CURVES = roc
PRECATK_VALS = 2 10 100 200 500 1000
```
Toolbox Use

Through CLI:

[BASILINES]
LP_BASELINES = common_neighbours
   jaccard_coefficient
   adamic_adar_index
   preferential_attachment
NEIGHBOURHOOD = in out

[OPENNE METHODS]
NAMES_OPNE = node2vec deepWalk line
METHODS_OPNE = python -m openne --method node2vec --epochs 100
   python -m openne --method deepWalk --epochs 100
   python -m openne --method line --epochs 100
TUNE_PARAMS_OPNE = --p 0.25 0.5 1 2 4 --q 0.25 0.5 1 2 4

[OTHER METHODS]
NAMES_OTHER = prune
EMBTYPE_OTHER = ne
METHODS_OTHER = python ../methods/PRUNE/src/main.py --inputgraph {} --output {} --dimension {} #
   ../methods/metapath2vec/metapath2vec -train {} -output {} -size {}
TUNE_PARAMS_OTHER = -negative 1 5 10
INPUT_DELIM_OTHER = \n
OUTPUT_DELIM_OTHER = ,
Toolbox Use

As an API:

```python
from evalne.evaluation import evaluator
from evalne.preprocessing import preprocess as pp

# Load and preprocess the network
G = pp.load_graph('./evalne/tests/data/network.edgelist')
G, _ = pp.prep_graph(G)

# Create an evaluator and generate train/test edge split
nee = evaluator.Evaluator()
_, = nee.train_test_split.compute_splits(G)

# Set the baselines
methods = ['random_prediction', 'common_neighbours', 'jaccard_coefficient']

# Evaluate baselines
nee.evaluate_baseline(methods=methods)
```
Toolbox Use

```python
try:
    # Check if OpenNE is installed
    import openne

    # Set embedding methods from OpenNE
    methods = ['node2vec', 'deepwalk', 'GraRep']
    commands = [
        'python -m openne --method node2vec --graph-format edgelist --p 1 --q 1',
        'python -m openne --method deepWalk --graph-format edgelist --number-walks 40',
        'python -m openne --method grarep --graph-format edgelist --epochs 10']
    edge_emb = ['average', 'hadamard']

    # Evaluate embedding methods
    for i in range(len(methods)):
        command = commands[i] + ' --input {} --output {} --representation-size {}'
        nee.evaluate_cmd(method_name=methods[i], method_type='ne', command=command,
                           edge_embedding_methods=edge_emb, input_delim=' ', output_delim=' ')

    # Get output
    results = nee.get_results()
    for result in results:
        result.pretty_print()
```
Experimental Results

Replicating the Node2vec [1] LP evaluation:

- Table presents original values for LP.
- In parenthesis (our - original) results.

Issues:

- Missing details in experimental setup
- Class probabilities vs class labels
- Method implementations used

<table>
<thead>
<tr>
<th></th>
<th>Facebook</th>
<th>PPI</th>
<th>arXiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>0.81 (+0.14)</td>
<td>0.71 (+0.06)</td>
<td>0.82 (+0.13)</td>
</tr>
<tr>
<td>JC</td>
<td>0.88 (+0.04)</td>
<td>0.70 (+0.04)</td>
<td>0.81 (+0.12)</td>
</tr>
<tr>
<td>AA</td>
<td>0.83 (+0.13)</td>
<td>0.71 (+0.06)</td>
<td>0.83 (+0.12)</td>
</tr>
<tr>
<td>PA</td>
<td>0.71 (+0.04)</td>
<td>0.67 (+0.13)</td>
<td>0.70 (+0.08)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Avg.</th>
<th>DeepWalk</th>
<th>LINE</th>
<th>node2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.72 (-0.01)</td>
<td>0.69 (+0.08)</td>
<td>0.71 (+0.01)</td>
</tr>
<tr>
<td></td>
<td>0.70 (-0.03)</td>
<td>0.63 (+0.12)</td>
<td>0.65 (+0.14)</td>
</tr>
<tr>
<td></td>
<td>0.73 (-0.01)</td>
<td>0.75 (-0.01)</td>
<td>0.72 (-0.02)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Had.</th>
<th>DeepWalk</th>
<th>LINE</th>
<th>node2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.97 (-0.03)</td>
<td><strong>0.74 (-0.2)</strong></td>
<td>0.93 (-0.12)</td>
</tr>
<tr>
<td></td>
<td>0.95 (-0.06)</td>
<td>0.72 (-0.01)</td>
<td>0.89 (+0.05)</td>
</tr>
<tr>
<td></td>
<td>0.97 (0.0)</td>
<td><strong>0.77 (-0.17)</strong></td>
<td>0.94 (-0.05)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W L_1</th>
<th>DeepWalk</th>
<th>LINE</th>
<th>node2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.96 (-0.01)</td>
<td>0.60 (+0.14)</td>
<td>0.83 (+0.09)</td>
</tr>
<tr>
<td></td>
<td><strong>0.95 (-0.33)</strong></td>
<td>0.70 (-0.01)</td>
<td><strong>0.88 (-0.28)</strong></td>
</tr>
<tr>
<td></td>
<td>0.96 (-0.01)</td>
<td>0.63 (-0.03)</td>
<td>0.85 (+0.03)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W L_2</th>
<th>DeepWalk</th>
<th>LINE</th>
<th>node2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.96 (-0.01)</td>
<td>0.61 (+0.14)</td>
<td>0.83 (+0.09)</td>
</tr>
<tr>
<td></td>
<td><strong>0.95 (-0.33)</strong></td>
<td>0.71 (-0.02)</td>
<td><strong>0.89 (-0.27)</strong></td>
</tr>
<tr>
<td></td>
<td>0.96 (0.0)</td>
<td>0.62 (-0.02)</td>
<td>0.85 (+0.03)</td>
</tr>
</tbody>
</table>
## Experimental Results

Replicating the CNE [2] LP evaluation:

### Issues:
- Performance degradation from parallelization (Metapath2vec)

<table>
<thead>
<tr>
<th>Method</th>
<th>Facebook</th>
<th>PPI</th>
<th>arXiv</th>
<th>BlogCatalog</th>
<th>wikipedia</th>
<th>studentdb</th>
</tr>
</thead>
<tbody>
<tr>
<td>CN</td>
<td>0.97 (+0.01)</td>
<td>0.77 (0.0)</td>
<td>0.94 (+0.01)</td>
<td>0.92 (+0.01)</td>
<td>0.84 (0.0)</td>
<td>0.42 (−0.01)</td>
</tr>
<tr>
<td>JS</td>
<td>0.97 (+0.01)</td>
<td>0.76 (0.0)</td>
<td>0.94 (+0.01)</td>
<td>0.78 (0.0)</td>
<td>0.50 (−0.01)</td>
<td>0.42 (−0.01)</td>
</tr>
<tr>
<td>AA</td>
<td>0.98 (0.0)</td>
<td>0.77 (+0.01)</td>
<td>0.94 (+0.01)</td>
<td>0.93 (0.0)</td>
<td>0.86 (+0.01)</td>
<td>0.42 (−0.01)</td>
</tr>
<tr>
<td>PA</td>
<td>0.83 (+0.01)</td>
<td>0.89 (+0.01)</td>
<td>0.86 (+0.01)</td>
<td>0.95 (0.0)</td>
<td>0.91 (0.01)</td>
<td>0.91 (0.01)</td>
</tr>
<tr>
<td>DeepWalk</td>
<td>0.98 (0.0)</td>
<td>0.64 (+0.01)</td>
<td>0.92 (+0.01)</td>
<td>0.61 (−0.01)</td>
<td>0.56 (0.0)</td>
<td>0.76 (+0.03)</td>
</tr>
<tr>
<td>LINE</td>
<td>0.95 (0.0)</td>
<td>0.75 (+0.01)</td>
<td>0.98 (0.0)</td>
<td>0.76 (+0.01)</td>
<td>0.71 (0.0)</td>
<td>0.86 (−0.02)</td>
</tr>
<tr>
<td>Node2vec</td>
<td>0.99 (0.0)</td>
<td>0.68 (+0.02)</td>
<td>0.97 (0.0)</td>
<td>0.73 (−0.05)</td>
<td>0.67 (−0.08)</td>
<td>0.83 (0.0)</td>
</tr>
<tr>
<td>Metapath</td>
<td>0.74 (+0.17)</td>
<td>0.85 (0.0)</td>
<td>0.83 (+0.02)</td>
<td>0.91 (0.0)</td>
<td>0.83 (+0.02)</td>
<td>0.92 (−0.01)</td>
</tr>
<tr>
<td>CNE(unif.)</td>
<td>0.99 (0.0)</td>
<td>0.89 (+0.01)</td>
<td>0.99 (0.0)</td>
<td>0.92 (+0.01)</td>
<td>0.84 (0.0)</td>
<td>0.93 (0.0)</td>
</tr>
<tr>
<td>CNE(deg.)</td>
<td>0.99 (0.0)</td>
<td>0.91 (0.0)</td>
<td>0.99 (0.0)</td>
<td>0.96 (0.0)</td>
<td>0.92 (−0.01)</td>
<td>0.94 (0.0)</td>
</tr>
</tbody>
</table>
Experimental Results

Replicating the PRUNE [3] LP evaluation:

Issues:

- Missing details in experimental setup
- No open-source implementation of one of the baselines (NRCL)

<table>
<thead>
<tr>
<th></th>
<th>DeepWalk</th>
<th>LINE</th>
<th>Node2vec</th>
<th>SDNE</th>
<th>PRUNE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hep-Ph</td>
<td>0.80 (-0.05)</td>
<td>0.80 (-0.09)</td>
<td>0.81 (-0.05)</td>
<td>0.75 (-0.04)</td>
<td>0.86 (-0.03)</td>
</tr>
<tr>
<td>FB-wallpost</td>
<td>0.83 (+0.01)</td>
<td>0.78 (+0.09)</td>
<td>0.85 (-0.09)</td>
<td>0.86 (-0.02)</td>
<td>0.88 (-0.01)</td>
</tr>
</tbody>
</table>
Experimental Results

Difference in performance between two popular implementations of NE methods (OpenNE and original)
Experimental Results

Scalability of the edge set selection method and comparison with the naive approach:
Future Work

● Integrate embedding visualization.
● Include multi-label classification evaluation.
● Design a flexible GUI capable of auto-generating configuration files.
● Include Wilson’s loop-erased random walk algorithm for selecting a spanning tree uniformly at random.
Acknowledgements

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 Skłodowska-Curie Grant Agreement no. 665501.
Thanks!

Questions
Experimental Results

Replicating the SDNE [4] LP evaluation:

Issues:

- Missing details in experimental setup
- The authors used all graph non-edges to compute prec@k. We approximated this values.

<table>
<thead>
<tr>
<th></th>
<th>prec@100</th>
<th>prec@200</th>
<th>prec@300</th>
<th>prec@500</th>
<th>prec@800</th>
<th>prec@1000</th>
<th>prec@10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDNE</td>
<td>1 (+0.00)</td>
<td>1 (+0.00)</td>
<td>1 (+0.01)</td>
<td>0.99 (+0.00)</td>
<td>0.97 (+0.03)</td>
<td>0.91 (+0.09)</td>
<td>0.25 (+0.12)</td>
</tr>
<tr>
<td>LINE</td>
<td>1 (+0.00)</td>
<td>1 (+0.00)</td>
<td>0.99 (+0.01)</td>
<td>0.93 (+0.06)</td>
<td>0.74 (+0.26)</td>
<td>0.79 (+0.11)</td>
<td>0.21 (+0.13)</td>
</tr>
<tr>
<td>DeepWalk</td>
<td>0.6 (+0.40)</td>
<td>0.55 (+0.45)</td>
<td>0.44 (+0.56)</td>
<td>0.34 (+0.65)</td>
<td>0.29 (+0.70)</td>
<td>0.29 (+0.71)</td>
<td>0.15 (+0.01)</td>
</tr>
<tr>
<td>GraRep</td>
<td>0.04 (+0.96)</td>
<td>0.03 (+0.97)</td>
<td>0.03 (+0.97)</td>
<td>0.04 (+0.96)</td>
<td>0.03 (+0.97)</td>
<td>0.08 (+0.97)</td>
<td>0.19 (+0.16)</td>
</tr>
<tr>
<td>CN</td>
<td>1 (+0.00)</td>
<td>0.96 (+0.03)</td>
<td>0.96 (+0.03)</td>
<td>0.98 (+0.00)</td>
<td>0.87 (+0.05)</td>
<td>0.79 (+0.09)</td>
<td>0.19 (+0.00)</td>
</tr>
<tr>
<td>LapEig</td>
<td>0.93 (+0.07)</td>
<td>0.85 (+0.15)</td>
<td>0.82 (+0.17)</td>
<td>0.66 (+0.34)</td>
<td>0.46 (+0.53)</td>
<td>0.39 (+0.48)</td>
<td>0.05 (+0.15)</td>
</tr>
</tbody>
</table>