LARGE PROJECT - NETWORK REPRESENTATION LEARNING
A COMPARATIVE STUDY FOR UNSUPERVISED NETWORK REPRESENTATION LEARNING

DD2434/FDD3434 MACHINE LEARNING, ADVANCED COURSE

Zakaria Nassreddine, Oliver Möller, Ernest Pokropek

Group no. 15

KTH Royal Institute of Technology

December 2021

ABSTRACT

In this work, we replicate the findings of Khosla et al. [7] by implementing five graph embedding algorithms: node2vec, DeepWalk, NetMF, GraphSage and LINE-1 and validate their performance on selected data sets. We support our investigation with external benchmarks to validate our results and to tackle the problem of limited computational resources available. We provide an investigation of link prediction and node classification based on the obtained embeddings and describe our limitations caused by the computational barrier. We find that the results for different embedding algorithms on the same data sets sometimes differ, not only in our investigation but in scientific literature as well. Trends of performance, however, are in majority similar to the investigated paper. Furthermore, we elaborate on difficulties and challenges of such empirical investigations supported by ours as well external results and discuss their possible implications.

1 Introduction

Network Representation Learning is a technique of learning that embeds the network into a low-dimensional vector space while preserving its information such as topology structure or content. Motivation behind it is to explore the most complex and high-order structures and exploit the network while preserving its structure, integrity, and content. We can think about networks in various manners, but imagine for example a network of users on social media website. We may want to classify them into different groups, control spread of misinformation and detect structures of particular communities. This is a great challenge for such algorithms that want to preserve how this network behaves to predict new situations and outcomes in a generalisable and reliable fashion. In this work, we analyse a comparative study for unsupervised network representation learning as presented in [7], where they performed a large empirical study of selected embedding algorithms on the most widely used network data sets. We implement and explore 5 network embedding algorithms, which are further explained and listed in section 3.1, that the original paper used. We also use the same data sets (listed in 3.4) and we aim to (1) experiment how well the algorithms perform and (2) explore whether our and [7] results are similar to each other and to related work from scientific literature.

2 Authors and contributions

Zakaria Nassreddine: Network embeddings with node2vec and DeepWalk, Link Prediction pipeline and results, Multi-label Node Classification pipeline and results.
Ernest Pokropek: Network embeddings with GraphSAGE, managing the external computational cluster for running the scripts, literature and benchmarking review.
Oliver Möller: Network embeddings with netMF and LINE-1, Node Classification pipeline.

3 Methods

To this end, we re-implemented five of the algorithms discussed in the paper and that we further explain in section 3.1. We used the data sets mentioned in the paper except for the one we couldn't find, more data details in section 3.4. However, due to hardware and time limitations, we were unfortunately not able to exploit all the data sets and perform all the tasks as we would have wished, more details on this and our approach to go around it in section 4.

3.1 Algorithms

node2vec: node2vec is a random walk based feature learning algorithm to perform node embedding on graphs. Inspired by greatly successful approaches from the field of Natural Language Processing, it was introduced by Grover et al. [5] and it relies on word2vec, a widely acclaimed NLP algorithm to learn feature representations from tokens of natural language (words), but generalises the structure from a corpus of linear sequences (sentences) to a graph of any shape. It does so by performing second order biased random walks on graphs to generate (linear) node sequences, and these walks can be fine tuned (with a return parameter p and an in-out parameter q) to flexibly generate neighbourhoods that encapsulate a mixture of homophily and structural equivalence similarities, depending on the properties of the graph that we want to favour capturing. The two extreme cases of this mixture would be analogous to a Depth First Search (q $<$ 1) and a Breadth First Search (p $<$ 1). From these sequences, we generate pairs of input-context nodes for a given context window size, and we feed them to a Skip Gram model trained with Negative Sampling and whose number of neurons in the hidden layer determines the size of the embeddings. Once the training is complete, we retrieve the hidden layer weights as node embeddings.
The Skip Gram model is provided in word2vec from the gensim library. However, in our implementation, we used a different version than the authors, which we think is important to keep in mind when comparing the results. Using the same version of the library would have been preferred with regards to result reproduction, but we decided against it knowing that the newer implementation is expected to increase execution speed.

DeepWalk: DeepWalk can be thought of as a predecessor to node2vec, it follows the same intuition and NLP insipiration, but samples random walks in a less sophisticated and more rigid way: at each step, the next vertex is chosen at random. It also does not go the extra mile to use Negative Sampling to speed the computations up, and instead uses Hiearchical Softmax to train the Skip Gram model. It was introduced by Perozzi et al. [9]

GraphSAGE [6] is an inductive learning framework that uses the so called *aggregator function* making it feasible to use on newly added nodes without having to retrain the whole thing. The aggregator function can induce this embedding basing on its neighbourhood and features. The main intuition behind this algorithm is that nodes being in the same neighbourhood should have rather similar embeddings (the neighbourhood is defined by parameter K being the depth, i.e. the distance or the geodesic from the examined node). The aggregator function works by, intuitively,

aggregating information from this neighbourhood to generate a new embedding, thus instead of inductively learning the embeddings we train the aggregator to generate them. As input it accepts the neighbourhood and combines each of the neighbour's embedding with weights, generating the neighbourhood embedding. This process is being done as (a) initialize embeddings of all nodes to their features (b) for each depth until K, for each node, create embedding of the neighbourhood with the aggregator function, (c) concatenate this embedding with the one that was previously there and (d) pass the concatenated vector through a layer of neural network to update the node's embedding. After this process, the embeddings are normalized to the unit form.

NetMF originates from the unification of several graph embeddings as matrix factorization. In their paper [10] the authors realize that the matrix factorization of DeepWalk is in regards to generalization and performance from particular interests. They, therefore, utilize it as the basis for a new embedding algorithm - NetMF or Network embedding as matrix factorization. The DeepWalk matrix factorization is closely related to Graph Laplacian, which we know from the lecture. The Graph-Laplacian aims to describe the relationships between vertices and edges of a Graph. However, this matrix is still high-dimensional since it has #vertices columns and rows. To reduce dimensionality, NetMF utilizes eigendecomposition and picks the top eigenpairs to approximate the necessary matrix. This methodology should improve performance compared with other embedding algorithms, such as DeepWalk or Line, which implicitly approximate similar matrix factorizations.

LINE/LINE-1 (Large-scale Information Network Embedding) [14] is designed to attack the problem that most existing graph embedding methods do not scale for more extensive networks, such as modern social networks. LINE is suitable for undirected, direct, unweighted, and weighted graphs. As with the different embeddings, LINE tries to capture the graph's structure and relations in vectors for each node. In our implementation, each of these vectors consists of 128 dimensions. LINE is therefore considered to embed a graph into low dimensional space. For people unfamiliar with Graph Embeddings, 128 dimensions may not sound low-dimensional. Therefore, we would like to mention that graphs with millions of vertices are nothing more than an adjacency matrix with number of vertices being dimensions. Consequently, reducing the complexity from millions to 128 is a big step. It is important to note that the original paper presents LINE in two variations, respectively LINE-1 and LINE-2. The difference lies on the way proximity for the embeddings is computed. LINE-1 considers first-order proximity, the local, pairwise distance between two nodes. For disconnected nodes, the first-order proximity is zero. For connected vertices, the weight of the edge indicates it. Note: LINE-1 is therefore only applicable to **undirected** graphs. However, LINE-2 calculates the second-order proximity and considers the similarity of the neighborhoods of the two nodes and not the connection between the nodes themselves. At this point, I like to mention that the original LINE-Algorithm was implemented with several hyperparameters. However, Khosla et al.[7] do not mention which hyperparameter setting they used for their performance measurements. In our implementation, we orientated on the default parameter setting from [14] but immensely reduced the batch size from 300000 to 10000 to allow faster computation times to the cost of less precise results.

3.2 Link Prediction

For this task, we implemented our own pipeline in a different way than what the authors of the paper used. We deemed that the most intuitive way to predict edges between two vertices would be to train on a data set that contains all the positive (existing) edges in the graph that we build from the edge list, to which we add a sample of negative (non existing) edges of the same size to ensure class balance for model training. We do so by creating all possible pairs of nodes in the graph, then we remove all the positive egdes, this leaves us with pairs of nodes that are not linked. From this we sample a set of negative edges of the same size as the edge list. We build a dictionary that maps a node to its embedding for a given algorithm and we build a training data set in the following format: **Pattern**: a 256 dimensional vector built from concatenating the embeddings of the two nodes in the edge and **Label**: 1 for positive edges and 0 for negative edges.
We use a 50-50 training-validation split, train a Logistic Regression Classifier using Scikit-Learn and report micro and macro F1 scores. Our cluster, As mentioned in section 4, broke down in the last few of days leading to the deadline and we could no longer perform any computing on it, neither of us having enough credit left with the Google Cloud Platform accounts linked to our credit cards, we could only run computations on our personal machines, which was very unfortunate. This mean that, for link prediction, which was also rather computationally greedy for large datasets, we could only run our LP pipeline on reasonably sized embedding files (cora, blogcat and us-airports). For the rest, we had to do some sub-sampling which, to the best of our knowledge, should have minimised the bias in prediction. We sample only 12000 node embedding and 350000 edges to build our supervised data set. However, this subsampling seems to have greatly impacted the prediction accuracy, and we want to highlight that the LP results for large datasets are not very reliable. This is something that we would be interested in discussing with the TAs.
Finally, we were late to come to the realisation that a Logistic Regression Classifier does not take order of input variables into account, which encodes edge direction, so we made a new classifier based a Random Forest and we ran

both. More details in the Results section.

3.3 Node Classification

For this task, we built two pipelines for single-label and multi-label classification. For the training patterns, we always take the node embeddings. The labels are either the class for single-label classification, or an array of classes in a multi-label setting. We perform padding by adding zeros so as to have the same length for all label arrays and train a single-output and a multi-output KNeighbourClassifier from Scikit-Learn for each situation.

3.4 Data

Some of the datasets were accessed using KONECT [8]. The original paper used following datasets: BlogCatalog [18], Flickr [12], Youtube [13], Reddit [6], Twitter [16], Epinion [4], DBLP-Ci [15], Cora [2], PubMed [17] and DBLP-Au [11]. Although we accessed them all and referenced properly, the majority of them were to big to run on the machines we were able to access as students (read more in section 4). We also used another dataset not included in the paper by name of "US Airports" [1] to explore the performance on even more lightweight data that we can easily run our algorithms on given our computational resources. Summary of used datasets can be found in Table 1. Due to immense troubles with searching for the CoCit dataset (misleading references or ones leading to non-existent versions of KONECT) it was not included in our findings - it has to be noted however, that this dataset was not special in any manner and no crucial findings were concerning it.

	|V|	|E|	r	clus	T	weighted	directed	labelled
US airports [1]	**1.5k**	**28k**	**0.780**	**0.384**	**0.384**	**Yes**	**Yes**	**No**
Cora [2]	23k	91k	0.05	0.2660	0.1169	No	Yes	Yes
BlogCat [18]	10k	333k	n.a.	0.463	0.0914	No	No	Yes
DBLP-au [11]	1.2M	10.3M	n.a.	0.0635	0.1718	No	Yes	No
Epinion [4]	75k	508k	0.4052	0.1378	0.0657	No	No	No
Twitter [16]	465k	834k	0.003	0.0006	0.0152	No	No	No

Table 1: Structural properties of used datasets, one added by us (outside the scope of [7] is bolded).

4 Limitations

The authors of the original paper report having used a very powerful infrastructure (80 core CPU servers, 16GB GPU units and a terabyte of main memory). We, unfortunately, did not have the luxury to run our algorithms on such hardware and very quickly and constantly ran into paralysing computing power challenges. Apart from the smaller datasets (cora, US airports and occasionally blogcat), none of the methods could finish without our computers running out of memory (getting a killed process after waiting for hours). We tried alternatives like a virtual machine instance on Google Cloud Platform (and used the free credits almost immediately) but our best shot was to use a High Performance Computing cluster. We used as much resources as we could get our hands on, with some experiments running on up to 80 CPUs and 256Gb of main memory, depending on what was available at the time (as the computing cluster is shared between the employees). For example, with node2vec, we managed to get embeddings for cora blogcat, epinion, dblp-au, twitter and us-airports. When running DeepWalk, we tried storing walks into files instead of saving them all to memory when we exceeded its limit (even though it was notably high, i.e. way more that a personal computer could possibly have), but the method still could not finish for all data sets with the available resources on the cluster at runtime.

This being said, although we used a professional computational cluster made for such experiments, in the short time period of the project we were physically unable to use the algorithms on the bigger datasets due to computational barrier. To make sure that the error is indeed caused by that and not possibly our faulty implementation, we have compared the runs and our code with external implementation of examined algorithms: GraphSAGE original implementation[1],GraphSAGE PyTorch implementation[2], NetMF original implementation[3], LINE-1 original implementation[4], DeepWalk original implementation[5], node2vec original implementation[6] as well as its Python interpretation[7]. Furthermore, we also investigated benchmarking/multi-algorithm frameworks such as StellarGraph [3] and GraphVite [19] to validate the benchmarks and once again check the validity of our implementations, as each of the frameworks listed is

[1] https://github.com/williamleif/GraphSAGE [2] https://github.com/twjiang/graphSAGE-pytorch
[3] https://github.com/xptree/NetMF [4] https://github.com/tangjianpku/LINE [5] https://github.com/phanein/deepwalk
[6] https://github.com/aditya-grover/node2vec [7] https://github.com/eliorc/node2vec

open source.
Another difficulty was caused by different purposes of the examined algorithms - GraphSAGE for example, has to learn on the features in the graph, however from the small sized datasets that had these features and we were able to effectively run the examination only on the Cora dataset.
With aforementioned limitations caused by simply not having enough computational resources, we did not want to leave the investigations of certain datasets empty. Thus, for some of the results, we conducted a literature review of different benchmarks and papers to learn whether different works agree with the results in [7]. Please note, that all such metrics that are not produced by us are marked with appropriate reference as we do not want to claim these metrics as our own in this sense and keep this report honest and reliable. It has to be noted as well, that all the benchmarks suffered from memory problems too, even the original paper that we tried to replicate. For example when it comes to flickr data, node2vec causes out of memory error just with 10% of it on 24 CPU threads and 4 V100 GPUs [19].
Lastly, we had no chance of including the time comparisons between algorithm as we ran our experiments on total of 7 completely different machines, varying from laptops, through personal computers, to computational cluster to meet the project deadline.

5 Results

Subsection 5.1 presents the link prediction results for our algorithms on a selection of data sets. In subsection 5.2 we present the results for multilabel node classification in a similar manner. x in the table means that we were not able to compute the embeddings, and n.a mean the data is unfit for a given algorithm (see section 4). The performance is measured using the F1 score (or F-measure). It is essentially a harmonic mean of precision and recall which takes values between 0 (worst) and 1 (best). It can be computed as:

$$F1 = 2 * \frac{precision \times recall}{precision + recall} \tag{1}$$

5.1 Link Prediction

5.1.1 Logistic Regression (Direction Agnostic)

In this subsection, we use the built-in Logistic Regression classifier from Scikir-Learn with the two-node embedding concatenation as input and a (0,1) output corresponding to a negative/positive edge respectively. Logistic Regression doesn't take input variable order into account, which means that this classifier is not aware of edge direction. We run it on all the data sets we managed to exploit and report the following results.

	Cora		Twitter		DBLP-Au		Epinion		Blogcat		US Airports	
method	mic	mac	mic	mac	mic	mac	mic	mac	mic	mac	mic	mac
DeepWalk	0.598	0.595	x	x	x	x	0.989	0.821	0.880	0.880	n.a	n.a
node2vec	0.554	0.542	0.999	0.499	x	x	0.985	0.666	0.845	0.845	0.819	0.809
NetMF	0.730	0.730	x	x	x	x	x	x	0.817	0.816	0.813	0.805
LINE-1	0.757	0.757	x	x	x	x	x	x	0.859	0.859	0.793	0.782
GraphSage	0.640	0.640	x	x	x	x	x	x	x	x	x	x

Table 2: Micro- and Macro F1 scores for link prediction with a classifier that doesn't take into account edge direction

5.1.2 Random Forest (Direction Aware)

In this subsection, we use a Random Forest Classifier that takes input variable order into account in order to encode edge direction. We run it only on directed graphs to see if differences arise and report the following results.

	Cora		Twitter		Epinion		US Airports	
method	mic	mac	mic	mac	mic	mac	mic	mac
DeepWalk	0.955	0.955	x	x	0.988	0.753	n.a	n.a
node2vec	0.832	0.832	0.999	0.499	0.984	0.547	0.872	0.866
NetMF	0.776	0.774	x	x	x	x	0.880	0.874
LINE-1	0.750	0.750	x	x	x	x	0.875	0.871
GraphSage	0.763	0.763	x	x	x	x	x	x

Table 3: Micro- and Macro F1 scores for link prediction with a classifier that doesn't take into account edge direction

5.2 Node Classification

Here we present the results of multi-class node classification. Table 4 and Table 5 present comparison of results from various scientific sources and benchmarks (referenced). Table 6 presents the results based on our empirical evaluation. The task consisted of predicting a missing node label, for which we employed the multi-label multi-output KNN classifier with $k = 3$.

Dataset	Algorithm	src	Micro-F1	Macro-F1	Diff
Flickr	DeepWalk	[7]	0.4220	0.3100	**ref**
		[14]	0.6113	0.5926	-0.23595 ↓
		[19]	0.6379	0.6242	-0.26505 ↓
	LINE-1	[7]	0.4092	0.2619	**ref**
		[14]	0.6396	0.6278	-0.29815 ↓
		[19]	0.6379	0.6246	-0.29570 ↓
	node2vec	[7]	0.4211	0.3057	x
		[19]	OOM	OOM	
Youtube	DeepWalk	[7]	0.4709	0.3989	**ref**
		[19]	0.4639	0.3851	0.01040 ↑
		[14]	0.4523	0.3586	0.01905 ↑
	LINE-1	[7]	0.4749	0.4117	**ref**
		[19]	0.4625	0.3840	0.02005 ↑
		[14]	0.4221	0.3444	0.04000↑
	node2vec	[7]	0.4841	0.4204	**ref**
		[19]	0.4641	0.3862	0.02710 ↑

Table 4: Comparison of literature benchmarks to the findings of [7] for Flickr and Youtube datasets that were too large for us to run on available computational resources. The difference is calculated as mean difference between micro and macro F1 scores, with [7] treated as reference (**ref**). OOM is an abbreviation for Out Of Memory.

Dataset	Algorithm	src	F1	diff
Reddit[6]	DeepWalk w. features	[7]	0.93205	**ref**
		[6]	0.69100	-0.24105 ↓
	GraphSAGE	[7]	0.86110	**ref**
		[6]	0.95000	0.08890 ↑

Table 5: F1 comparison between GraphSAGE and DeepWalk basing on different sources on Reddit data.

	Cora		BlogCat	
method	mic.	mac.	mic.	mac.
DeepWalk	0.834	0.814	0.890	0.195
node2vec	0.492	0.455	0.889	0.193
NetMF	0.314	0.262	0.894	0.198
LINE-1	0.620	0.597	0.875	0.184
GraphSage	0.648	0.622	x	x

Table 6: Multilabel Node Classification in terms of Micro- and Macro-F1

6 Discussion

6.1 Link Prediction

We start by observing that DeepWalk seems to outperform node2vec in terms of prediction accuracy, which raises the question of the relevance of the sophisticated second order random walks, and this is something we were able to reproduce from the original paper that yields a similar observation. However, the negative sampling optimisation seems to improve the cost considering that node2vec was able to finish on more data sets than its simpler predecessor. Another result that we reproduced is that node2vec performs better on data sets with high reciprocity like Epinion, although we didn't reproduce the gain compared to DeepWalk. The reason behind this could be that we didn't generate our random

negative edges in the test set in the same way that the authors did. We also observe that LINE can outperform DeepWalk and node2vec on graphs with low clustering coefficients and transitivity (e.g Cora) and that this gain shrinks when the graphs present higher values for these properties (e.g US Airports), but we unfortunately did not have results on enough data sets to really challenge this claim made in the paper. We note that NetMF yields some very good results on smaller data sets, and while it is true that it has no real advantage over other methods when the data is of significant size, we would disagree with the paper (that claims it adds no value) and suggest that it has this nice quality of performing well when the data is scarce. While the results for GraphSage seem promising on Cora, we did not have the resources to train it on larger data sets as our cluster broke before we got to it, so the data we have doesn't present enough statistical power to make any conclusions.
A critical point that requires further investigation is the use of LINE-1 in the benchmarks for directed graphs in the paper from Khosla et al. [7]. LINE-1 is based on first-order proximity and therefore only applicable for undirected graphs, not for directed graphs [14]. In some cases, the fraudulent use of LINE-1 can still approximate a good solution. However, the risk of erroneous results is very high. It is possible that the authors were not aware of that pitfall and falsy interpreted a directed graph (e.g., Cora) as an undirected graph to run LINE-1. Note that if a subset of a directed graph is undirected, LINE-1 can be used on the subset to approximate results. In order to reproduce the results from the paper, we used LINE-1 on directed graphs too. You need to be aware of this when comparing results.
Finally, we would like to note that it's important to keep in mind that some of the deviation between the performances on different graphs may also be due to the size of the data, but with our limited resources we couldn't conduct conclusive experiments to control for that background variable.

6.2 Node classification

Looking at Table 6 we clearly see that DeepWalk outperforms all of the embedding algorithms for the Cora dataset, followed by GraphSage and then LINE-1, which is rather similar to the results obtained in [7]. However, node2vec and NetMF suffer from a considerable drop in performance in comparison. For the BlogCat dataset, all algorithms have a very high difference between macro and micro F1 scores, with macro being really small, and micr relatively big. The trend here is similar to the one in [7], but our micro values are significantly larger. Our discrepancies are much bigger however, and this is probably caused by improper fit due to limited number of possible training epochs caused by the lack of time. It needs to be noted that the literature review is not fully agreeing with the paper either - in Table 4 for Flickr dataset, we see that the listed algorithms have rather notable differences in F1 scores. The difference for the YouTube dataset is rather small, but still clearly visible. For the Reddit dataset, the results in Table 5 once again follow a notable differences in performance - note that the GraphSAGE in the paper describing it ([6]) performs vastly better than in [7].

6.3 Closing remarks

The task of empirical investigations of various embedding algorithms is an incredibly tough challenge. The main issue that we experienced was the problem concerning computational resources, or more precisely, lack of them - although we did suffer from it to a great extent due to being students who do not have access to proper resources in terms of computing clusters, many researchers experienced similar issues. This has caused the research community to try to overcome this problem, for example by sampling the data sets to achieve smaller ones - this however is done differently by different benchmarks, and thus results in more or less notable differences in them. Furthermore, it is often unclear how and how much of the data was sampled, making it even more challenging to replicate.

Furthermore, we would like to address the following points made in [7]:
"Methods respecting the vertex's role as source and context during learning of representations as well as in their use for a task are recommended for link prediction in directed graphs." **Our findings**: Here we can consider only GraphSAGE as such a method, and we can agree with the authors as we observed this very algorithm to perform well and never being the worst. Although our experiment is too narrow to make such a claim, its results fall into the point that the authors made.
"Certain structural properties like clustering coefficient, transitivity, reciprocity etc. are recommended to be considered while choosing a specific method." **Our findings**: Correct. Based on our experiments, we have seen that data sets of different values for these parameters make the algorithms perform differently, as they are fit to different problems. And obviously, algorithms that require features in the graph, should be used only for data sets having such features.

All this being said, we have successfully implemented the five listed algorithms of graph embedding and empirically tested that they are more or less successful on the selected data sets. However, we also proved, that in various sources, results of similar looking investigations might differ even vastly, and it is really important for the research community to be critical of the results as they may not be entirely correct. Another point we would like to make, is that among all the algorithms we implemented there were no faulty ones - a poor performance is solely the result of using it not in the

proper way, e.g. not enough training. There is a saying in the tech community that computational resources are cheaper than skilled programmers. Although this is valid for plenty of problems, the task of network representation learning is not one of them, and we think that one of the major concerns that the scientific community should focus on now is how to enable this technology to work on big data efficiently, as the hard metrics e.g. f1 score, accuracy etc. are not too big of a concern if we cannot even train the models.

We think the original paper authors did some excellent and much needed work to provide a unified framework to compare the ever expanding array of network embedding approaches that are produced by the research community. These guidelines would prove to be very valuable to practitioners and other researchers that don't have the resources to dedicate to benchmarks prior to a study or product development. But we would have appreciated it if more effort was put into making their methodology clearer and more reproducible, e.g elaborating on the negative edge generation process and providing direct links to the data sets. We learnt a lot from the papers we had to read to understand and implement the algorithms and how to build robust pipelines to process heterogeneously formatted data produced by different people and programmes.

References

[1] Vittoria Colizza, Romualdo Pastor-Satorras, and Alessandro Vespignani. "Reaction–diffusion processes and metapopulation models in heterogeneous networks". In: *Nature Physics* 3.4 (Apr. 2007). Bandiera_abtest: a Cg_type: Nature Research Journals Number: 4 Primary_atype: Research Publisher: Nature Publishing Group, pp. 276–282. ISSN: 1745-2481. DOI: 10.1038/nphys560. URL: https://www.nature.com/articles/nphys560 (visited on 01/07/2022).

[2] *Cora dataset*. URL: https://web.archive.org/web/20150918182409/http://www.cs.umd.edu/~sen/lbc-proj/data/cora.tgz.

[3] CSIRO's Data61. *StellarGraph Machine Learning Library*. https://github.com/stellargraph/stellargraph. 2018.

[4] *Epinions*. URL: http://konect.cc/networks/soc-Epinions1/ (visited on 01/02/2022).

[5] Aditya Grover and Jure Leskovec. *node2vec: Scalable Feature Learning for Networks*. 2016. arXiv: 1607.00653 [cs.SI].

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

[7] Megha Khosla, Vinay Setty, and Avishek Anand. "A Comparative Study for Unsupervised Network Representation Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 33.5 (2021), pp. 1807–1818. DOI: 10.1109/TKDE.2019.2951398.

[8] Jérôme Kunegis. "KONECT: The Koblenz Network Collection". In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13 Companion. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, pp. 1343–1350. ISBN: 9781450320382. DOI: 10.1145/2487788.2488173. URL: https://doi.org/10.1145/2487788.2488173.

[9] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: ACM, 2014, pp. 701–710. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623732. URL: http://doi.acm.org/10.1145/2623330.2623732.

[10] Jiezhong Qiu et al. "Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec". In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM. 2018, pp. 459–467.

[11] *SNAP: Network datasets: DBLP collaboration network*. URL: https://snap.stanford.edu/data/com-DBLP.html (visited on 01/02/2022).

[12] *Social Computing / Online Social Networks Research @ MPI-SWS: Data sets - Flickr*. URL: http://socialnetworks.mpi-sws.mpg.de/data/flickr-links.txt.gz (visited on 01/02/2022).

[13] *Social Computing / Online Social Networks Research @ MPI-SWS: Data sets - Youtube*. URL: http://socialnetworks.mpi-sws.mpg.de/data/youtube-links.txt.gz (visited on 01/02/2022).

[14] Jian Tang et al. "LINE: Large-scale Information Network Embedding." In: *WWW*. ACM. 2015.

[15] Jie Tang et al. "ArnetMiner: extraction and mining of academic social networks". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '08. New York, NY, USA: Association for Computing Machinery, Aug. 24, 2008, pp. 990–998. ISBN: 978-1-60558-193-4. DOI: 10.1145/1401890.1402008. URL: https://doi.org/10.1145/1401890.1402008 (visited on 01/02/2022).

[16] *Twitter (ICWSM)*. URL: http://konect.cc/networks/munmun_twitter_social/ (visited on 01/02/2022).

[17] *UCI Machine Learning Repository: Bag of Words Data Set*. URL: http://archive.ics.uci.edu/ml/datasets/Bag+of+Words (visited on 01/02/2022).

[18] R. Zafarani and H. Liu. *Social Computing Data Repository at ASU*. 2009. URL: http://socialcomputing.asu.edu.

[19] Zhaocheng Zhu et al. "GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding". In: *The World Wide Web Conference*. ACM. 2019, pp. 2494–2504.