Symbolic Graph Embedding using Frequent Pattern Mining

Blaž Škrlj[1,2], Nada Lavrač[2,1,3], and Jan Kralj[2]

[1] Jožef Stefan International Postgraduate School
[2] Jožef Stefan Institute, Slovenia
[3] University of Nova Gorica, Slovenia

Abstract. Relational data mining is becoming ubiquitous in many fields of study. It offers insights into behaviour of complex, real-world systems which cannot be modeled directly using propositional learning. We propose Symbolic Graph Embedding (SGE), an algorithm aimed to learn symbolic node representations. Built on the ideas from the field of inductive logic programming, SGE first samples a given node's neighborhood and interprets it as a transaction database, which is used for frequent pattern mining to identify logical conjuncts of items that co-occur frequently in a given context. Such patterns are in this work used as features to represent individual nodes, yielding interpretable, symbolic node embeddings. The proposed SGE approach on a venue classification task outperforms shallow node embedding methods such as DeepWalk, and performs similarly to metapath2vec, a black-box representation learner that can exploit node and edge types in a given graph. The proposed SGE approach performs especially well when small amounts of data are used for learning, scales to graphs with millions of nodes and edges, and can be run on an of-the-shelf laptop.

Keywords: Graphs, machine learning, relational data mining, symbolic learning, embedding

1 Introduction

Many contemporary databases are comprised of vast, linked and annotated data, which can be hard to exploit for various modeling purposes. In this work, we explore how learning from heterogeneous graphs (i.e. heterogeneous information networks with different types of nodes and edges) can be conducted using the ideas from the fields of symbolic relational learning and inductive logic programming [16], as well as contemporary representation learning on graphs [2].

Relational datasets have been considered in machine learning since the early 1990s, where tools such as Aleph [27] have been widely used for relational data analysis. However, recent advancements in deep learning, a field of subsymbolic machine learning, which allows for learning from relational data in the form of graphs, was shown as useful for many contemporary relational learning tasks at scale, including recommendation, anomaly detection and similar [25]. The

state-of-the-art methodology exploits the notion of *node embeddings*—nodes, represented using real-valued vectors. As such, node embeddings can be simply used with propositional learners such as e.g., logistic regression or neural networks. The node embeddings, however, directly offer little to no insight into connectivity patterns relevant for representing individual nodes.

In this work we demonstrate that symbolic pattern mining can be used for learning *symbolic node embeddings* in heterogeneous information graphs. The main contributions of this work include:

1. An efficient graph sampler which samples based on a distribution of lengths of random walks, implemented in Numba, offering 15x faster sampling than a Python-native implementation, scaling to graphs with millions of nodes and edges on an of-the-shelf laptop.
2. Symbolic graph embedding (SGE), a symbolic representation learner that is explainable and achieves state-of-the-art performance for the task of node classification.
3. Evidence that symbolic node embeddings can perform comparably to black-box node embeddings, whilst requiring *less* space and data.

This paper is structured as follows. We first discuss the related work (Section 2), followed by the description of the proposed approach (Section 3), its computational and spatial complexity (Section 4), and its empirical evaluation (Section 5). We finally discuss the obtained results and potential further work (Section 6).

2 Related work

Symbolic representation learning has already been considered in the early 1990s in the inductive learning community, when addressing multi-relational learning problems through the so-called *propositionalization* approach [16]. The goal of propositionalization is to transform multi-relational data into real-valued vectors describing the individual training instances, that are a part of a relational data structure. The values of the vectors are obtained by evaluating a relational feature (e.g., a conjunct of conditions) as true (value 1) or false (value 0). For example, if all conditions of a conjunct are true, the relational feature is evaluated as true, resulting in value 1, and gets value 0 otherwise. We next discuss the approaches which were most influential for this work. The in-house developed Wordification [22] explores how relational databases can be unfolded into bags of relational words, used in the same manner as done in the area of natural language processing via Bag-of-words-based representations. Wordification, albeit very fast, can be spatially expensive, and was designed for SQL-based datasets. Our work was also inspired by the recently introduced HINMINE methodology [14], where Personalized PageRank vectors were used as the propositionalization mechanism. Here, each node is described via its probability to visit any other node, thus, a node of a netwrk is described using a distribution over the remainder of the nodes. Further, propositionalization has recently been explored

in combination with artificial neural networks [8], and as a building block of deep relational machines [6].

Frequent pattern mining is widely used for identifying interesting patterns in real-world transaction databases. Extension of this paradigm to graphs was already explored [12], using the Apriori algorithm [1] for the pattern mining. In this work we rely on the efficient FP-Growth [4] algorithm, which employs more structured counting compared to Apriori using fp-trees as the data structure. Frequent pattern mining is commonly used to identify logical patterns which appear above a certain e.g., frequency threshold. Efficiently mining for such patterns remains a lively research area on its own, and can be scaled to large computing clusters [11].

The proposed work also explores how a given graph can be sampled, as well as embedded efficiently. Many contemporary node representation learning methods, such as node2vec [9], DeepWalk [23], PTE [28] and metapath2vec [7], exploit such ideas in combination with e.g., the skip-gram model in order to obtain low-dimensional embeddings of nodes. Out of the aforementioned methods, only metapath2vec was adapted specifically to operate on *heterogeneous information networks*, i.e. graphs with additional information on node and edge types. It samples pre-defined meta paths, yielding type-aware node representations which serve better for classifying e.g., different research venues to topics.

Heterogeneous (non-attributed) graphs are often formalized as RDF triplets. Relevant methods, which explore how such triplets can be embedded are considered in [5], as well as in [24]. The latter introduced RDF2vec, a methodology for direct transformation of a RDF database to the space of real-valued entity embeddings. Understanding how such graphs can be efficiently sampled, as well as embedded into low-dimensional, real-valued vectors is a challenging problem on its own.

3 Proposed SGE algorithm

In this section we describe Symbolic Graph Embedding (SGE), a new algorithm for symbolic node embedding. The algorithm is summarized in Figure 1. The algorithm consists of two basic steps. First, for each node in an input graph, the neighborhood of the node is sampled (Section 3.2). Next, the patterns, emerging from the walks around a given node are transformed into a set of features whose values describe the node (Section 3.3). In this section, we first introduce some basic definitions and then explain both steps of SGE in more detail.

3.1 Overview and definitions

We first define the notions of a graph as used in this work.

Definition 1 (Graph). *A* graph *is a tuple* $G = (N, E)$*, where* N *is a set of nodes and* E *is a set of edges. The elements of* E *can be either subset* N *of size* 2 *(e.g.,* $\{n_1, n_2\} \subseteq N$*), in which case, we say the graph is* undirected.

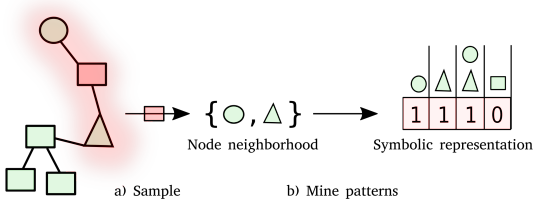

Fig. 1. Schematic represenatation of SGE. The red square's neighborhood (red highlight) is first used to construct symbolic features, forming a propositional graph representation (part a of the figure). The presence of various symbolic patterns (FP) is recorded and used to determine feature vectors for individual nodes (part b of the figure). The obtained representation can be used for subsequent data analysis tasks such as classification or visualization.

Alternatively, E can consist of ordered pairs of elements from N (e.g., $(n_1, n_2) \in N \times N$) – in this case, the graph is directed.

In this work we focus on directed graphs, yet the proposed methodology can also be extended to undirected ones. In this work we also use the notion of a *walk*.

Definition 2 (Walk). *Given a directed graph $G = (N, E)$, a* walk *is any sequence of nodes $n_1, n_2 \ldots, n_k \in N$ so that each pair n_i, n_{i+1} of consecutive nodes is connected by an edge, i.e. $(n_i, n_{i+k}) \in E$.*

Finally, we define the notion of node embedding as used throughout this work.

Definition 3 (Symbolic node embedding). *Given a directed graph $G = (N, E)$, a d-dimensional node embedding of graph G is a matrix $\mathcal{M}$ in a vector space $\mathbb{R}^{|N| \times d}$, i.e. $\mathcal{M} \in \mathbb{R}^{|N| \times d}$. Such embedding is considered* symbolic*, when each column represents a symbolic expression, which, when evaluated against a given node's neighborhood information, returns a real number representing a given node.*

We first discuss the proposed neighborhood sampling routine, followed by the description of pattern learning as used in this work.

3.2 Sampling node neighborhoods

Sampling a given node's local and global neighborhoods offers insights into connectivity patterns of the node with respect to the rest of the graph. Many contemporary methods resort to node neighborhood sampling for obtaining the node co-occurrence information. In this work we propose a simple sampling scheme

which produces a series of graph walks. The walks can be further used for learning tasks—in this work, we use them to produce node representations. Building on recent research ideas [17, 18], the proposed scheme consists of two steps: selection of walk sampling distribution and sampling. We first discuss the notion of distribution-based sampling, followed by the implemented sampling scheme.

Distribution-based sampling Our algorithm is based on the assumption that when learning a representation of a given node, nodes at various distances from the considered node are relevant. In order to use the information on neighborhood nodes, we sample several random walks starting at each node of the graph. Because real-world graphs are diverse, it is unlikely that the same sampling scheme would suffice for arbitrary graphs. To account for such uncertainty, we introduce the notion of *walk distribution vector*, a vector describing how many walks of a certain length shall be sampled. Let $w \in \mathbb{R}^s$ denote a vector of length s (a parameter of the approach). The i-th value of the vector corresponds to the proportion of walks of length i that are to be sampled. Note that the longest walk that can occur is of length s. For example consider the following vector w of length $s = 4$, $w = [0.2, 0, 0.5, 0.3]$. Assuming we sample e.g, 100 random walks, 20 walks will be of length one, zero of length 2, 50 of length three and 30 of length four. As w represents a probability distribution of different walk lengths to be sampled, $\sum_{i=1}^{s} w_i = 1$ must hold.

Having defined the formalism for describing the number of walks of different lengths, we have yet to describe the following two aspects in order to fully formalize the proposed sampling scheme: how to parametrize w and walk efficiently?

How to parametrize w We next discuss the considered initialization of the probability vector w. We attempt to model such vector by assuming a prior *walk length* distribution, from which we first sample ϕ samples—these samples represent different random walk *lengths*. In this work, we consider Uniform walk

Algorithm 1: Order-aware random walker.

Data: A graph $G = (N, E)$
Parameters : Starting node n_i, walk length s

$c \leftarrow n_i$;
$\delta \leftarrow 0$;
$\mathcal{W} \leftarrow$ multiset;
for $\alpha \in [1 \dots s]$ **do**
  $o := \text{Uniform}(N_G(c))$; ▷ `Select a random node.`
  $\mathcal{W} \leftarrow \mathcal{W} \cup (o, \alpha)$; ▷ `Store visited node.`
  $c \leftarrow o$;
end

Result: A random walk $\mathcal{W}$

length distribution, where the i-th element of vector w is defined as $\frac{1}{s}$, where s represents the length of w (maximum walk length).

The considered variant of graph sampling procedure does not take into account node or edge types. One of the purposes of this work is to explore whether such naïve sampling—when combined with symbolic learning—achieves good performance. The rationale for not exploring how to incorporate node and edge types is thus twofold: First, we explore whether symbolic learning, as discussed in the next section, detects heterogeneous node patterns on its own, as the node representations are discrete and could, as such, provide such information. Further, as exact node information is kept intact, and each node can be mapped to its type, the node types are implicitly incorporated. Next, we believe that by selecting the appropriate prior walk distribution w, node types can be to some extent taken into account (yet this claim depends largely on a given graph's topology).

How to walk efficiently An example random walker, which produces walks of a given length (used in this work) is formalized in Algorithm 1. Here, we denote with $N_G(n_i)$ the neighbors of the i-th node. The Uniform$(N_G(c))$ represents a randomly picked neighbor of a given node c, where picking each neighbor is equiprobable. We mark such picked node with o. Note that the walker is essentially a probabilistic depth-first search. The algorithm returns a list of tuples where each tuple (o, α) contains both the visited node o and the step α at which the node was visited. On line 6, we append to a current walk a tuple, comprised of a certain node and the overall walk length, it is a part of. Note that such inclusion of node IDs is suitable in a learning setting, where e.g., part of the graph's labels are not known and are to be predicted using the remainder of the graph. Even though inclusion of such information might seem redundant, we would like to remind the reader that the presented algorithm represents only a single random walker for a single walk length. In reality, multiple walkers yielding walks of different lengths are simulated, since including such positional (walk length) information can be beneficial for the subsequent representation learning step. The proposed Algorithm 1 represents a simple random walker. In practice, thousands of random walks are considered. As discussed, their lengths are distributed according to w. In theory, one could learn the optimal w by using e.g., stochastic optimization, yet we explore a different, computationally more feasible approach for obtaining a given w. We next discuss the notion of symbolic pattern mining and final formalization of the proposed SGE algorithm.

3.3 Symbolic pattern mining

In the previous section we discussed how a given node's neighborhood can be efficiently sampled. In this section we first discuss the general idea behind forming node representations, followed by a description of the frequent pattern mining algorithms employed.

Forming node representations Algorithm 1 outputs a multiset comprised of nodes, represented by (node id, walk order) tuples. Such multisets are in the following discussion considered as *itemsets*, as this is the terminology used in [4]. In the next step of SGE, we use the itemsets to obtain individual node representations. We first give an outline of this step, and provide additional details in the next section. The set of all nodes is considered as a transaction database, and the itemsets comprised of node id and walk order are used to identify frequent patterns (of tuples). Best patterns, selected based on their frequency of occurrence, are used as *features*. The way of determining the best patterns is approach specific, and is discussed in the following paragraphs. Feature values are determined based on the pattern identification method, and are either real-, natural- or binary-valued. Intuitively, they represent the presence of a given node pattern in a given node's neighborhood.

Frequent pattern mining We next discuss the frequent pattern mining approaches explored as part of SGE. The described approaches constitute the *findPatterns* method discussed in the next section. For each node, a multiset of (node ID, walk length) tuples is obtained. In each of the described approaches, the result is a transformation of the set of multisets, describing the network nodes, into a set of feature vectors describing these nodes.

Relational BoW. This paradigm leverages the Bag-of-words (BOW) constructors widely known in natural language processing [31]. Here, the tuples forming the itemsets, output by Algorithm 1, are considered as words. Thus, each word is comprised of a node and the order of a random walk in which that node was identified as connected to the node for which the representation is being constructed.
For the purpose of BOW construction, we consider the multiset of (node ID, walk length) tuples, generated by random walks that start at node n. This multiset is viewed as a "*node document*", consisting of individual words—i.e. the tuples contained in the multiset. The number of total features, d, is a parameter of the SGE algorithm. We consider the following variations of this paradigm for transforming each node "document" $\mathfrak{T}_n$ into one feature vector of size d:

- **Binary**. In a binary conversion, the values of the vector represent the presence or absence of a given tuple k-gram (a combination of k tuples; k is a free parameter of SGE) in the set of random walks associated with a given node document. The features, represented by such tuple k-grams, can have values of either 0 or 1.
- **TF**. Here, counts of a given k-gram t in a given node document $\mathfrak{T}_n$ are used as feature values ($TF_{t,\mathfrak{T}_n}$). The values are integers. Note that $TF_{t,\mathfrak{T}_n}$ represents the multiplicity of a given tuple k-gram in the multiset (node document).
- **TF-IDF**. Here, TF-IDF weighting scheme is employed to weight the values of individual features. The obtained values are real numbers. Given

a tuple k-gram t and the transaction database $\mathfrak{T}$, it is computed as:

$$\text{TF-IDF}(t,n) = (1 + \log \text{TF}_{t,\mathfrak{T}_n}) \cdot \log \frac{|N|}{\mathfrak{T}_t},$$

where $TF_{t,\mathfrak{T}_N}$ is the number of t's occurrences in a given document of node n and $\mathfrak{T}_t$ is the overall occurrence of this k-gram in the whole transaction database.

FP-Growth. This well known variant of association rule learning [4] constructs a specialized data structure termed fp-tree, which is used to count combinations of tuples of different lengths. It is more efficient than the well known Apriori algorithm [3, 21].
For the purpuse of FP-growth, the obtained multiset $\mathfrak{T}$ is viewed as a set of *itemsets* - a transaction database. For each itemset, only the set of unique tuples is considered as the input while their multiplicity is ignored. The FP-Growth algorithm next considers such non-redundant transaction databse $\mathfrak{T}$ to identify frequent combinations of tuples, similarly to the TF and TF-IDF schemes described above. The free parameter we consider in this work is *support*, which controls how frequent tuple combinations shall be considered. Similarly to TF and TF-IDF schemes, once the representative tuple combinations are obtained, they are considered as features, whose values are determined based on their presence in a given node document, and are binary (0 = not present, 1 = present). Note that some of these features may correspond to the features generated by TF-IDF, however, in the case of FP-growth, we allow sizes of tuple combinations to be arbitrary, rather than fixed to k. Also unlike the BOW approaches, the dimension of the constructed feature vectors constructed is not fixed but is controlled implicitly by varying the value of the *support* parameter.

SGE formulation The formulation of the whole approach is given in Algorithm 2. Here, first the sampling vector w is constructed. Next, the vector is traversed. The i-th component of vector w represents the number of walks of length i that will be simulated. For each component of w cell, a series of random walks (lines 6-8) is simulated, which produces sequences of nodes that are used to fill a node-level walk container $\mathcal{D}$. Thus, $\mathcal{D}$, once filled, consists of o sets representing individual random walks of length α. The walks are added into a single multiset, prior to being stored into the global transaction structure $\mathfrak{T}$. Once w is traversed, frequent patterns are found (line 11), where the transaction structure $\mathfrak{T}$ comprised of all node-level walks is used as the input. The findPatterns method in line 11 can be any method that takes a transaction database as input, the considered ones are discussed in the following section. The top most frequent d patterns are used as features, and represent the columns (dimensions) of the final representation $\mathcal{M}$. Here, the representNodes method (line 12) fills the values according to the considered weighting scheme (part of r)[4].

[4] Note that this method takes as input random walk samples for *all* nodes.

Algorithm 2: Symbolic graph embedding.

Data: A graph $G = (N, E)$
Parameters : Number of walk samples ν, sampling distribution η, pattern finder r, embedding dimensionality d, starting node n_i
Result: Symbolic node embedding M

$\tau :=$ generateSamplingVector(η, ν);
$\mathfrak{T} \leftarrow$ multiset;
for $o \in \tau$ **do**
  $\alpha \leftarrow o$'s index ; ▷ `Walk length.`
  $\mathcal{D} \leftarrow \{\}$;
  for $k \in [1 \dots o]$ **do**
    $\mathcal{D} \leftarrow \mathcal{D} \cup$ Walk(G, n_i, α) ; ▷ `Sample with Algorithm 1.`
  end
  $\mathfrak{T} \leftarrow \mathfrak{T} \cup \mathcal{D}$; ▷ `Update walk object.`
end
$\mathcal{P} \leftarrow$ findPatterns($\mathfrak{T}$, r) ; ▷ `Find patterns.`
$\mathcal{M} \leftarrow$ representNodes($\mathfrak{T}$, $\mathcal{P}$, r, d) ; ▷ `Represent nodes.`
return $\mathcal{M}$;

4 Computational and spatial complexity

In this section we discuss the computational aspects of the proposed approach. We split this section into two main parts, where we first discuss the complexity of the sampling, followed by the pattern mining part.

The time complexity of the proposed sampling strategy depends on the number of simulated walks and the walk lengths. The complexity of a single walk is linear with respect to the length of the walk. If we define the average walk length as $\bar{l}$, and the number of all samples as ν, the spatial complexity, required to store all walks amounts to $\mathcal{O}(|N| \cdot \nu \cdot \bar{l})$. As the complexity of a single random walk is linear with respect to the length of the walk, the considered sampling's time complexity amounts to $\mathcal{O}(\nu \cdot \bar{l})$ for a single node. The proposed approach is also linear with respect to the number of nodes both in space and time.

The computational complexity of pattern mining varies based on the algorithm used for this step. The considered FP-Growth's complexity is linear with respect to the number of transactions, whereas its spatial complexity is, due to efficient counting employed, similarly efficient and does not explode as for example with the Apriori family of algorithms. The result of the pattern mining step is a $|N| \times d$ matrix, where d is the number of patterns considered as features. Compared to e.g., metapath2vec and other shallow graph embedding methods, which yield a dense matrix, this matrix is *sparse*, and potentially requires orders of magnitude less space for the same d[5]. As storing large dense matrices can be spatially demanding, the proposed sparse feature representation requires less space, especially if high-dimensional embeddings are considered (the black-box

[5] In practice, however, larger dimensions are needed to represent the set of nodes well by using symbolic representations.

methods commonly yield dense representation matrices). The difference arises especially for very large datasets, where dense node representations can become a spatial bottleneck. We observe that $\approx 10\%$ of elements are non-zero, indicating that storing the feature space as a sparse matrix results in smaller time complexity. Worst case spatial complexity of storing the embedding, however, is for both types of methods $\mathcal{O}(|N| \cdot d)$.

5 Empirical evaluation

In this section we present the evaluation setting, where we demonstrate the performance of the proposed Symbolic Graph Embedding approach. We follow closely the evaluation introduced by metapath2vec, where the representation is first obtained, and next used for the classification task, where logistic regression is used as a classifier of choice. We test the performance on a heterogeneous information graph, comprised of authors, papers and venues[6]. The task is to classify venues into one of eight possible topics. The dataset was first used for evaluation of metapath2vec, hence we refer to the original results when comparing with the proposed approach. The considered graph consists of 2,766,148 nodes and 2,503,628 edges, where the 133 venues are to be classified into correct classes. We compare SGE against previously reported performances [7] of DeepWalk [23], LINE [29], PTE [28], metapath2vec and metapath2vec++ [7]. All methods are considered state-of-the-art for black-box node representation learning. The PTE and two variations of metapath2vec can take into account different (typed) paths during sampling.

We tested the following SGE variants. For pattern learning, we varied the TF-IDF, BoW and TF-, as well as the FP-Growth methods. The parameter search space used to obtain the results was as follows. The number of features = [500,1000,1500,2000,3000], considered vectorizers = ["TF-IDF","TF","FP-growth","Binary"], relation order (relevant for TF-based vectorizers—the highest k-gram order considered) = [2, 3, 4], walks of lengths = [2, 3, 5, 10], and number of walk samples = [1000, 10000] were considered. The support parameter of the FP-Growth parameter was varied in the range [3, 5, 8]. The Uniform walk length distribution was used. In addition to the proposed graph sampling (FS), a simple breadth-first search (BFS) that explores neighborhood of order two was also tested. We report the best performing learners' scores based on the type of the vectorizer and the sampling distribution. Ten repetitions of ten-fold, stratified cross validation is used, the resulting micro and macro F1 scores are averaged to obtain the final performance estimate. We report the performance of logistic regression classifier when varying the percentage of training data.

5.1 Results

In this section we discuss in detail the results for the node classification task. The results in Table 1 are presented in terms of micro and macro F1 scores, with

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

respect to training set percentage. We visualize the performance of the compared representations in Figure 2.

Table 1. Numeric results of the proposed SGE approach compared to the state-of-the-art approaches, presented in terms of micro and macro F1 scores, with respect to training set percentage. Best performing approaches are highlighted in green.

Method / Percentage	10%	20%	30%	40%	50%	60%	70%	80%	90%
Macro-F1									
DeepWalk/node2vec	0.140	0.191	0.280	0.343	0.391	0.442	0.478	0.496	0.446
LINE (1st+2nd)	0.463	0.701	0.847	0.895	0.920	0.931	0.947	0.941	0.947
PTE	0.170	0.654	0.830	0.894	0.921	0.935	0.951	0.953	0.949
metapath2vec	0.525	0.803	0.897	0.940	0.953	0.953	**0.970**	**0.968**	**0.967**
metapath2vec++	0.544	0.805	0.900	**0.947**	**0.958**	**0.956**	0.968	0.953	0.950
SGE (binary + FS)	**0.815**	**0.883**	**0.918**	0.919	0.922	0.937	0.942	0.931	0.950
SGE (TF + FS)	0.716	0.769	0.826	0.853	0.875	0.886	0.906	0.914	0.919
SGE (TF-IDF + FS)	0.364	0.114	0.121	0.03	0.354	0.353	0.363	0.148	0.146
SGE (FP-growth + FS)	0.523	0.684	0.712	0.771	0.785	0.815	0.801	0.816	0.838
SGE (Binary + BFS)	0.396	0.589	0.685	0.710	0.702	0.772	0.792	0.759	0.778
SGE (TF + BFS)	0.054	0.058	0.070	0.087	0.091	0.083	0.094	0.088	0.091
SGE (TF-IDF + BFS)	0.360	0.090	0.113	0.047	0.324	0.321	0.325	0.122	0.122
SGE (FP-growth + BFS)	0.400	0.547	0.586	0.591	0.594	0.646	0.587	0.609	0.553
Micro-F1									
DeepWalk/node2vec	0.214	0.249	0.327	0.379	0.409	0.463	0.498	0.526	0.529
LINE (1st+2nd)	0.517	0.716	0.846	0.895	0.920	0.933	0.950	0.956	0.957
PTE	0.427	0.688	0.837	0.895	0.924	0.935	0.955	0.967	0.957
metapath2vec	0.598	0.833	0.901	0.940	0.952	0.954	**0.973**	**0.982**	**0.986**
metapath2vec++	0.619	0.834	0.903	**0.946**	**0.958**	**0.957**	0.970	0.974	0.979
SGE (Binary + FS)	**0.815**	**0.880**	**0.918**	0.918	0.921	0.935	0.940	0.933	0.964
SGE (TF + FS)	0.718	0.771	0.824	0.850	0.872	0.881	0.900	0.911	0.921
SGE (TF-IDF + FS)	0.477	0.231	0.245	0.138	0.518	0.522	0.528	0.289	0.279
SGE (FP-growth + FS)	0.515	0.655	0.700	0.758	0.775	0.807	0.800	0.815	0.864
SGE (Binary + BFS)	0.388	0.557	0.657	0.692	0.678	0.750	0.785	0.759	0.821
SGE (TF + BFS)	0.148	0.150	0.155	0.168	0.175	0.167	0.172	0.159	0.193
SGE (TF-IDF + BFS)	0.461	0.195	0.226	0.144	0.469	0.467	0.478	0.252	0.250
SGE (FP-growth + BFS)	0.381	0.512	0.553	0.558	0.563	0.619	0.577	0.600	0.607

The first observation is that shallow node embedding methods, e.g., node2vec and LINE, do not perform as well as the best performing SGE variants (Binary with Uniform sampling). Further, we can observe that best performing SGE also outperforms metapath2vec and metapath2vec++, indicating that symbolic representations can (at least for this particular dataset) offer sufficient node description. The best performing SGE variant was the simplest one, with simple binary features obtained via fast sampling. Here, 10,000 walks were sampled and feature matrix of dimension 3000 was considered along with up to three-gram patterns. Finally, we visualized the embeddings by projecting them to 2D using the UMAP algorithm [19]. The resulting visualization, shown in Figure 3, shows that the obtained symbolic node embeddings maintain the class structure of the data.

5.2 Implementation details and reproducibility

In this section we discuss the details of the proposed SGE. The main part of the implementation is Python-based, where Numpy [30] and Scipy [13] libraries were used for efficient processing. The Py3plex library[7] was used to parse the heteroge-

[7] `https://github.com/SkBlaz/Py3plex`

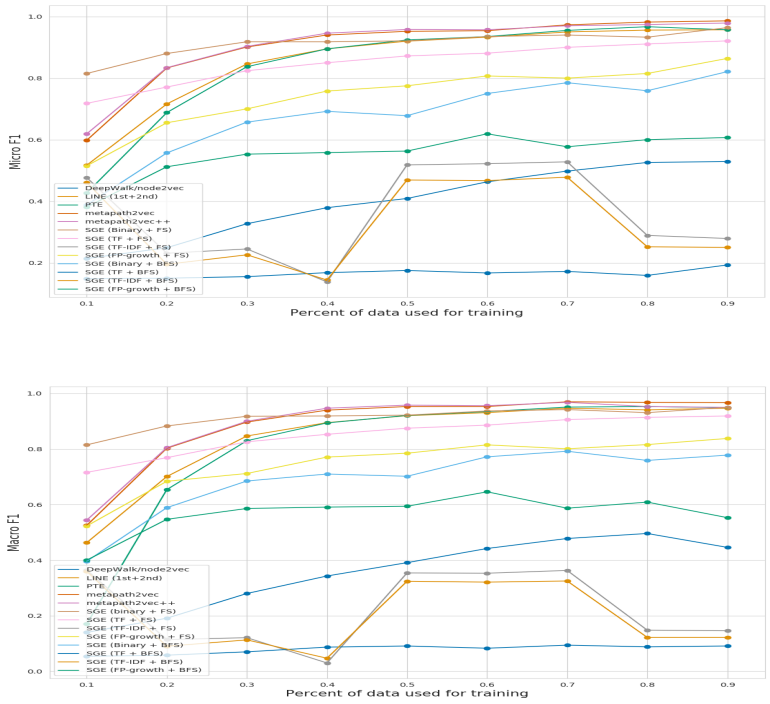

Fig. 2. Micro and macro F1 performance with respect to train percentages.

neous graph used as input [26]. The final graph was returned as a MultiDiGraph object compatible with NetworkX [10]. The TF, TF-IDF and Binary vectorizers implementations from the Scikit-learn library [20] were used. As the main bottleneck we recognized the graph sampling, which we further implemented using the Numba [15] framework for production of compiled code from native Python. After re-implementing the walk sampling part in Numba, we achieved approximately 15x speedup, which was enough to consider up to 10,000 walk samples of the large benchmark graph used in this work[8].

6 Discussion and conclusions

In this work we compared the down-stream learning performance of symbolic features, obtained by sampling a given node's neighborhood, to the performance of black-box learners. Testing the approaches on the venue classification task, we find that Symbolic Graph Embedding offers similar performance on a large, real-world graph comprised of millions of nodes and edges. The proposed method

[8] The code repository is available at `https://github.com/SkBlaz/SGE`

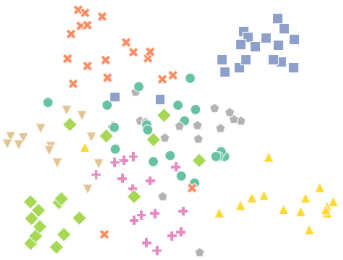

Fig. 3. UMAP projection of the best performing SGE embedding into 2D. Colors represent different types of venues (the class to be predicted). It can be observed that the obtained embeddings maintain the class-dependent structure, even though they were constructed in a completely unsupervised manner. The visualization was obtained using UMAP's default parameters.

outperforms the state-of-the-art shallow embeddings by up to ≈65%, and heterogeneous graph embeddings by up to ≈27% when only small percentages of the representation are used for learning (e.g., 10%). The method performs comparably to metapath2vec and metapath2vec++ when the whole embedding is considered for learning. One of the most apparent results is the well performing Binary + Uniform SGE, which indicates that simply checking the presence of relational features potentially offers enough descriptive power for successful classification. This result indicates that certain graph patterns emerge as important, where their presence or absence in a given node's neighborhood can serve as relevant for classification. The TF-IDF-based SGE variants performed the worst, indicating that more complex weighting schemes are not as applicable as in the other areas of text mining. We believe the proposed methodology could be further compared with RDF2vec and similar triplet embedding methods. The obtained symbolic embeddings were also explored qualitatively, where UMAP projection to 2D was leveraged to inspect whether the SGE symbolic node representations group according to their assigned classes. Such grouping indicates potential quality of the embedding, as venues of similar topics should be clustered together in the latent space.

7 Acknowledgements

We acknowledge the financial support from the Slovenian Research Agency through core research programmes P2-0103 and P6-0411 and project *Semantic Data Mining for Linked Open Data* (financed under the ERC Complementary Scheme, N2-0078). The authors have received funding also from the European Unions Horizon 2020 research and innovation programme under grant agreement No 825153 (EMBEDDIA). The work of the second author was funded by the

Slovenian Research Agency through a young researcher grant (BS). We would finally like to thank to Jan Kralj for his insightful comments on formulation of the proposed framework and mathematical proofreading.

Bibliography

[1] Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. 20th int. conf. very large data bases, VLDB. vol. 1215, pp. 487–499 (1994)
[2] Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)
[3] Borgelt, C.: Efficient implementations of apriori and eclat. In: FIMI03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations (2003)
[4] Borgelt, C.: An Implementation of the FP-growth Algorithm. In: Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations. pp. 1–5. ACM (2005)
[5] Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global rdf vector space embeddings. In: International Semantic Web Conference. pp. 190–207. Springer (2017)
[6] Dash, T., Srinivasan, A., Vig, L., Orhobor, O.I., King, R.D.: Large-scale assessment of deep relational machines. In: International Conference on Inductive Logic Programming. pp. 22–37. Springer (2018)
[7] Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 135–144. ACM (2017)
[8] França, M.V., Zaverucha, G., Garcez, A.S.d.: Fast relational learning using bottom clause propositionalization with artificial neural networks. Machine learning **94**(1), 81–104 (2014)
[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. ACM (2016)
[10] Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (SciPy) (1 2008)
[11] Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. Data mining and knowledge discovery **15**(1), 55–86 (2007)
[12] Inokuchi, A., Washio, T., Motoda, H.: An apriori-based algorithm for mining frequent substructures from graph data. In: European conference on principles of data mining and knowledge discovery. pp. 13–23. Springer (2000)
[13] Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–), `http://www.scipy.org/`

[14] Kralj, J., Robnik-Šikonja, M., Lavrač, N.: HINMINE: heterogeneous information network mining with information retrieval heuristics. Journal of Intelligent Information Systems **50**(1), 29–61 (Feb 2018)
[15] Lam, S.K., Pitrou, A., Seibert, S.: Numba: A llvm-based python JIT compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. p. 7. ACM (2015)
[16] Lavrač, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994)
[17] Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 631–636. ACM (2006)
[18] Maiya, A.S., Berger-Wolf, T.Y.: Sampling community structure. In: Proceedings of the 19th international conference on World wide web. pp. 701–710. ACM (2010)
[19] McInnes, L., Healy, J., Saul, N., Grossberger, L.: Umap: Uniform manifold approximation and projection. The Journal of Open Source Software **3**(29), 861 (2018)
[20] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. Journal of machine learning research **12**(Oct), 2825–2830 (2011)
[21] Perego, R., Orlando, S., Palmerini, P.: Enhancing the apriori algorithm for frequent set counting. In: International Conference on Data Warehousing and Knowledge Discovery. pp. 71–82. Springer (2001)
[22] Perovšek, M., Vavpetič, A., Kranjc, J., Cestnik, B., Lavrač, N.: Wordification: Propositionalization by unfolding relational data into bags of words. Expert Systems with Applications **42**(17-18), 6442–6456 (2015)
[23] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proc. of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
[24] Ristoski, P., Paulheim, H.: Rdf2vec: Rdf graph embeddings for data mining. In: International Semantic Web Conference. pp. 498–514. Springer (2016)
[25] Shi, C., Hu, B., Zhao, W.X., Philip, S.Y.: Heterogeneous information network embedding for recommendation. IEEE Transactions on Knowledge and Data Engineering **31**(2), 357–370 (2018)
[26] Škrlj, B., Kralj, J., Lavrač, N.: Py3plex: a library for scalable multilayer network analysis and visualization. In: International Conference on Complex Networks and their Applications. pp. 757–768. Springer (2018)
[27] Srinivasan, A.: The aleph manual (2001)
[28] Tang, J., Qu, M., Mei, Q.: Pte: Predictive text embedding through large-scale heterogeneous text networks. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1165–1174. ACM (2015)
[29] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web. pp. 1067–1077. International World Wide Web Conferences Steering Committee (2015)

[30] Walt, S.v.d., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Computing in Science & Engineering **13**(2), 22–30 (2011)

[31] Zhang, Y., Jin, R., Zhou, Z.H.: Understanding bag-of-words model: a statistical framework. International Journal of Machine Learning and Cybernetics **1**(1-4), 43–52 (2010)