Unsupervised Framework for Evaluating Structural Node Embeddings of Graphs

Ashkan Dehghan[1], Kinga Siuta[1,2], Agata Skorupka[1,2], Andrei Betlen[3], David Miller[3], Bogumił Kamiński[2], and Paweł Prałat[1]

[1] Department of Mathematics, Toronto Metropolitan University, Toronto, ON, Canada {ashkan.dehghan,pralat}@torontomu.ca
[2] Decision Analysis and Support Unit, SGH Warsaw School of Economics, Warsaw, Poland bogumil.kaminski@sgh.waw.pl
[3] Patagona Technologies, Pickering, ON, Canada

Abstract. An embedding is a mapping from a set of nodes of a network into a real vector space. Embeddings can have various aims like capturing the underlying graph topology and structure, node-to-node relationship, or other relevant information about the graph, its subgraphs or nodes themselves. A practical challenge with using embeddings is that there are many available variants to choose from. Selecting a small set of most promising embeddings from the long list of possible options for a given task is challenging and often requires domain expertise. Embeddings can be categorized into two main types: classical embeddings and structural embeddings. Classical embeddings focus on learning both local and global proximity of nodes, while structural embeddings learn information specifically about the local structure of nodes' neighbourhood. For classical node embeddings there exists a framework which helps data scientists to identify (in an unsupervised way) a few embeddings that are worth further investigation. Unfortunately, no such framework exists for structural embeddings. In this paper we propose a framework for unsupervised ranking of structural graph embeddings. The proposed framework, apart from assigning an aggregate quality score for a structural embedding, additionally gives a data scientist insights into properties of this embedding. It produces information which predefined node features the embedding learns, how well it learns them, and which dimensions in the embedded space represent the predefined node features. Using this information the user gets a level of explainability to an otherwise complex black-box embedding algorithm.

Keywords: Node Embeddings · Structural Node Embeddings.

1 Introduction

Inspired by early work in word embedding techniques [18], node, edge and graph embedding algorithms have gained a lot attention in the machine learning community, in recent years. Indeed, learning an accurate and useful latent representation from network-data is an important and necessary step for any successful

machine learning task, including node classification [19], anomaly detection [3], link prediction [11], and community detection [20] (see also surveys [4, 10]).

In this paper, we distinguish two families of node embeddings: classical node embeddings and structural node embeddings. The first family is very rich with already over 100 algorithms proposed in the literature. Informally speaking, *classical node embeddings* fall into a broad and diverse family of embeddings that try to assign vectors in some high dimensional space to nodes of the graph that would allow for its approximate reconstruction using such encapsulated information. Different classical embedding algorithms use different approaches to achieve this task. Some of them, in order to extract useful information from graphs, try to create an embedding in a geometric space by assigning coordinates to each node such that nearby nodes are more likely to share an edge than those far from each other. Some other approaches postulate that pairs of nodes that have overlapping neighbourhoods (not necessarily intermediate ones) should have similar representations in the embedded space. Independently, the techniques to construct the desired classical embeddings can be broadly divided into the following three groups: linear algebra algorithms, random walk based algorithms, and deep learning methods [1, 17].

Classical embeddings work well for machine learning tasks such as link prediction. However, as the study of [23] shows, they do not guarantee good performance, for example, in tasks such as community labeling that can be viewed as a classification task or role detection. The reason is that in these challenging-for-classical-embeddings machine learning problems, when doing inference, it is important to preserve structural characteristics of nodes. Informally speaking, by structural characteristics of nodes we mean the structure of nodes' egonets, which is the induced subgraphs of given nodes and their neighbourhoods up to some fixed depth. The simplest form of one-dimensional structural embeddings are node features such as degree or local clustering coefficient. Indeed, node features have been used extensively since the very beginning of network analysis, as most of them have natural interpretations and are usually relatively easy to compute. From the standpoint of this discussion, it is important to highlight that such node features do not depend on node labels, but rather on the relationships between them. For example, two nodes might both have large and comparable degrees or similar pageranks (and, as a result, end up close to each other in the embedded space) but be distant from each other in terms of concrete neighbours (and so they would be far apart in classical embeddings). The already mentioned study [23] shows that such node features are efficient in various tasks such as community labeling. The reason is that often the role of a node within a graph is an important predictor of some features but not necessarily its concrete neighbours. Since using hand-crafted node features in various machine learning tools has proven to be a useful technique, researcher have developed various *structural embedding algorithms* such as **RolX** [12], **Struct2Vec** [22], **GraphWave** [7] and **Role2Vec** [2]. Such embedding algorithms try to capture structural characteristics of nodes, that is, put nodes that have similar structural characteristics close together in the embedded space. Again, like with classical

embeddings, implementations of structural embeddings differ in the way how they define similarity between neighbourhoods of two nodes.

There are two important questions to consider when studying embedding algorithms. The first question is concerned with what node-features about the graph is learned by a given embedding algorithm. And the second focuses on how well a given node feature is learned by the set algorithm. Answering and understanding these questions is crucial for practitioners of the field, as it will dictate which embedding algorithm is optimal for a given task. Of course, the decision of which algorithm to use might also depend on the properties of the investigated network [6]. There are existing works by [14–16], which aim to answer these questions for the classical type embedding algorithms[4]. There is however no such work to our knowledge that has been done to answer these questions for structural embedding algorithms. In this work, we introduce an unsupervised technique for quantifying how well a given embedding algorithm learns a predefined set of structural features. This provides an explainability of the embedding space in terms of structural-node-features, in addition to allowing one to compare between various different algorithms to identify the most optimal embedding for a given application.

2 Framework

2.1 Input/Output

In this section we introduce our framework and highlight its properties. The goal of the framework is to evaluate possible correlations of various node embeddings with a number of classical node features of a single graph $G = (V, E)$ on $n = |V|$ nodes. The input consists of

- k dimensional node embedding—k vectors of real numbers, each of length n,
- ℓ node features—ℓ vectors of real numbers, each of length n.

The framework outputs the following

- a real number (represented by symbol ψ) from the interval $[0, 1]$ representing how well given feature vectors may be approximated by given embedding vectors; $\psi = 0$ indicates a good approximation and the other extreme value, $\psi = 1$, represents a bad approximation; both pre- and post-optimization values of ψ are returned, where the post-optimization ψ value is computed by minimizing ψ as a function of a vector $\mathbf{w}$—formal definition and more details will be provided soon,
- a vector $\mathbf{w}$ of non-negative real values of length k and L^1-norm equal to 1 that indicates which embedding dimensions contribute to the explanation of features; here, larger values correspond to larger contribution; the $\mathbf{w}$ vector consists of the weights in the embedding distance computation, and is used to identify which embedding dimension the structural feature is mapped onto.

[4] https://github.com/KrainskiL/CGE.jl

The structure of our framework is designed to output a quantitative metric ψ, which measures how well an embedding algorithm has learned a given feature (or a collection of features). This metric can be used to both identify what features embedding algorithms learn, in addition to how well they learn those features. A more comprehensive explanation of this is given in the following section.

2.2 Formal Description of the Algorithm

In our framework, nodes are clustered (using k-means clustering) in the feature-space, and distance between sampled nodes in the feature-space are calculated and compared to the distance measured in the embedded-space. Therefore, the algorithm has a few parameters that the user might experiment with but each of them has a default value:

- s: the number of clusters in the feature space generated by the k-means algorithm (by default, $s = \sqrt{n}$, where n is the number of nodes of a network); the value $s = \sqrt{n}$ is a safe estimated to ensure the convergence and stability of the calculated ψ metric—more on this is discussed in the following section,
- p: the fraction of sampled pairs of nodes that are from the same cluster (by default, $p = 0.5$),
- c: the total number of sampled pairs of nodes (by default, $c = \min\{10^5, n^2/s\}$; apart from a natural upper bound of 10^5, for small networks we need to make sure that the number of pairs of nodes sampled within clusters, $p \cdot c$, is at most the number of all pairs of nodes from the same cluster; indeed, at the worst case scenario each cluster could consist of n/s nodes and so there could be only $\binom{n/s}{2} \cdot s \approx n^2/(2s)$ pairs of nodes within clusters; this would cause a problem as the algorithm samples pairs without replacement),
- standardization method: we provide two methods, MinMax that scales and translates each feature individually such that all of them are in the range between zero and one, and StandardScaler that scales features such that the mean and the standard deviation are equal to zero and, respectively, one (by default, we use the StandardScaler normalization).

The algorithm performs the following steps:

1. **Standardization**. Transform all feature and embedding vectors using one of the two methods, MinMax or StandardScaler. After this transformation, all vectors are appropriately normalized and standardized. As a result, later steps are invariant with respect to any affine transformation of these vectors.
2. **Clustering**. Perform the classical k-means clustering of nodes (into s clusters) in the feature space using the selected metrics. Let $(c_1, \ldots, c_s)$ with $n = \sum_{i=1}^{s} c_i$ be the distribution of cluster sizes.
3. **Sampling**. There are two types of pairs of nodes that are independently sampled as follows.

a) sample

$$\hat{m} = \min\left\{\left\lfloor p \cdot c \right\rfloor, \sum_{1\le a\le s}\binom{c_a}{2}\right\}$$

unique pairs of nodes within clusters; a single pair of nodes is sampled by first selecting cluster i of size c_i with probability equal to

$$p(i) = \frac{\binom{c_i}{2} - x_i}{\sum_{1\le a\le s}\left(\binom{c_a}{2} - x_a\right)},$$

where x_i is the number of pairs already sampled from cluster i, and then selecting a pair of nodes from the chosen cluster, uniformly at random; if a pair of nodes sampled this way is already present in the sampled set we discard it, otherwise we keep it.

b) sample

$$\bar{m} = \min\left\{\left\lfloor (1-p) \cdot c \right\rfloor, \sum_{1\le a<b\le s} c_a c_b\right\}$$

unique pairs of nodes that are between clusters; a single pair of nodes is sampled by first selecting two clusters i, j $(i < j)$ with probability equal to

$$p(i,j) = \frac{c_i c_j - x_{i,j}}{\sum_{1\le a<b\le s}(c_a c_b - x_{a,b})},$$

where $x_{i,j}$ is the number of pairs between cluster i and cluster j already sampled, and then selecting one node from each of the chosen clusters, uniformly at random; if a pair of nodes sampled this way is already present in the sampled set we discard it, otherwise we keep it.

4. **Computing Feature Distance**. For each of the sampled pairs of nodes, compute the corresponding distance in the ℓ-dimensional feature space d_f. For the Euclidean metric we have

$$d_f(v_i, v_j) = \sqrt{\sum_{1\le a\le \ell}(f_a^i - f_a^j)^2},$$

where $(f_1^i, \ldots, f_\ell^i)$ and $(f_1^j, \ldots, f_\ell^j)$ are features of nodes v_i and, respectively, v_j.

5. **Computing Embedded Distance**. Suppose for a moment that a normalized vector of non-negative weights $\mathbf{w} = (w_1, \ldots, w_k)$ with $\sum_{i=1}^k w_i = 1$ is fixed. For each of the sampled pairs of nodes, compute the corresponding distance in the k-dimensional embedded space d_e. The weighted Euclidean distance is given by

$$d_e(v_i, v_j) = \sqrt{\sum_{1\le a\le k} w_a(e_a^i - e_a^j)^2},$$

where $(e_1^i, \ldots, e_k^i)$ and $(e_1^j, \ldots, e_k^j)$ are embeddings of nodes v_i and, respectively, v_j.

6. **Correlation between the two spaces**. To compute the correlation between the two spaces, we define a metric $\psi = 1 - r^2 \in [0, 1]$, where $r \in [-1, 1]$ is the Pearson correlation between vectors in the embedding space and vectors in the feature space. As a result, ψ is defined such that both large positive (close to 1) and large negative (close to -1) correlation would have small values (close to 0). This is done so that the optimization scheme (see the next bullet-point) is more stable.
7. **Optimization**. Optimize vector $\mathbf{w}$ to minimize ψ, where the final value of ψ is referred to as the post-optimization ψ. These optimized vectors reflect the importance of embedded dimensions for selected features. We note that the optimization is done using Quasi-Newtonian bounded constraint minimization technique from *Scipy* Optimize method. We note that the pre-optimization ψ value measures the overall raw embedding of a particular feature. To measure how well a feature is learned by a particular embedding algorithm, ψ is optimized against that feature. The optimization process removes (or minimizes) any embedded information that does not contribute to the representation of the feature at study. Therefore, we use the post-optimization ψ value to conduct all experiments in this study.

2.3 Properties

Let us briefly highlight some basic and desired properties of the framework which, in particular, justify its design and show its potential usefulness.

- The framework is designed in such a way that affine transformations of any of the feature or embedding vectors do not change the results.
- The framework does not assume any particular type of the relationship between feature space and embedded space. Instead, it is desired that if two nodes are close in the feature space, then they are also close in the embedded space (with a proper metrics/weighting in the embedded space).
- The sampling strategy used in the framework has the following consequence. Achieving a good ψ score ensures that close pairs of nodes in the feature space are close in the embedded space. On the other hand, if some pairs of points are far in the feature space, then the framework puts less weight on the fact whether they are close or not in the embedded space. The rationale behind this property is that a typical pair of nodes are likely to be far in both spaces and so the framework should not pay too much attention to these pairs.
- An embedding algorithm might learn many node features, which may not contribute to the representation of particular structural feature. This additional learned information can be removed and minimized by adjusting the weights associated with appropriate embedding dimensions. For example, the feature *PageRank* may get mapped to dimension of 1 (out of 8) of an embedding space. In this case, dimensions 2 to 7 do not contribute to the representation of *PageRank* and can be removed by setting the weights for those dimensions to 0. This process is done automatically by our framework during the optimization process of the ψ value.

3 Experimentation

In this section, we investigate and analyze various desired algorithmic properties of our framework. We focus on six embedding algorithms, four structural ones (**LSME** [5], **Role2Vec** [2], **Struc2Vec** [22], and **RolX** [13]) and two classical ones (**Node2Vec** [9] and **DeepWalk** [21]). The goal of our analysis here is to understand and analyze various properties of our framework. In addition, we showcase how one could use our framework in applications such as node classification, by investigating the performance of a number of node and structural embedding algorithms. We break up our analysis into two main parts. First (Subsection 3.2), we explore some of the basic properties of our framework, such as algorithm stability and behaviour. Second (Subsection 3.3), we showcase the application of our framework in a node classification case study. In this section, we use the default hyper-parameters for every embedding algorithm.

3.1 Synthetic Graphs Design

For experiments in this section we use synthetically generated graph $\mathcal{G}$ which is composed of three structurally distinct sets of subgraphs. As shown in Figure 1, these subgraphs are labelled Web, Star and dStar. The Web and Star subgraphs each have three types of nodes (w0, w1 and w2) and (s0, s1, s2), while dStar subgraph has two types of nodes (ds0, ds1). The overall synthetic graph is created by joining N_w Web, N_s Star and N_{ds} dStar subgraphs by randomly creating links between w2, s2 and ds1 nodes. The edge creation process is as follows; from joined set of w2, s2 and ds1 nodes randomly select two nodes n_a and n_b. If $n_a \neq n_b$ and $e_{ab} \notin E$, where E is the set of edges of $\mathcal{G}$, then create an edge e_{ab}. Repeat this process until all w2, s2 and ds1 nodes are connected to at least one other node. Based on this description, we can fully define our synthetic graph using 8 parameters: $\mathcal{G}(\{N_w, N_s, N_{ds}\}, \{k_{w1}, k_{w2}\}, \{k_{s1}, k_{s2}\}, \{k_{ds1}\})$. Here, N_w, N_s, N_{ds} are the number of Web, Star and dStar subgraphs in the overall graph and, k_x correspond to the number of nodes in layer x of each subgraph. Each layer of the subgraphs is connected to the previous/next layers as shown in Figure 1. For example, $k_{w1} = 5$, based on Figure 1. In this section, we create a synthetic graph with the following parameters: $\mathcal{G}(\{N_w = 200, N_s = 200, N_{ds} = 200\}, \{k_{w1} = 5, k_{w2} = 10\}, \{k_{s1} = 5, k_{s2} = 10\}, \{k_{ds1} = 5\})$, resulting a synthetic graph with 7,600 nodes. We have chosen this structure for our synthetic graph to allow for a simple yet structurally distinct nodes to be used for our classification tasks. Since our framework is designed for structural embedding algorithms, we wanted to use synthetic graphs where nodes have known structural roles (ground-truth). As we shall show in Section 3.3, we use the synthetic graph described above to build classifiers for identifying root nodes S_0, and analyze each embedding algorithm's performance for this task.

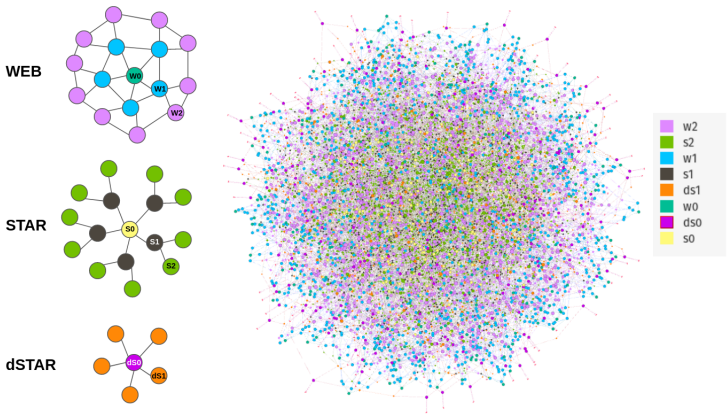

Fig. 1. Synthetic graph $\mathcal{G}(\{N_w = 200, N_s = 200, N_{ds} = 200\}, \{k_{w1} = 5, k_{w2} = 10\}, \{k_{s1} = 5, k_{s2} = 10\}, \{k_{ds1} = 5\})$ composed of a collection of Web, Star and dStar subgraphs.

3.2 Algorithmic Properties of the Framework

In this section, we analyze various algorithmic properties of our framework such as the convergence and stability of various metrics and the behaviour of structural vs. classical embedding algorithms. As we described in Section 2.2, the quality of learned representation of a structural feature (for example degree centrality) is measured using the post-optimization ψ value. The optimization is done by minimizing ψ as a function of weights associated with each embedding dimension. To test the effectiveness of this approach, we performed two experiments. In each experiment, we embed the synthetically created graph described in the previous section using either **LSME** or a fixed-embedding algorithm. Here, the fixed-embedding maps the simplest centrality measure, degree centrality, directly onto one of the N embedding dimensions. Furthermore, the other $N-1$ dimensions are filled with random numbers. This simulates a synthetic embedding algorithm, which learns a perfect representation of a feature and maps it onto one of the dimensions of the embedding space. For our experiments, we used $N = 8$ as the dimension of both embedding algorithms. Once the embedding vectors are generated, we use our framework to measure the performance of each embedding with respect to the degree centrality. In other words, we measure how well did each embedding learn the representation of degree centrality. The purpose for this experiment is to showcase the optimization process and highlight how our framework can be used to study how well features are mapped into the embedded space.

Figure 2 shows the results for the pre and post optimization values for the weights associated with each embedding dimension. In the pre-optimization state (top-left and bottom-left), the values for the weights are set randomly. The optimization algorithm then identifies the dimensions which the representation of the

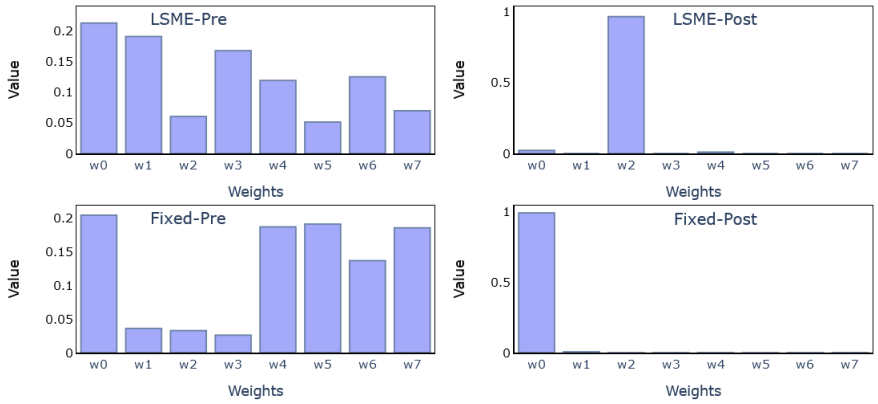

Fig. 2. Pre and post optimization values for weights associated with each embedding dimension (8 dimensional embedding). Top-left and bottom-left figures show pre-optimization random initialization of the weights for **LSME** embedding and fixed embedding, respectively. Top-right and bottom-right are the post-optimization values of the weight for **LSME** and fixed embedding.

degree centrality was mapped onto. As expected, the post-optimization weights for the fixed embedding is collapsed to only dimension 0 (w0), which holds a copy of degree centrality for the nodes. For the **LSME** algorithm, degree centrality was mapped by the embedding algorithm onto primarily dimension 2 and partially onto dimension 0 and 4. It is important to note that the post optimization weights by themselves are not complete measure of how well the embedding has learned the node feature. To capture the complete picture, one has to also consider the post optimization score ψ—more on this soon. The experiments in Figure 2 were repeated multiple times with randomized initial weights, and the results of the experiments were consistent with the above findings.

Before we dive deeper into the other properties of the framework, we want to consider the stability of the algorithm as a function of the node sample size (parameter c) and the number of clusters produced by k-means algorithm (parameter s). To apply the framework to large graphs, we would want our algorithm to converge for both $c \ll n$ and $s \ll n$, where n is the number of nodes of the graph. To measure the stability of the algorithm, we perform two experiments using **LSME** and **Role2Vec** on a synthetic graph. The performance of the embedding algorithms are measured using the degree centrality as the node feature. Figures 3 and 4 show the convergence of ψ as a function of the normalized number of clusters and the normalized sample size respectively, where normalization is done with respect to the size of the graph. Let us first consider the behaviour of ψ as a function of s, the number of clusters. It is important to note that while we vary s in this experiment, values for other parameters are kept as default. As we can see in Figure 3, ψ converges to its long-run average when the number of clusters is approximately 2% to 3% of the total size of the graph. Both here

and in Figure 4, the long-run average is defined as the expected value for ψ as sample size or number of clusters approaches the size of the graph. Finally, we conclude that our default value for the number of clusters ($s = \sqrt{n}$) is a good approximation since for the current experiment ($n = 1,000$) the number of clusters is approximately 3% of the size of the graph ($\frac{\sqrt{1,000}}{1,000} \approx 0.03$). Next, we look at the convergence of ψ as a function of c, the sample size. Similarly to the previous experiment, we vary c while setting other parameters to their default values. As one can see in Figure 4, the value for ψ converges for sample of sizes greater than or equal to 20% to 30% of the size of the graph. The results of our experiments point at two facts. First that the algorithm converges and is stable for both $c \ll n$ and $s \ll n$. Second, the default values for the hyperparameters are good and safe estimates.

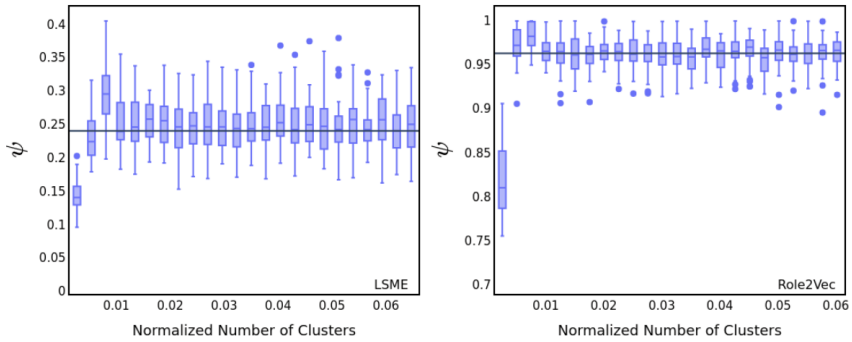

Fig. 3. Post-optimization score ψ as a function of normalized number of clusters. Normalization is with respect to the size of the graph. On the left/right ψ is computed for the degree centrality using the **LSME**/**Role2Vec** embedding algorithms, respectively. The horizontal lines are the long-run average of ψ.

We now turn our attention to experiments comparing the general behaviour of structural embedding as compared to classical node embedding algorithms. Classical node embedding algorithms such as **Node2Vec** [9] and **DeepWalk** [21] have difficulty learning structural properties of graphs. To showcase that our framework can be used as an unsupervised method for capturing this effect, we perform four experiments using two structural and two classical embedding algorithms. All four algorithms are ran against synthetic graphs (created using the procedure in Section 3.1) to generate 8-dimensional embedding vectors. Each embedding is then evaluated using our framework, where its performance is measured against 12 classical and widely used node features. For each node feature, we compute the post optimization ψ value. As we noted previously, the value of ψ is inversely correlated to how well the embedding was able to learn a given representation. In particular, $\psi = 1$ means that the embedding was not able to learn anything for a given feature, and in the other extreme

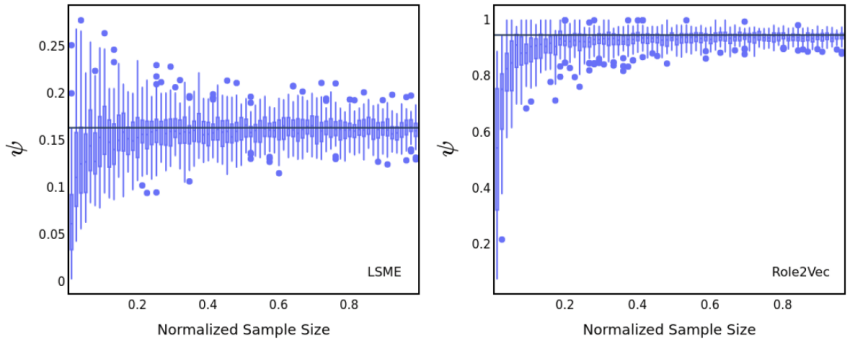

Fig. 4. Post-optimization score ψ as a function of normalized sample size. Normalization is with respect to the size of the graph. On the left/right ψ is computed for the degree centrality using the **LSME**/**Role2Vec** embedding algorithms, respectively. The horizontal lines are the long-run average of ψ.

$\psi = 0$ means that the embedding was able to learn prefect representation of the feature. With that said, let us consider the results presented in Figure 5. On the top-left and bottom-left, we show the post-optimization value for ψ as a function of various node features for **LSME** and **Role2Vec**, respectively. It is clear that these two embedding algorithms performed differently, as measure by our framework. The **LSME** algorithm performs much better than **Role2Vec**. It is important to note that we have chosen to use the default settings and parameters for each algorithm and did not perform any optimization. In addition, we have only focused on a set of 12 structural features, while algorithms could be learning features not in our set. While the structural embedding algorithm **LSME** was able to learn some structural node features, as expected, classical embedding algorithms (**Node2Vec** and **DeepWalk**) struggle with this task. This is clearly shown in the top-right and bottom right-plots of Figure 5, where the post-optimization ψ values for all node features are close to 1. Lastly, both **Node2Vec** and **DeepWalk** perform similarly to one another, indicating the similarity in the underlying algorithms.

3.3 Role Classification Case Study

In this subsection, we explore the use case of our framework for analyzing role classification in a synthetic network introduced earlier. A common task in network analysis is to classify nodes based on the role the nodes play in their local network structure. To build features for a classification algorithm, one could either use manually calculated structural properties of the nodes (node features) or leverage structural or node embedding features; as an automated way of learning various structural properties of nodes. One major challenge with using some embedding algorithms as a source for feature engineering, is the lack of explainability of the learned representations. It is not easy to identify what structural properties of the nodes are learned and how a given learned representation is

mapped onto the embedding space. To explore these ideas and showcase one possible use case of our framework, we consider the synthetic network introduced in Subsection 3.1, and use w0, s0 and ds0 root nodes as the target nodes we would like to classify. The root nodes considered here have very similar local structure, creating a relatively challenging tasks for a classifier. The goal of our analysis is to design and build a classifier using both node features and features extracted from various embedding algorithms. Lastly, we show how one could use our framework as an unsupervised technique to gain insight into the performance of embedding algorithms in applications such as role classification. We use six embedding algorithms, two classical algorithms (**Node2Vec** and **DeepWalk**) and four structural based ones (**LSME**, **Struc2Vec**, **RolX**, and **Role2Vec**). We hope to answer the following questions: can one use our framework to identify embedding algorithms that best learn various structural properties of nodes, which could hint at their performance in a role classification task? Additionally, can one extract insights into the predictability strength of each node feature and how those features are learned by a given embedding algorithm?

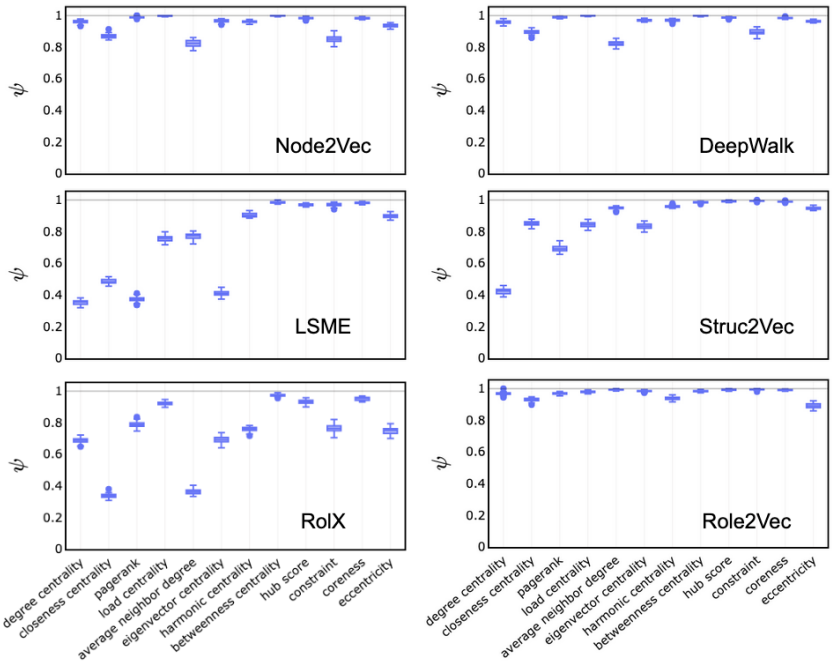

Fig. 5. Post-optimization ψ values (y-axis) computed as a function of 12 node features (x-axis) for various classical and structural embedding algorithms.

We first start by analyzing each embedding algorithm using our framework. We consider 12 node features as a benchmark and compute ψ for each feature.

The performance of each embedding algorithm is presented in Figure 5. As before, ψ is inversely proportional to how well an embedding algorithm has learned a given feature, where $\psi = 1$ means that a given feature was not learned by the algorithm. As we can see in Figure 5, classical embedding algorithms (**Node2Vec** and **DeepWalk**) fail to learn the structural properties of the graphs. This aligns with our expectations, since classical algorithms are designed to learn classical node properties. Furthermore, **Role2Vec** also fails to learn any structural proprieties of the graph, while **LSME**, **Struc2Vec**, and **RolX** perform quite well. It is important to note that we used the default hyper-parameters for each algorithm, and it is possible to achieve better results if one optimizes the learning process. Using current results, one would expect **LSME**, **Struc2Vec**, and **RolX** algorithms to perform better than **Node2Vec**, **DeepWalk**, and **Role2Vec** in classification tasks, where the structural properties of the nodes are of importance. With the results from Figure 5 in mind, we build 7 classifiers with the goal of classifying w0, s0 and ds0 nodes in graph G using features built using manually computed node features and features from each of our six embedding algorithms. The classifiers are trained to predict 3 classes, one for each root node (w0, s0 and ds0). Note that we do not mix features between embedding algorithms and node-features, in this analysis. For example, the accuracy of the classifier built using **DeepWalk**, in Figure 6, only includes features extracted from the embedding of the nodes by the **DeepWalk** algorithm.

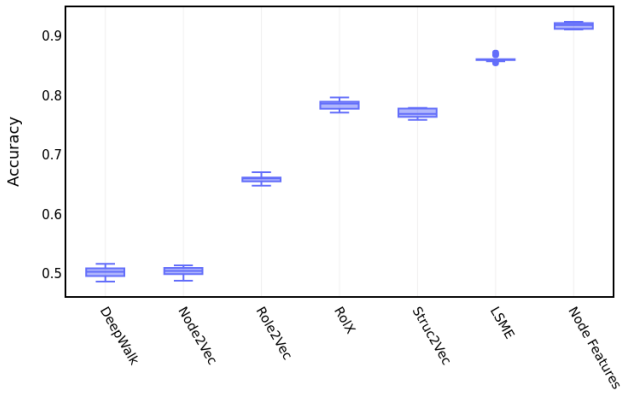

Fig. 6. Accuracy of 7 classifiers built using node and embedding features. Here, accuracy is measured as the combined accuracy of the following classes (w0, s0 and ds0).

The overall accuracy of each classifier is plotted in Figure 6. Here, accuracy is measured as the combined accuracy of the following classes (w0, s0 and ds0). For each embedding, we train 10 models and select the best performing

model and overage the performance across 10 samples. As expected, based on our analysis in Figure 5, classifiers built using node features, **Struc2Vec**, **RolX**, and **LSME** perform much better than those built using **Role2Vc**, **DeepWalk**, and **Node2Vec**. It is important to consider the following when analyzing these results. The fact that a classifier built using solely node feature performs well, indicates that any embedding that learns structural properties of the node should also perform well. However, this logic does not apply in reverse. The poor performance of an embedding based on our framework does not necessarily indicate that it will perform poorly in a classification task, since there may be features with predictive power which are not captured by the reference features of our framework. We note that, the set of reference features is modifiable and could be updated to include additional structural features. One could extend the 12 features in the benchmarking set to capture high order structural properties of the graph, to allow for a more extensive list of structural properties. Lastly, we point out that one could use the output of the framework to study the specific features learned by each embedding algorithm. For example, both **LSME** and **Struc2Vec** fail to learn *Constrain*, which is the measure of Burt's Constraint [8] for each node, (see Figure 5) as a structural feature, while **RolX** performs better in this regard. This is an important observation in scenarios where one would want to combine the features from different embedding algorithm to built feature sets with more predictive power. It is natural that each embedding algorithm learns slightly different properties of the graph. Our framework can be used as a tool to map out the embedding space and understand it through the lens of structural features of the graph.

4 Conclusion

In this work, we introduced an unsupervised embedding evaluation framework which can be used to both explain what structural properties of nodes embedding algorithms learn, in addition to how well each algorithm learns a particular structural feature. As we noted above, for tasks such as role-discovery or role-classification, one needs to rely on structural properties of nodes learned by structural embedding algorithms. However, there are numerous challenges with using structural embedding algorithms. First, there is a diverse set of structural features that an algorithm could learn. Therefore, it is not easy to define a single metric for measuring the performance of structural embedding algorithms. Second, measuring performance of embedding algorithms is often done using supervised techniques, which relies on the availability of labeled dataset.

In Section 2, we introduced a framework, which addresses the above two challenges. In our framework, we introduce a collection of core structural features, against which one could measure the performance of a structural embedding algorithm. In addition, we introduce a technique for performing these measurements in an unsupervised way, which avoids the need for the availability of labelled datasets. By introducing a mapping between the embedding and the feature space, we are able to define a metric (ψ) for measuring the performance

of embedding algorithms. In addition, we can use this metric to explain which features are learned by a given algorithm. As we have shown in Section 3.2, this feature of our framework is especially useful for the explainability of algorithms that rely on deep-learning such as **LSME**. Furthermore, using a synthetic graph as a benchmark, we showcased several use cases for our framework.

In Section 3.3, we showed that one could use our framework to measure the performance of a number of classical and structural embedding algorithms against a set of structural features. The performance of the embedding algorithms, as measured by ψ, correlates with the performance of the algorithms in a role classification task. This highlights the utility of our framework, which can be used to gain insights into the performance of embedding algorithms in scenarios where labeled data is not available. In addition, one could use our framework to identify difficult to embed structural features and use the ψ value as iterative way of modifying an embedding algorithm to learn specific feature-sets. The unsupervised framework developed and showcased in this work can be used as a versatile and useful tool for practitioners studying structural properties of complex networks.

References

1. Aggarwal, M., Murty, M.N.: Machine learning in social networks: embedding nodes, edges, communities, and graphs. Springer Nature (2020)
2. Ahmed, N.K., Rossi, R., Lee, J.B., Willke, T.L., Zhou, R., Kong, X., Eldardiry, H.: Learning role-based graph embeddings. arXiv preprint arXiv:1802.02896 (2018)
3. Akoglu, L., Tong, H., Koutra, D.: Graph based anomaly detection and description: a survey. Data mining and knowledge discovery **29**(3), 626–688 (2015)
4. Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., Murphy, K.: Machine learning on graphs: A model and comprehensive taxonomy. arXiv preprint arXiv:2005.03675 p. 1 (2020)
5. Dehghan, A., Kamiński, B., Prałat, P.: Node structural representation learning using local signature matrix embedding [lsme] (2022 (work in progress))
6. Dehghan-Kooshkghazi, A., Kamiński, B., Kraiński, Ł., Prałat, P., Théberge, F.: Evaluating node embeddings of complex networks. Journal of Complex Networks **10**(4), cnac030 (2022)
7. Donnat, C., Zitnik, M., Hallac, D., Leskovec, J.: Learning structural node embeddings via diffusion wavelets. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 1320–1329 (2018)
8. Everett, M.G., Borgatti, S.P.: Unpacking burt's constraint measure. Social Networks **62**, 50–57 (2020)
9. 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. pp. 855–864 (2016)
10. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584 (2017)
11. Hasan, M.A., Zaki, M.J.: A survey of link prediction in social networks. In: Social network data analytics, pp. 243–275. Springer (2011)
12. Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., Li, L.: Rolx: structural role extraction & mining in large

graphs. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1231–1239 (2012)
13. Henderson, K., Gallagher, B., Eliassi-Rad, T., Tong, H., Basu, S., Akoglu, L., Koutra, D., Faloutsos, C., Li, L.: Rolx: structural role extraction & mining in large graphs. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1231–1239 (2012)
14. Kamiński, B., Kraiński, Ł., Prałat, P., Théberge, F.: A multi-purposed unsupervised framework for comparing embeddings of undirected and directed graphs. Network Science (2022 (in press))
15. Kamiński, B., Prałat, P., Théberge, F.: A scalable unsupervised framework for comparing graph embeddings. In: International Workshop on Algorithms and Models for the Web-Graph. pp. 52–67. Springer (2020)
16. Kamiński, B., Prałat, P., Théberge, F.: An unsupervised framework for comparing graph embeddings. Journal of Complex Networks **8**(5), cnz043 (2020)
17. Kamiński, B., Prałat, P., Théberge, F.: Mining Complex Networks. Chapman and Hall/CRC (2021)
18. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
19. Neville, J., Jensen, D.: Iterative classification in relational data. In: Proc. AAAI-2000 workshop on learning statistical models from relational data. pp. 13–20 (2000)
20. Pankratz, B., Kamiński, B., Prałat, P.: Community detection supported by node embeddings. In: International Conference on Complex Networks and Their Applications. Springer (2022 (preprint))
21. 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. pp. 701–710 (2014)
22. Ribeiro, L.F., Saverese, P.H., Figueiredo, D.R.: struc2vec: Learning node representations from structural identity. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 385–394 (2017)
23. Stolman, A., Levy, C., Seshadhri, C., Sharma, A.: Classic graph structural features outperform factorization-based graph embedding methods on community labeling. In: Proceedings of the 2022 SIAM International Conference on Data Mining (SDM). pp. 388–396. SIAM (2022)