Information Sciences 512 (2020) 935–951

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Improving network embedding with partially available vertex and edge content

Lin Lan[a], Pinghui Wang[b,a,*], Junzhou Zhao[a], Jing Tao[a], John C.S. Lui[c], Xiaohong Guan[a,b,d]

[a] *MOE NSKEY Lab, Xi'an Jiaotong University, China*
[b] *Shenzhen Research School, Xi'an Jiaotong University, China*
[c] *The Chinese University of Hong Kong, Hong Kong*
[d] *Department of Automation and NLIST Lab, Tsinghua University, China*

ARTICLE INFO

Article history:
Received 4 January 2018
Revised 17 September 2019
Accepted 30 September 2019
Available online 1 October 2019

Keywords:
Network embedding
Content information
Representation learning

ABSTRACT

Network embedding aims to learn a low-dimensional representation for each vertex in a network, which has recently shown its power in many graph mining problems such as vertex classification and link prediction. Most existing methods learn such representations according to network structure information, and some methods further consider vertex content in a network. Unlike prior works, we study the problem of network embedding with two distinctive properties: (1) content information exists on both vertices and edges; (2) only a part of vertices and edges have content information. To solve this problem, we propose a novel **P**artially available **V**ertex and **E**dge **C**ontent **B**oosted network embedding method, namely *PVECB*, which uses available vertex and edge content information to fine-tune structure-only representations through two hand-designed mechanisms respectively. Empirical results on four real-world datasets demonstrate that our method can effectively boost structure-only representations to capture more accurate proximities between vertices.

1. Introduction

Graphs (or networks) are widely used in modeling complex real-world systems such as online social networks (OSNs), biological networks, and communication networks, where entities in these systems are modeled as vertices, and entity relations are modeled as edges. Many practical problems are then converted to the corresponding graph mining problems, e.g., the friend recommendation in Facebook and the who-to-follow service in Twitter become classic link prediction problems on graphs [6,12,32]. In recent years, *network embedding* techniques [3,7,16,20] are gaining popularity and demonstrated to be effective in solving a wide scope of graph mining problems, including link prediction, vertex classification, and community detection. The basic idea behind network embedding is to design a mapping that maps vertices in a graph into a low-dimensional space while preserving meaningful structure information contained in the original graph, and the mapped vertices, called *vertex embedding vectors* or *vertex representations*, allow leveraging off-the-shelf machine learning algorithms to effectively solve aforementioned graph mining problems[1].

[*] Corresponding author.
E-mail address: phwang@mail.xjtu.edu.cn (P. Wang).
[1] In this paper, we use the term *embedding vector* and *representation* interchangeably.

https://doi.org/10.1016/j.ins.2019.09.083
0020-0255/© 2019 Elsevier Inc. All rights reserved.

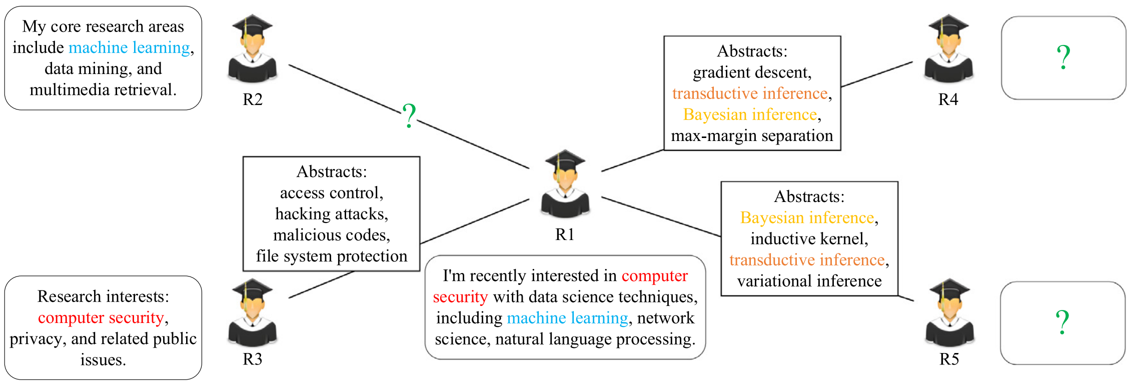

My core research areas include machine learning, data mining, and multimedia retrieval.
R2
?
Abstracts: gradient descent, transductive inference, Bayesian inference, max-margin separation
R4
?
Abstracts: access control, hacking attacks, malicious codes, file system protection
R1
Abstracts: Bayesian inference, inductive kernel, transductive inference, variational inference
Research interests: computer security, privacy, and related public issues.
R3
I'm recently interested in computer security with data science techniques, including machine learning, network science, natural language processing.
R5
?

Fig. 1. Illustration of a researcher social network. Two researchers are connected if they *are friends*, *follow each other*, or *have co-authored at least one paper*. Apart from the fundamental structure information, the network contains vertex content (i.e., text descriptions inside rounded rectangles, which can be obtained from a researcher's profile in ResearchGate or homepage) and edge content (i.e., papers collaborated by two researchers, of which only keywords of abstracts are shown inside rectangles for easy reading). The question marks indicate that the corresponding vertex and edge content are non-existent or unavailable, which almost always holds in practice. For instance, the incompleteness of vertex content occurs because R_4 and R_5 do not fill out profiles or do not have their own homepages, and edge content is non-existent since R_1 and R_2 follow each other but have not yet collaborated papers.

In the literature, most existing network embedding methods [3,16,20] mainly leverage the network structure information to design the mapping, and if two vertices are connected or have small shortest-path distance, the embedding methods tend to map them to two close locations in the low-dimensional space. Although embedding vectors learned by structure-only methods have been demonstrated to be useful in some applications, they do not consider *rich semantic information contained on vertices and edges* and may obtain inappropriate vertex representations. In real-world networks, besides structural information, vertices and edges in a network often contain meaningful semantic information such as attributes and tags, and the semantic information is sometimes important when determining vertex representations. For example, let us consider a simple researcher social network in ResearchGate in Fig. 1. Here, a vertex represents a researcher, and two researchers are connected if they are friends, follow each other, or have co-authored at least one paper. We notice that R_1 is an interdisciplinary researcher, who is interested in both *Computer Security* and *Machine Learning*. R_1 has a connection with R_2 who is interested in *Machine Learning*, and R_1 has another connection with R_3 who is interested in *Computer Security*. If we omit these vertex and edge semantics, and only consider structural information, structure-only embedding methods (such as the first-order proximity [20] that assumes all neighbors of a vertex are similar) will produce similar representations for vertices R_2 and R_3 in the low-dimensional space, implying that R_2 and R_3 have similar research interests, which violates our observation.

Present work. In this work, we investigate how to leverage structural information and content information to obtain better vertex representations. Using the example in Fig. 1, we briefly explain why both vertex content and edge content are helpful in obtaining better vertex representations. First, the content information on vertices R_2 and R_3 indicates that the two researchers actually belong to two different research communities, i.e., *Machine Learning* and *Computer Security*, respectively. Thus, their representations in the low-dimensional space should not be too close to each other. Second, although researchers R_4 and R_5 lack the vertex content, each of them has an edge with R_1 and the edge content indicates that they have a lot of common research interests, and hence their vertex representations in the low-dimensional space should be close to each other.

Inspired by the above observations, we come up with the idea of enhancing structure-only network embedding methods by leveraging the vertex and edge content information. However, we still face the following challenges.

•**Challenge 1: Missing vertex/edge content.** It is very common that some vertices or edges in a network do not have content in real-world networks. For example, some researchers may not provide their research interests (vertex content missing). Or, a researcher may follow some researchers but has not co-authored with them yet (edge content missing). How to handle such missing vertex/edge content remains difficult.

•**Challenge 2: Semantic gaps between vertex and edge content.** Usually, vertex and edge content describe characteristics of vertices within different contexts, and thus there is a semantic gap between them. For example, researchers tend to use some simple and general words to describe their research interests (vertex content), such as *Machine Learning* used by R_2. On the contrary, researchers need to elaborate their papers (edge content) using more technical and specialized words (e.g., *Bayesian inference* and *transductive inference*). The semantic gap between vertex and edge content will give rise to inaccurate relationships between vertices, and how to solve this issue needs to be further studied.

In this paper, we develop a novel method that jointly leverages both vertex and edge content. More importantly, our method allows incomplete vertex and edge content, and hence we refer the method as **P**artially available **V**ertex and **E**dge **C**ontent **B**oosted network embedding, namely *PVECB*. In particular, we address the above challenges from the following two perspectives correspondingly:

•**Fine-tuning with available content information.** To address the first challenge, we first initialize embedding vectors of vertices according to the structure information and then fine-tune structure-only embedding vectors using available content information on vertices and edges. To some extent, the philosophy of fine-tuning can alleviate the problem of the incompleteness of content information.

•**Different functionalities of vertex and edge content.** To address the second challenge, instead of gathering all content associated with a vertex together, we investigate the functionalities of vertex and edge content, and design two different objectives to fine-tune structure-only embedding vectors with them respectively.

Our main contributions are summarized as follows:

- We propose a novel network embedding method that jointly leverages vertex and edge content to learn vertex representations. Our method tolerates missing vertex/edge content which makes it applicable in realistic scenarios.
- We construct three real-world co-authorship networks, which include rich vertex and edge content that can be used to characterize relationships between vertices, and empirical results on these networks and another social network demonstrate that our method achieves a significant performance improvement (up to around 18%) over the best-performed structure-only method in terms of Micro-F_1 scores. On the contrary, existing content-enhanced methods, such as TADW [30] and TriDNR [15], fail to handle the missing content information, and even their performance is worse than the best-performed structure-only method.

The remainder of this paper will proceed as follows. Section 2 summarizes related work. Section 3 introduces some background knowledge. Section 4 formulates the studied problem and introduces the proposed method *PVECB* in detail. Section 5 presents experimental results, and Section 6 concludes.

2. Related work

In this section, we summarize the related literature of network embedding methods from two aspects: methods that only use network structure information, and methods that also use auxiliary information such as content information and label information.

2.1. Methods only using network structure information

Spectral methods for dimensionality reduction such as MDS [5], IsoMap [23], LLE [18], and Laplacian Eigenmap [2] learn vertices' low-dimensional representations by solving the leading eigenvectors of a graph (given in advance or learned from available data), which is computationally expensive for large networks. Inspired by recent progress in embedding words in the field of natural language processing (NLP), several methods are recently proposed to learn networks' vertex embedding vectors. For example, DeepWalk [16] converts network structure into sequences of vertices by a truncated random walk algorithm and learns vertex embedding vectors with the skip-gram model [13] which has been successfully applied to learn word embedding vectors. Node2vec [7] modifies DeepWalk to transform graph structure into vertex sequences by mixing two random walk strategies: breadth- and depth-first searches. LINE [20] learns vertex embedding vectors with the intention of preserving vertices' first- and second-step neighborhood information. GraRep [3] investigates k-step subgraph structure in a graph and further improves LINE by considering different k-step subgraph patterns. SDNE [27] is proposed to capture highly non-linear network structure using a multi-layer neural network. [28] develops a modularity-based method to learn vertex embedding vectors preserving network community structure. [17] proposes to model the characteristic of structure identity in a network during the process of embedding, which captures the proximity between two vertices that have structurally similar neighborhoods but are far apart in the network (i.e., two vertices have a long distance). In addition, [14] observes that the above network embedding methods cannot well preserve the asymmetric transitivity in directed graphs, and develops a method HOPE to solve the problem. However, all the above methods ignore content information in a network.

2.2. Methods also using auxiliary information

Some works are proposed to enhance structure-only embedding methods with content information. TADW [30] learns vertex representations from both network structure and text information on vertices based on matrix factorization. TriDNR [15] exploits structure information, vertex content, and vertex labels to jointly learn vertex representations. [11] combines variational auto-encoders [8] and doc2vec [9] to preserve network structure and vertex content simultaneously. However, these content-enhanced methods assume that the content information is available for each vertex, and their performance usually degrades significantly when there is only partially available content information.

Recently, SINE [31] regards content information as a kind of contextual vertices, and applies the skip-gram model [13] to learn vertex representations such that vertices with similar content context have similar representations. In effect, SINE uses embedding vectors of vertices to predict their corresponding content information. To some extent, this way could eliminate the negative impacts caused by missing content information. However, simply regarding content information (e.g., text) as separate contextual vertices (e.g., words) ignores high-level similarities (e.g., the semantic similarity) between content information. On the contrary, our proposed fine-tuning mechanisms are able to flexibly utilize advanced deep learning techniques to capture comprehensive content similarities and further guides the learning of vertex representations.

On the other hand, to the best of our knowledge, no methods have been developed to investigate how to incorporate edge content into network embedding. A relevant work [4] considers edge labels in the process of learning representations and models edge labels in a way similar to SINE. It constructs edge representations by concatenating embedding vectors of endpoints and then uses edge representations to predict the corresponding edge labels. However, when dealing with edge content, similar to SINE, it also has the limitation that high-level similarities between edge content cannot be captured.

3. Preliminaries

In this section, we first give some background knowledge about structure-only network embedding methods and then analyze the drawbacks of the existing content-enhanced methods. Finally, we introduce a kind of deep learning technique, which is employed to extract features from content information in this paper.

3.1. Structure-only network embedding methods

We first formally define the problem of network embedding, and then briefly review two state-of-the-art structure-only network embedding methods, LINE [20] and GraRep [3], for consistency.

A network is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, \dots, v_{|\mathcal{V}|}\}$ is a set of vertices, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. We assume $\mathcal{G}$ is undirected for ease of presenting our idea. We define $\mathcal{D} = \{(v_k, v_i, v_j) | v_k, v_i, v_j \in \mathcal{V}, (v_k, v_i) \in \mathcal{E}, (v_k, v_j) \in \mathcal{E}\}$ to represent all triplets in a network, and define $\mathcal{N}_k = \{v_i | v_i \in \mathcal{V}, (v_k, v_i) \in \mathcal{E}\}$ as the set of neighbors of v_k. The goal of network embedding is to learn a mapping function $f : \mathcal{V} \mapsto \mathbb{R}^d$ where $d \ll |\mathcal{V}|$. In the latent space $\mathbb{R}^d$, graph properties, such as first- and second-order proximities [20], are expected to be preserved as much as possible.

LINE [20] is an effective and efficient network embedding method, which models the first- and second-order proximities between vertices. Specifically, LINE uses a logistic function to approximate the first-order proximity between two connected vertices:

$$p_{1st}(v_i, v_j) = \frac{1}{1 + \exp(-\mathbf{u}_i^{\top} \mathbf{u}_j)},$$

where $\mathbf{u}_i, \mathbf{u}_j \in \mathbb{R}^d$ are embedding vectors of vertices v_i and v_j respectively. LINE preserves the first-order proximity over the entire network by optimizing the following objective:

$$\max_{\mathbf{U}} \sum_{(v_i, v_j) \in \mathcal{E}} \log p_{1st}(v_i, v_j) + \sum_{(v_i, v_j) \notin \mathcal{E}} \log \left(1 - p_{1st}(v_i, v_j)\right), \tag{1}$$

where $\mathbf{U} \in \mathbb{R}^{|V| \times d}$ indicates embedding vectors of all vertices.

For the second-order proximity, LINE introduces a context representation $\mathbf{u}'_k$ for each vertex $v_k \in V$, and defines the probability that a randomly selected vertex v_j is connected to vertex v_i as:

$$p_{2nd}(v_j | v_i) = \frac{\exp\left(\mathbf{u}_j'^{\top} \mathbf{u}_i\right)}{\sum_{k=1}^{|\mathcal{V}|} \exp\left(\mathbf{u}_k'^{\top} \mathbf{u}_i\right)}.$$

Therefore, we can use $p_{2nd}(\cdot | v_i)$ to represent the distribution of neighbors of vertex v_i. To preserve the neighbor structure of each vertex $v_i \in \mathcal{V}$ (i.e., the second-order proximity), LINE optimizes the following objective function:

$$\max_{\mathbf{U}} \sum_{v_i \in \mathcal{V}} \left(\sum_{v_j \in \mathcal{N}_i} \log p_{2nd}(v_j | v_i) + \sum_{v_j \notin \mathcal{N}_i} \log(1 - p_{2nd}(v_j | v_i)) \right). \tag{2}$$

GraRep [3] extends LINE to investigate high-order proximities in a network, and formulates the process of network embedding as matrix factorization. Specifically, GraRep defines the following probability transition matrix to represent the first-order proximity between vertices:

$$A = D^{-1} S,$$

where D and S are the degree matrix and adjacency matrix of a graph respectively. Furthermore, the k^{th}-order proximity can be represented as a matrix:

$$A^k = \underbrace{A \cdots A}_{k}.$$

Following [10], GraRep defines a matrix $Y^k \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ to integrate negative sampling, where

$$Y_{i,j}^k = \max\left(\log\left(\frac{A_{i,j}^k}{\sum_t A_{t,j}^k} \right) - \log\left(\frac{K}{|\mathcal{V}|} \right), 0 \right),$$

and K is the number of negative samples. To obtain vertex representations $\mathbf{U}^k$ that capture the k^{th}-order proximity in the graph, we factorize the matrix Y^k into two matrices $W^k \in \mathbb{R}^{|\mathcal{V}| \times d}$ and $H^k \in \mathbb{R}^{d \times |\mathcal{V}|}$, where the i-th row $W_{i,*}^k \in \mathbb{R}^d$ of W^k is the low-dimensional representation of v_i in the context of the k^{th}-order proximity.

3.2. TADW Is not good

As alluded before, the existing content-enhanced methods fail to handle the circumstance where only part of vertices have content information. Here, we analyze TADW [30] for illustration. Specifically, TADW optimizes the objective

$$\min_{W,H} \|M - W^{\mathsf{T}}HT\|^2,$$

where $M \in \mathbb{R}^{|V|\times|V|}$ is a transition probability matrix derived from the adjacency matrix, $T \in \mathbb{R}^{f_t\times|V|}$ is the text feature matrix, and $W \in \mathbb{R}^{d\times|V|}$ and $H \in \mathbb{R}^{d\times f_t}$ are the expected latent features. The final embedding matrix is the concatenation of W and HT. In the scenario of missing content information, the columns corresponding to vertices without content will have all zero entries, which will be problematic during optimization. In particular, for each vertex v_i without content, the corresponding column $T_{*,i}$ has all zero entries. Thus, no matter how W and H are updated, the i-th column $M'_{*,i}$ of the reconstructed matrix $M' = W^{\mathsf{T}}HT$ will also have all zero entries, which means vertex v_i has no connections with other vertices and implies the learned representation of vertex v_i cannot preserve connections between vertex v_i and other vertices. Furthermore, the experimental results in Section 5.4 confirm that the existing content-enhanced methods, especially TADW [30], will lead to a severe inconsistency between the multi-label classification performance of vertices with and without content.

3.3. Stacked denoising auto-Encoders

In this paper, we apply stacked denoising auto-encoders (SDAE) [25] to learn latent representations from text-related content. SDAE is an unsupervised deep learning model comprising of "Encoders" and "Decoders". The "Encoders" is a multi-layer neural network and aims to map high-dimensional corrupted input signals into a low-dimensional latent space. Given a $\frac{K}{2}$-layer "Encoders", we regard the bag-of-words representation of a piece of textual information $\mathbf{x}_{\mathcal{T}}$ as the input signal, and obtain the corresponding latent representationq $\mathbf{u}_{\mathcal{T}}$ as follows:

$$\begin{aligned}
\mathbf{y}_{\mathcal{T}}^{(1)} &= f\left(\mathbf{W}^{(1)}(\mathbf{x}_{\mathcal{T}} + \epsilon) + \mathbf{b}^{(1)}\right),\\
\mathbf{y}_{\mathcal{T}}^{(k)} &= f\left(\mathbf{W}^{(k)}\mathbf{y}_{\mathcal{T}}^{(k-1)} + \mathbf{b}^{(k)}\right), k = 2, \dots, \frac{K}{2},\\
\mathbf{u}_{\mathcal{T}} &= \mathbf{y}_{\mathcal{T}}^{(\frac{K}{2})},
\end{aligned}$$

where ϵ is the noise signal, and $\mathbf{W}_{\mathcal{T}}^{(k)}$, $\mathbf{b}_{\mathcal{T}}^{(k)}$ and $\mathbf{y}_{\mathcal{T}}^{(k)}$ are the weight matrix, bias term and output of k^{th} neural layer respectively.

On the other hand, the "Decoders" is also a multi-layer neural network, which reverses the calculation of "Encoders", and aims to reconstruct the original input signal according to latent representations learned by the "Encoders" such that the learned latent representations can preserve the characteristics of the original input signals. Given the latent representation $\mathbf{u}_{\mathcal{T}}$, a $\frac{K}{2}$-layer "Decoders" is defined as follows:

$$\begin{aligned}
\mathbf{y}_{\mathcal{T}}^{(\frac{K}{2}+1)} &= f\left(\mathbf{W}^{(\frac{K}{2}+1)}\mathbf{u}_{\mathcal{T}} + \mathbf{b}^{(\frac{K}{2}+1)}\right),\\
\mathbf{y}_{\mathcal{T}}^{(k)} &= f\left(\mathbf{W}^{(k)}\mathbf{y}_{\mathcal{T}}^{(k-1)} + \mathbf{b}^{(k)}\right), k = \frac{K}{2} + 1, \dots, K.
\end{aligned}$$

The reconstruction loss is calculated by

$$\mathcal{L}_{\mathcal{T}} = \|\mathbf{y}_{\mathcal{T}}^{(K)} - \mathbf{x}_{\mathcal{T}}\|^2.$$

Further, in order to capture the similarities between different pieces of text, we define the following objective function:

$$\mathcal{O}_{\mathcal{T}} = \sum_{\mathbf{x}_{\mathcal{T}}} \|\mathbf{y}_{\mathcal{T}}^{(K)} - \mathbf{x}_{\mathcal{T}}\|^2. \tag{3}$$

Although Eq. (3) does not explicitly model the similarity between two pieces of text, the work [19] has shown that minimizing the reconstruction loss of different input signals can make latent representations of similar input signals locate close in the latent space. Thus, the latent content representations learned by minimizing Eq. (3) can be used to measure similarities between text.

4. Our proposed approach

In this section, we first formulate our studied problem. Then, we describe *PVECB* in detail, and discuss the model optimization.

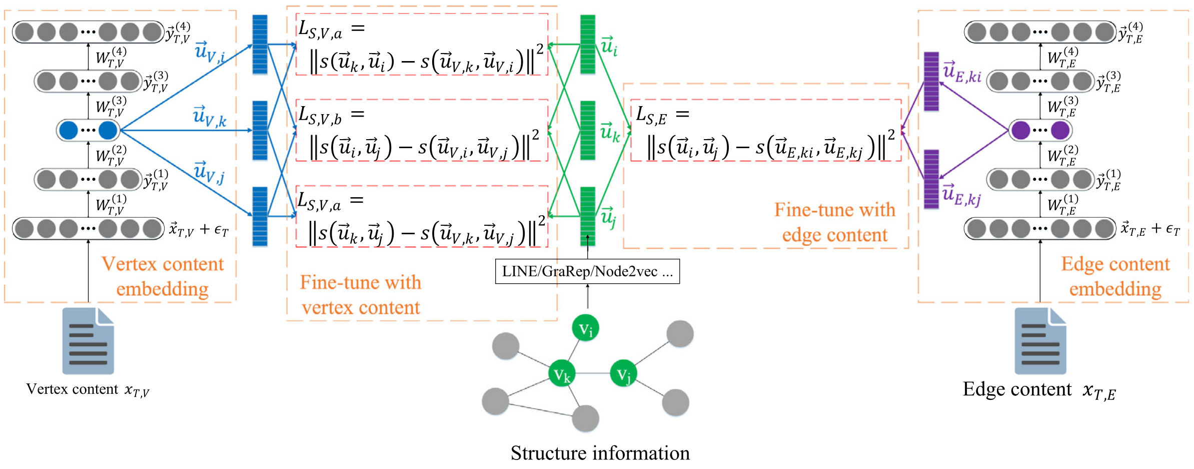


Fig. 2. Illustration of our model PVECB. For each triplet (v_k, v_i, v_j), where v_i and v_j are connected with v_k, PVECB first employs two stacked denoising auto-encoders ($K = 4$ for illustration) to extract content embedding vectors from vertex and edge content respectively as shown on two sides of the figure. Then, PVECB uses extracted vertex content embeddings $\mathbf{u}_{V,k}$, $\mathbf{u}_{V,i}$, and $\mathbf{u}_{V,j}$ to fine-tune structure-only embedding vectors $\mathbf{u}_k$, $\mathbf{u}_i$, and $\mathbf{u}_j$. Similarly, PVECB fine-tunes $\mathbf{u}_i$ and $\mathbf{u}_j$ with edge content embeddings $\mathbf{u}_{E,ki}$ and $\mathbf{u}_{E,kj}$.

4.1. Our problem

We denote by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{C}_\mathcal{V}, \mathcal{C}_\mathcal{E})$ the network studied in this paper, where $\mathcal{C}_\mathcal{V}$ and $\mathcal{C}_\mathcal{E}$ represent content information on vertices and edges respectively. Let $c_{v,i}$ denote the content information on the vertex v_i. For each edge $(v_i, v_j) \in \mathcal{E}$, let $c_{e,ij}$ denote the content information on the edge (v_i, v_j). Our goal is to enhance traditional structure-only embedding methods by leveraging available content information on vertices and edges and learn a better representation $\mathbf{u}_i \in \mathbb{R}^d$ for each vertex $v_i \in V$.

4.2. Overview of our model PVECB

As shown in Fig. 2, PVECB mainly consists of two stages: 1) content embedding vector extraction, i.e., vertex content embedding vectors and edge content embedding vectors; 2) fusion of structure-only embedding vectors and content embedding vectors. In the former step, we apply a kind of deep learning technique, namely stacked denoising auto-encoder (SDAE) [25], to extract latent features from vertex and edge content. In the latter, to alleviate the problem of missing vertex/edge content in reality (as mentioned in Section 1), we propose to use vertex content and edge content embedding vectors to fine-tune structure-only embedding vectors, and regard fine-tuned structure-only embeddings as the final vertex embedding vectors. The fusing step makes three kinds of information (e.g., structure information, vertex content, and edge content) complementary to each other, such that the learned vertex embedding vectors can capture more accurate characteristics of vertices and relationships between vertices in a network. Note that the modeling of structure information is not the key point of our paper, and we just apply two kinds of state-of-the-art methods, LINE [20] and GraRep [3], to learn structure-only embedding vectors.

4.3. Fusion of structure-only and content embedding vectors

The key point of this paper is to use available content information to assist structure information during the procedure of network embedding, and thus we next first elaborate how to fuse structure-only embedding vectors and content embedding vectors in this section. Given a triplet $(v_k, v_i, v_j) \in \mathcal{D}$, where v_i and v_j are connected with v_k, suppose we have obtained the corresponding structure-only embedding vectors, vertex content embedding vectors, and edge content embedding vectors. We propose to use vertex and edge content embedding vectors (i.e., $\mathbf{u}_{\mathcal{V},k}$, $\mathbf{u}_{\mathcal{V},i}$, $\mathbf{u}_{\mathcal{V},j}$, $\mathbf{u}_{\mathcal{E},ki}$, and $\mathbf{u}_{\mathcal{E},kj}$) to fine-tune structure-only embeddings (i.e., $\mathbf{u}_k$, $\mathbf{u}_i$, and $\mathbf{u}_j$), such that the fine-tuned structure-only embedding vectors can characterize vertices of a network more accurately.

4.3.1. Fine-tune with vertex content embedding vectors

Vertex content can be used to fine-tune structure-only embeddings from two aspects: (**aspect a**) For two connected vertices v_k and v_i, the similarity between their vertex content can reflect the connection weight between them. That is, the more similar vertex content two connected vertices show, the closer their embedding vectors should be in the latent space; (**aspect b**) For two vertices v_i and v_j connecting to the same vertex v_k, the similarity between their vertex content identifies whether they are similar or not. Based on the above principles, we first define the similarity between two vectors as:

$$s(\mathbf{m},\mathbf{n}) = \frac{\mathbf{m}^{\mathsf{T}}\mathbf{n}}{\|\mathbf{m}\|\,\|\mathbf{n}\|}. \tag{4}$$

Consider the above aspect (a), for each triplet (v_k, v_i, v_j), we define the following loss function:

$$\mathcal{L}_{\mathcal{S},\mathcal{V},a} = \|s(\mathbf{u}_k,\mathbf{u}_i) - s\big(\mathbf{u}_{\mathcal{V},k},\mathbf{u}_{\mathcal{V},i}\big)\|^2 + \|s\big(\mathbf{u}_k,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{V},k},\mathbf{u}_{\mathcal{V},j}\big)\|^2. \tag{5}$$

On the other hand, the loss function of the aspect (b) is defined as:

$$\mathcal{L}_{\mathcal{S},\mathcal{V},b} = \|s\big(\mathbf{u}_i,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{V},i},\mathbf{u}_{\mathcal{V},j}\big)\|^2. \tag{6}$$

Furthermore, structure-only embedding vectors can be fine-tuned with all available vertex content by minimizing the following objective function:

$$\begin{aligned}\mathcal{O}_{\mathcal{S},\mathcal{V}} &= \mathcal{O}_{\mathcal{S},\mathcal{V},a} + \alpha\mathcal{O}_{\mathcal{S},\mathcal{V},b}\\ &= \sum_{(v_k,v_i,v_j)\in\mathcal{D}} \mathbf{1}\big(c_{v,k}\&c_{v,i}\big)\|s(\mathbf{u}_k,\mathbf{u}_i) - s\big(\mathbf{u}_{\mathcal{V},k},\mathbf{u}_{\mathcal{V},i}\big)\|^2\\ &\quad + \mathbf{1}\big(c_{v,k}\&c_{v,j}\big)\|s\big(\mathbf{u}_k,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{V},k},\mathbf{u}_{\mathcal{V},j}\big)\|^2\\ &\quad + \alpha\mathbf{1}\big(c_{v,i}\&c_{v,j}\big)\|s\big(\mathbf{u}_i,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{V},i},\mathbf{u}_{\mathcal{V},j}\big)\|^2,\end{aligned} \tag{7}$$

where α is a hyper-parameter that balances the effects of two aspects of fine-tuning with vertex content, and $\mathbf{1}(x)$ is the indicator function that equals one when the predicate x is true (e.g., $\mathbf{1}(c_{v,k})$ equals one when the vertex content of v_k is available, and $\mathbf{1}(c_{v,k}\&c_{v,i})$ equals one if the vertex content of both v_k and v_i is available) and equals zero otherwise.

4.3.2. Fine-tune with edge content embedding vectors

Edge content represents characteristics of pair-wise interactions (preferences) between vertices. For two vertices v_i and v_j, each of which is connected with v_k, if v_k has similar preferences with v_i and v_j, we can infer that v_i and v_j also have similar preferences. On the contrary, it is more possible that v_i and v_j belong to different social circles of v_k, and the preferences of v_i and v_j are different.

Based on the above principle, given a triplet (v_k, v_i, v_j), it is expected that the embedding vectors of v_i and v_j should locate close in the latent space if v_k has similar edge content with them (i.e., $c_{e,ki}$ and $c_{e,kj}$ are similar), and vice versa. Thus, for each triplet (v_k, v_i, v_j), we define the following loss function:

$$\mathcal{L}_{\mathcal{S},\mathcal{E}} = \|s\big(\mathbf{u}_i,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{E},ki},\mathbf{u}_{\mathcal{E},kj}\big)\|^2. \tag{8}$$

Further, we define the following objective function:

$$\mathcal{O}_{\mathcal{S},\mathcal{E}} = \sum_{(v_k,v_i,v_j)\in\mathcal{D}} \mathbf{1}\big(c_{e,ki}\&c_{e,kj}\big)\|s\big(\mathbf{u}_i,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{E},ki},\mathbf{u}_{\mathcal{E},kj}\big)\|^2, \tag{9}$$

where $\mathbf{1}(c_{e,ki}\&c_{e,kj})$ equals one if edge content of both (v_k, v_i) and (v_k, v_j) are available and equals zero otherwise. By minimizing Eq. (9), we can further fine-tune structure-only embeddings with available edge content.

In order to fine-tune structure-only embeddings with available vertex and edge content embeddings simultaneously, combining Eqs. (7) and (9), we define the following objective function:

$$\mathcal{O}_{Fusion} = \mathcal{O}_{\mathcal{S},\mathcal{V}} + \beta\mathcal{O}_{\mathcal{S},\mathcal{E}}, \tag{10}$$

where β is a hyper-parameter that balances the effects of vertex content and edge content during the fusing phase.

4.4. Content embedding vector extraction

Here, for the sake of completeness, we give a brief introduction to the extraction of content embedding vectors. As described above, the more similar content two vertices have, the more similar the embedding vectors of them should be, and vice versa. Likewise, the more similar content the edges (v_k, v_i) and (v_k, v_j) have, the more similar the learned representations of v_i and v_j should be. Therefore, we need to be capable of capturing similarities between content information.

We employ two SDAEs to extract content embedding vectors from vertex and edge content respectively. Specifically, let $\mathbf{c}_i$ and $\mathbf{c}_{ij}$ denote vertex content $c_{v,i} \in \mathcal{C}_{\mathcal{V}}$ and edge content $c_{e,ij} \in \mathcal{C}_{\mathcal{E}}$ respectively, and substitute $\mathbf{x}_{\mathcal{T}} = \mathbf{c}_i$ and $\mathbf{x}_{\mathcal{T}} = \mathbf{c}_{ij}$ into Eq. (3) successively, the objective function of extracting content embeddings can be defined as

$$\begin{aligned}\mathcal{O}_{\mathcal{C}} &= \mathcal{O}_{\mathcal{V}} + \mathcal{O}_{\mathcal{E}}\\ &= \sum_{(v_i,v_j)\in\mathcal{E}} \|\mathbf{y}_i^{(K_V)} - \mathbf{c}_i\|^2 + \|\mathbf{y}_j^{(K_V)} - \mathbf{c}_j\|^2 + \gamma\|\mathbf{y}_{ij}^{(K_E)} - \mathbf{c}_{ij}\|^2,\end{aligned} \tag{11}$$

where K_V and K_E are the number of layers of two SDAEs for vertex and edge content respectively, and γ is a hyper-parameter that balances the effects of vertex content and edge content during content embedding phase.

By minimizing Eq. (11), we learn the vertex content embedding vectors $\{\mathbf{u}_{\mathcal{V},i}|\ v_i \in V, \mathbf{1}(c_{v,i}) = 1\}$ and the edge content embedding vectors $\{\mathbf{u}_{\mathcal{E},ij}|\ (v_i, v_j) \in \mathcal{E}, \mathbf{1}(c_{e,ij}) = 1\}$.

4.5. Model optimization

In this section, we summarize how to learn an embedding vector for each vertex $v_i \in \mathcal{V}$ by taking account of structure information, vertex content, and edge content. A simple and effective way is to consider the process of learning content embeddings and the fusion of three kinds of embeddings simultaneously, and jointly train the objective functions (10) and (11). After refactoring hyper-parameters, the final objective function is defined as:

$$\begin{aligned}\mathcal{O} &= \mathcal{O}_{\mathcal{S},\mathcal{V},a} + \alpha\mathcal{O}_{\mathcal{S},\mathcal{V},b} + \beta\mathcal{O}_{\mathcal{S},\mathcal{E}} + \gamma\mathcal{O}_{\mathcal{V}} + \eta\mathcal{O}_{\mathcal{E}} + \lambda\mathcal{O}_{reg}\\ &= \sum_{(v_k,v_i,v_j)\in\mathcal{D}} \sum_{v_m\in\{v_i,v_j\}} \mathbf{1}\big(c_{v,k}\&c_{v,m}\big)\|s(\mathbf{u}_k,\mathbf{u}_m) - s\big(\mathbf{u}_{\mathcal{V},k},\mathbf{u}_{\mathcal{V},m}\big)\|^2\\ &\quad+\alpha\mathbf{1}(c_{v,i}\&c_{v,j})\|s\big(\mathbf{u}_i,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{V},i},\mathbf{u}_{\mathcal{V},j}\big)\|^2\\ &\quad+\beta\mathbf{1}\big(c_{e,ki}\&c_{e,kj}\big)\|s\big(\mathbf{u}_i,\mathbf{u}_j\big) - s\big(\mathbf{u}_{\mathcal{E},ki},\mathbf{u}_{\mathcal{E},kj}\big)\|^2\\ &\quad+\gamma\sum_{v_m\in\{v_k,v_i,v_j\}}\mathbf{1}(c_{v,m})\|\mathbf{y}_m^{(K_V)} - \mathbf{c}_m\|^2\\ &\quad+\eta\sum_{e_{mn}\in\{e_{ki},e_{kj}\}}\mathbf{1}(c_{e,mn})\|\mathbf{y}_{mn}^{(K_E)} - \mathbf{c}_{mn}\|^2\\ &\quad+\frac{\lambda}{2}\left(\sum_{k_v}\|\mathbf{W}_V^{(k_v)}\|^2 + \sum_{k_e}\|\mathbf{W}_E^{(k_e)}\|^2\right),\end{aligned} \tag{12}$$

where $\mathbf{u}_i \in \mathbb{R}^d$ is the final embedding vector of vertex v_i, $\mathbf{W}_V^{(k_v)}$ and $\mathbf{W}_E^{(k_e)}$ denote weight matrices of SDAEs for extracting features from vertex content and edge content respectively, and α, β, γ, η and λ are hyper-parameters that balance the effects of different components of our model (i.e., two aspects of fine-tuning with vertex content, fine-tuning with edge content, reconstructions of vertex and edge content, and regularization terms). In this paper, although we just employ structure-only representations to initialize $\mathbf{u}_i$ for each vertex, experimental results show that this simple way still achieves significant performance gains. A more principled way is to incorporate the objectives of structure preserving (e.g., Eqs. (1) and (2) of LINE) into the above objective function, which can be easily extended and we do not elaborate here.

To optimize the aforementioned objective in Eq. (12), we adopt the stochastic gradient descent (SGD) algorithm. For each iteration, we first uniformly sample a batch of triplets from $\mathcal{D}$ and then perform parameter updating according to the gradients of Eq. (12). The parameters involved in our method include embedding vectors of all vertices $\{\mathbf{u}_i\}_{i=1,\ldots,|\mathcal{V}|}$, and the weight matrices (i.e., $\mathbf{W}_{\mathcal{V}}^{(k)}$ and $\mathbf{W}_{\mathcal{E}}^{(k)}$) and bias terms (i.e., $\mathbf{b}_{\mathcal{V}}^{(k)}$ and $\mathbf{b}_{\mathcal{E}}^{(k)}$) of two SDAEs. The gradients of Eq. (12) with respect to them can be easily determined through backpropagation algorithm. For a sampled triplet (v_k, v_i, v_j), we update embedding vectors of these three vertices according to the following function:

$$\mathbf{u}_m = \mathbf{u}_m - \mu\frac{\partial\mathcal{O}}{\partial\mathbf{u}_m}, \qquad m = k, i, j, \tag{13}$$

where $\mu > 0$ is the learning rate.

It is notable that the above naive sampling strategy will be problematic during optimization. Particularly, in a social network, some users have many co-followers (e.g., a thousand), while some users only have few co-followers (e.g., ten). In this case, the number of times vertices with high degrees appear in $\mathcal{D}$ will be dozens of times larger than that of vertices with low degrees. For example, consider two vertices v_i and v_j, of which degrees are 1,000 and 10 respectively. The number of times v_i appears in $\mathcal{D}$ is around $\binom{1000}{2}/\binom{10}{2} \approx 100$ times as large as that of v_j. That is, the embedding vector of v_i will be updated 100 times according to Eq. (13) while the embedding vector of v_j is updated one time during an epoch. Therefore, it is hardly possible to find a good learning rate μ which is suitable to both vertices with high and low degrees. A large learning rate, which is suitable to low-degree vertices, will result in the explosion of updating of high-degree vertex vectors, while a small learning rate, which is suitable to high-degree vertices, cannot optimize embedding vectors of low-degree vertices effectively.

To solve the above optimization problem, we adopt a new sampling strategy, such that a global unique learning rate can adjust to both high-degree and low-degree vertices. To be specific, instead of sampling a batch of triplets from $\mathcal{D}$ directly, we first use the alias method [26] to sample a batch of vertices according to the probability distribution $\Pr(v_k) = \binom{|\mathcal{N}_k|}{2}/\sum_{v_i\in\mathcal{V}}\binom{|\mathcal{N}_i|}{2}$. Then, for each sampled vertex v_k, we uniformly sample two different vertices v_i and v_j to generate the triplet (v_k, v_i, v_j).

Since the time complexities of both sampling a vertex v_k from the alias table and sampling a vertex v_i uniformly from the set $\mathcal{N}_k$ are $O(1)$, the overall time complexity of sampling is $O(m)$ time, where m is the number of sampled triplets in each iteration. For each sampled triplet, the procedure of parameter updating is composed of two parts (i.e., SDAE and fine-tuning), and takes $O(d_c^2h_{\max} + d_uh_c)$ where d_c is the dimension of content information, $h_{\max}$ is the maximum size of hidden layers of SDAE, d_u is the dimension of vertex representations, and h_c is the dimension of latent content embeddings learned by SDAE. Therefore, the overall time complexity of our method is $O(m(d_c^2h_{\max} + d_uh_c))$ for each iteration. Note that the major computation cost comes from matrix multiplications in SDAE, which can be greatly accelerated by GPUs.

Table 1
Statistics of datasets.

Dataset	RSRnet-S	RSRnet-M	RSRnet-L	Flickr
#Vertices	42,069	86,850	132,400	30,386
#Edges	253,155	651,230	1,393,362	443,333
#Labels	5	8	14	9
#Vertices with content	11,139	22,685	37,773	17,959
#Edges with content	56,614	106,469	187,491	N/A
Dim. of vertex content	4,767	8,161	10,926	6,379
Dim. of edge content	7,447	11,367	15,094	N/A

5. Experiments

In order to validate the effectiveness of our model, we compare our method with several state-of-the-art methods on multi-label vertex classification. The empirical results demonstrate that our proposed method can effectively enhance the performance of structure-only network embedding methods with partially available vertex and edge content. Let us first briefly introduce the datasets and experimental settings.

5.1. Datasets

5.1.1. Co-authorship network

AMiner [21,22] is a large-scale publicly available co-author network, which consists of 1,712,433 authors, 2,092,356 papers, and 4,258,615 collaboration relationships. In addition, this dataset contains rich content information, such as author information (e.g., affiliations and research interests) and paper information (e.g., authors, paper title, abstract, and publication venue). Note that only a part of authors and papers have content information.

We construct a researcher network *RSRnet* based on the above dataset. Specifically, we first draw 14 research areas in terms of some popular conferences: *Data Mining* (KDD, ICDM, SDM, PAKDD, and SIGKDD), *Computer Network* (MOBICOM, SIGCOMM, and INFOCOM), *Machine Learning* (ICML, NIPS, ECML, AAAI, and IJCAI), *Database* (SIGMOD, ICDE, VLDB, EDBT, and PODS), *Information Retrieval* (WWW, WSDM, ECIR, CIKM, and SIGIR), *Software Engineering* (FSE/ESEC, OOPSLA, SOSP, and ICSE), *Computer Graphics* (SIGGRAPH, VR, SCA, and SGP), *Information Security* (CCS, S&P, USENIX, NDSS, and RAID), *Computer Architecture* (HPCA, ISCA, and FAST), *Computer Theory* (STOC, FOCS, and LICS), *Multimedia* (ACM MM, ICMR, and ICME), *Computer Vision* (CVPR, ICCV, and ECCV), *Natural Language Processing* (ACL, EMNLP, and COLING), and *Human Computer Interaction* (CHI, IUI, ITS, and MobileHCI). Then, we filter out all researchers who have papers published in the above conference proceedings and regard the corresponding research areas as their labels. Note that each researcher could have multiple labels. Further, two researchers are linked by an edge if they belong to the same affiliation or have collaboration relationships.

On the other hand, in order to construct vertex content, we collect all words that appear in research interests of selected researchers and build a word vocabulary V_v after the words are stemmed and the stop words and meaningless words are removed. For each researcher whose research interests are available, we filter out in-vocabulary words in its research interests and regard these words as vertex content. Likewise, to construct edge content, we collect all words in the titles and abstracts of chosen papers and build a word vocabulary V_e. For each edge of which two endpoint researchers have collaborated papers, we filter out in-vocabulary words in the titles and abstracts of all their co-authored papers and use these words as edge content.

Following [24], to better investigate the effectiveness of different methods on datasets with different scales, we construct three datasets: *RSRnet-S* (Small) with only researchers from 5 research fields including *Data Mining*, *Computer Network*, *Machine Learning*, *Database*, and *Information Retrieval*, *RSRnet-M* (Middle) with researchers from 3 additional fields (i.e., *Software Engineering*, *Computer Graphics*, and *Information Security*) compared to *RSRnet-S*, and *RSRnet-L* (Large) with researchers from all 14 research fields. The statistics of the three datasets in our experiments are summarized in Table 1.

5.1.2. Social network

Flickr is an online content sharing platform, where users can share their content, upload their tags, and join different interest groups. We use a publicly available dataset [29] crawled from this site, where each vertex represents a user in Flickr and each edge indicate the friendship and/or the commentship (i.e., who comments on whose photos) between two users. This dataset includes user-generated content (i.e., tags aggregated on their photos) which is regarded as vertex content, and includes users' joined groups for classification. We randomly select a portion of vertices and drop their content to simulate the missing of content information. Since there is no edge content in this dataset, we only use available vertex content to perform fine-tuning. The statistics of this dataset are also summarized in Table 1.

5.2. Experimental settings

We use several state-of-the-art network embedding methods as baselines:
Structure:

Table 2
The structures of SDAE.

(a) SDAE for vertex content.

Dataset	Neural network structure
RSRnet-S	4767-512-128-512-4767
RSRnet-M	8161-512-128-512-8161
RSRnet-L	10926-1024-128-1024-10926
Flickr	6379-512-128-512-6379

(b) SDAE for edge content.

Dataset	Neural network structure
RSRnet-S	7447-512-128-512-7447
RSRnet-M	11367-1024-128-1024-11367
RSRnet-L	15094-2048-1024-128-1024-2048-15094
Flickr	N/A

- *LINE* [20] learns vertex embedding vectors from 1- and 2-step neighborhood information separately, and then concatenates two kinds of vertex embedding vectors in a direct manner.
- *GraRep* [3] further generalizes LINE to k-step neighborhood information and formulates k-step proximity preservation as matrix factorization. The concatenation of different steps of representations is regarded as the final vertex embedding vectors.
- *Node2vec* [7] proposes two kinds of strategies of random walks based on DeepWalk [16], such that high-order proximities between vertices could be explored more effectively and efficiently.

Structure and content:

- *Naive Combination (NC)* concatenates the best-performed structure-only embedding vectors and the low-dimensional vectors of content information learned by SDAE.
- *TADW* [30] incorporates text content into the processing of network embedding via matrix factorization.
- *TriDNR* [15] exploits structure, textual vertex content, and vertex labels for network embedding. It has two kinds of learning modes: unsupervised and semi-supervised manners. For a fair comparison, the unsupervised manner is used in our experiments.
- *SINE* [31] regards content information as a kind of contextual vertices and applies the skip-gram model [13] to learn vertex representations. In other words, SINE uses the embedding vector of each vertex to predict the corresponding content information.

For content-enhanced baselines, we use the suffix “-V” to refer that we only consider vertex content (i.e., research interests) as content information. The suffix “-VE” denotes that both the original vertex content (i.e., research interests) and the pseudo vertex content (derived from the aggregation of all edge content associated with a vertex, i.e., titles and abstracts of co-authored papers) are considered as content information.

For LINE, we set the initial learning rate to 0.025, and the number of negative samples to 5 as suggested in [20]. For GraRep, as mentioned in [3], we set the maximum step $K = 5$. For Node2vec, the best in-out and return hyper-parameters are selected with a grid search over $p, q \in \{0.25, 0.50, 1, 2, 4\}$ on validation sets as the original paper. For TADW, as selected by [30], we set $T \in \mathbb{R}^{200\times|V|}$, $k = 80$, and $\lambda = 0.2$. For TriDNR, we set *window size* to 8, and the text weight α to 0.8 as the original papers. The rest of parameters are set as defaults.

For our model PVECB, we use the suffix “-L” (resp. “-G”) to denote that the embedding vectors learned by LINE (resp. GraRep) are employed as the structure-only representations in our experiments. The structures of two SDAEs for extracting vertex and edge content embedding vectors are set empirically as listed in Table 2. To stabilize the learning process, we first pre-train SDAEs using available content information and RMSprop optimizer with the learning rate set to 0.01. When it comes to fine-tuning (i.e., optimizing the objective function (12)), we set the learning rate to $\mu = 0.1$. Following the original paper of node2vec [7], we randomly sample 10% labeled vertices as the validation set and tune other hyperparameters with Bayesian optimization over $\alpha, \beta, \gamma, \eta \sim U(0.1, 10)$ and $\lambda \sim U(0.001, 0.1)$ on the validation set. The maximum number of trials of Bayesian optimization is set to 100. Table 3 lists the best set of hyperparameters on each dataset.

5.3. Multi-label vertex classification

In this section, we evaluate the effectiveness of different vertex embedding vectors through the task of multi-label vertex classification on three co-authorship networks and one social network. We take vertex embeddings as features to train a one-vs-rest logistic regression classifier with L2 regularization. We randomly sample a portion of labeled vertices to train the classifier during the training phase and predict the labels of the rest of vertices during the test phase. With the percentage of training vertices p% varying from 10% to 90%, Tables 4, 5,6, and 7 report the Micro-F_1 scores and Fig. 3 shows the Macro-F_1 scores on four networks. All of the results are averaged over 10 different trials.

Table 3
The best sets of hyperparameters on four datasets.

(a) PVECB-L.

Dataset	α	β	γ	η	λ
RSRnet-S	0.1598	8.6729	6.7242	3.5265	0.02041
RSRnet-M	0.1327	8.9495	3.4778	0.4902	0.01296
RSRnet-L	0.2817	9.0051	4.6419	7.1676	0.03676
Flickr	0.1639	N/A	2.4719	N/A	0.01020

PVECB-G.

Dataset	α	β	γ	η	λ
RSRnet-S	0.1743	9.5474	1.6053	4.8667	0.01927
RSRnet-M	0.2561	8.3144	8.7058	9.8542	0.03047
RSRnet-L	0.2618	9.7414	6.0233	1.9424	0.05583
Flickr	2.0266	N/A	2.1713	N/A	0.01689

Table 4
Results of multi-label vertex classification on *RSRnet-S*. Gain-L and Gain-G refer to the gains of PVECB-L and PVECB-G over LINE and GraRep respectively.

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1 (%)	LINE	42.29	43.72	44.24	44.52	44.66	44.83	44.91	44.89	45.00
	GraRep	47.90	49.16	49.59	49.84	50.03	50.23	50.33	50.45	50.76
	Node2vec	44.25	44.68	44.67	44.75	44.86	44.86	44.72	44.75	44.82
	NC-V	48.66	49.90	50.36	50.57	50.76	50.93	51.03	51.18	51.36
	TADW-V	30.09	30.43	30.52	30.68	30.92	31.27	31.55	32.09	32.51
	TriDNR-V	44.43	44.69	44.59	44.65	44.65	44.69	44.69	44.70	44.81
	SINE-V	49.16	49.62	49.73	49.78	49.80	49.83	49.83	49.77	49.86
	NC-VE	49.36	50.51	50.93	51.18	51.38	51.51	51.65	51.77	51.90
	TADW-VE	41.34	41.79	42.05	42.22	42.51	42.74	42.97	43.38	43.65
	TriDNR-VE	44.81	45.09	45.03	44.96	45.01	44.99	44.88	44.83	45.09
	SINE-VE	50.50	51.04	51.14	51.13	51.13	51.25	51.27	51.23	51.07
	PVECB-L	48.17	49.20	49.53	49.71	49.77	49.84	49.85	49.89	49.97
	PVECB-G	**51.48**	**52.34**	**52.71**	**52.76**	**52.83**	**52.88**	**52.90**	**52.90**	**52.95**
	Gain-L (%)	13.91	12.55	11.96	11.65	11.45	11.17	10.99	11.14	11.05
	Gain-G (%)	7.48	6.47	6.29	5.84	5.59	5.27	5.12	4.84	4.31

Table 5
Results of multi-label vertex classification on *RSRnet-M*. Gain-L and Gain-G refer to the gains of PVECB-L and PVECB-G over LINE and GraRep respectively.

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1 (%)	LINE	26.45	27.48	27.98	28.05	28.22	28.30	28.32	28.42	28.44
	GraRep	35.71	36.65	37.10	37.40	37.62	37.79	37.90	38.13	38.35
	Node2vec	30.88	30.85	30.90	30.92	30.96	30.98	30.97	31.04	31.16
	NC-V	36.26	37.23	37.67	37.92	38.10	38.28	38.42	38.60	38.85
	TADW-V	23.19	23.27	23.33	23.45	23.50	23.55	23.61	23.72	23.73
	TriDNR-V	24.54	24.19	24.03	24.03	24.03	24.06	24.02	24.00	24.05
	SINE-V	31.66	31.80	31.80	31.88	31.86	31.84	31.87	31.88	31.93
	NC-VE	36.74	37.70	38.13	38.39	38.57	38.74	38.87	39.07	39.32
	TADW-VE	34.57	34.66	34.70	34.77	34.83	34.88	34.88	34.95	35.21
	TriDNR-VE	27.37	27.11	27.00	27.00	26.98	26.90	26.90	26.74	26.88
	SINE-VE	35.54	35.75	35.84	35.93	35.96	35.99	35.99	35.95	36.13
	PVECB-L	35.43	36.10	36.35	36.49	36.60	36.66	36.62	36.72	37.03
	PVECB-G	**40.05**	**40.80**	**41.09**	**41.31**	**41.43**	**41.52**	**41.60**	**41.78**	**42.00**
	Gain-L (%)	33.94	31.35	29.93	30.11	29.66	29.53	29.28	29.19	30.21
	Gain-G (%)	12.15	11.32	10.74	10.44	10.14	9.87	9.74	9.59	9.50

From the results on three co-authorship networks, we have the following observations:

1. Our method PVECB consistently and significantly outperforms all baselines on all three datasets, which demonstrates that PVECB can effectively utilize available content information to improve structure-only representations. Specifically, our method achieves an average gain of around 12% (resp. 30%, and 29%) over LINE in term of Micro-F_1 on *RSRnet-S* (resp. *RSRnet-M*, and *RSRnet-L*). On the other hand, PVECB can increase the performance of GraRep by around 6%, 10%, and 15% on average on *RSRnet-S*, *RSRnet-M*, and *RSRnet-L* respectively.
2. The naive combination concatenates the best-performed structure-only embeddings (i.e., GraRep) and the latent content embeddings as the final vertex embedding vectors. Both of NC-VE and PVECB-G use vertex and edge content to

Table 6
Results of multi-label vertex classification on *RSRnet-L*. Gain-L and Gain-G refer to the gains of PVECB-L and PVECB-G over LINE and GraRep respectively.

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1 (%)	LINE	23.68	24.87	25.37	25.63	25.83	25.88	25.96	26.05	26.10
	GraRep	30.81	31.83	32.30	32.58	32.81	32.97	33.07	33.11	33.26
	Node2vec	26.45	26.82	26.93	27.00	27.04	27.01	27.02	27.00	27.02
	NC-V	31.43	32.38	32.84	33.09	33.28	33.43	33.54	33.59	33.65
	TADW-V	21.69	21.71	21.81	21.86	21.97	22.00	22.06	22.08	22.08
	TriDNR-V	18.39	18.20	18.15	18.09	18.17	18.18	18.24	18.23	18.17
	SINE-V	24.30	24.54	24.67	24.70	24.80	24.79	24.82	24.77	24.73
	NC-VE	32.19	33.09	33.56	33.81	34.01	34.14	34.22	34.26	34.36
	TADW-VE	34.83	34.86	34.93	35.00	35.11	35.16	35.18	35.19	35.37
	TriDNR-VE	22.42	22.26	22.33	22.37	22.40	22.32	22.34	22.27	22.31
	SINE-VE	28.75	29.37	29.64	29.84	29.93	30.05	30.11	30.15	30.27
	PVECB-L	31.64	32.50	32.88	33.07	33.22	33.28	33.30	33.27	33.15
	PVECB-G	**36.30**	**37.00**	**37.35**	**37.56**	**37.76**	**37.87**	**37.93**	**37.96**	**37.90**
	Gain-L (%)	33.62	30.67	29.60	29.02	28.62	28.59	28.28	27.74	27.04
	Gain-G (%)	17.81	16.24	15.62	15.30	15.08	14.85	14.69	14.66	13.93

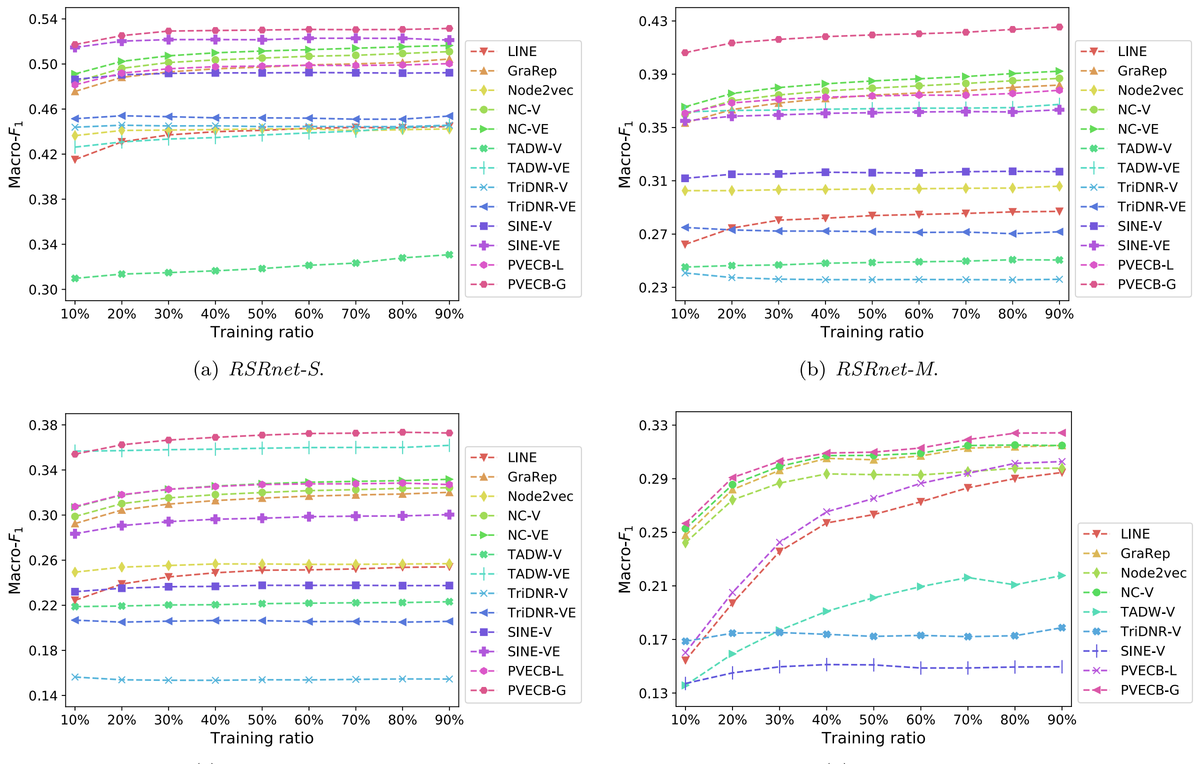
Table 7
Results of multi-label vertex classification on *Flickr*. Gain-L and Gain-G refer to the gains of PVECB-L and PVECB-G over LINE and GraRep respectively.

Metric	Algorithm	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1 (%)	LINE	20.59	24.27	26.33	27.37	27.59	27.98	28.74	28.96	29.54
	GraRep	27.00	29.35	30.60	31.21	31.15	31.24	31.72	31.67	31.98
	Node2vec	24.37	26.37	27.26	27.82	27.76	27.50	27.88	28.05	27.93
	NC-V	27.40	29.52	30.73	31.31	31.35	31.33	31.89	31.77	32.02
	TADW-V	18.64	20.92	22.54	23.47	23.88	24.15	24.74	24.10	24.75
	TriDNR-V	17.58	18.19	18.21	18.10	17.90	17.89	17.92	17.92	18.22
	SINE-V	17.80	18.45	18.91	18.98	19.07	18.65	18.68	18.76	19.10
	PVECB-L	21.05	24.62	26.70	27.95	28.33	28.89	29.44	29.75	30.36
	PVECB-G	**27.50**	**29.80**	**30.94**	**31.77**	**31.82**	**31.98**	**32.22**	**32.45**	**32.80**
	Gain-L (%)	2.22	1.45	1.40	2.10	2.66	3.26	2.42	2.75	2.78
	Gain-G (%)	1.86	1.54	1.12	1.79	2.14	2.38	1.60	2.46	2.57

improve structure-only representations learned by GraRep. As we can see, the performance of NC-VE is only slightly better than the original GraRep, and specifically NC-VE only achieves on average about 2.6% performance gains for *RSRnet-S* and *RSRnet-M*, and 3.7% for *RSRnet-L*. The main reason why the gain of NC-VE is quite smaller than that of PVECB-G is that the former just aggregates all content information associated with a vertex and does not explicitly model how vertex content and edge content should be employed during the process of learning vertex representations.

3. TADW and TriDNR do not perform well on all three datasets, and the performance of them is worse than that of NC, especially on *RSRnet-S* and *RSRnet-M*. The reasons are twofold: (a) The imprecision caused by the straightforward aggregation of all content information as explained above; (b) Both of two methods assume that content information of each vertex is available. To make these two methods suitable to the studied scenario where a part of vertices have no available content, it will bring in a lot of noise to feed zero vectors for vertices without content, which we will analyze in detail in Section 5.4. On the other hand, although SINE shows competitive performance on *RSRnet-S*, our method significantly outperforms SINE on other datasets. The reasons could be also two-fold: (a) SINE fails to capture high-level semantic similarities between content information as we mentioned in Section 2; (b) SINE directly uses available content information to manipulate the embedding vectors of the corresponding vertices, which has a bad impact on the structure of the latent space and the embedding vectors of vertices that do not have content, just like TADW.

In addition, our method PVECB also achieves better performance than baselines on the social network *Flickr*. We observe that the gains of PVECB over LINE and GraRep on Flickr are smaller than the gains on the three co-authorship networks. The reasons could be twofold: (a) The text on social networks is noisy [1] since people usually use informal expressions when uploading content. However, for co-authorship networks, content information (e.g., titles and abstracts of papers) is well organized and formal. Therefore, it is easier to improve vertex embeddings by comparing similarities between content information; (b) The social network *Flickr* only contains vertex content, while the three co-authorship networks contain both vertex and edge content. In general, edge content exhibits the characteristics of pairwise interactions rather than individual vertices, and thus can be used to more accurately calibrate embeddings of a pair of vertices (e.g., for (v_k, v_i, v_j), using $c_{e,ki}$ and $c_{e,kj}$ to adjust $\mathbf{u}_i$ and $\mathbf{u}_j$).



(a) *RSRnet-S*. (b) *RSRnet-M*.

(c) *RSRnet-L*. (d) *Flickr*

Fig. 3. The Macro-F_1 scores of multi-label vertex classification on three datasets.

5.4. Analysis: Performance on vertices with and without content

To further investigate the consistency of PVECB in incorporating content information into the process of embedding, we define a *vertex with content* as a vertex of which either vertex content or associated edge content is available, and evaluate the performance of PVECB and content-enhanced baselines with suffix "-VE" on vertices with and without content respectively. In particular, we randomly select a portion of vertices to train a one-vs-rest logistic regression classifier. Then, the rest of vertices are split into two parts: vertices with and without content. Fig. 4 reports the Micro-F_1 scores with respect to two parts of testing vertices on the three co-authorship networks. From the results, we have the following observations:

1. PVECB vs. TADW-VE: The performance of TADW-VE is extremely biased towards vertices with content. As shown on the left side of Fig. 4, TADW-VE achieves very high Micro-F_1 scores on vertices with content for all three datasets. Specifically, the average Micro-F_1 scores over different training ratios of TADW-VE are 75.9%, 67.8% and 63.3% for predicting labels of vertices with content on *RSRnet-S*, *RSRnet-M*, and *RSRnet-L* respectively. However, TADW-VE fails to predict labels of vertices without content, and the average Micro-F_1 scores are only 1.60%, 0.09%, and 0.01% on *RSRnet-S*, *RSRnet-M*, and *RSRnet-L* respectively as shown on the right side of the figure. On the contrary, our proposed method achieves the best trade-off between the performance on vertices with and without content. Specifically, PVECB-G and PVECB-L obtain the best performance, except for TADW-VE, for predicting labels of vertices with content. On the other hand, PVECB-G and PVECB-L significantly outperform TADW-VE and TriDNR-VE when it comes to vertices without content.
2. PVECB-G vs. NC-VE: Both PVECB-G and NC-VE exploit vertex and edge content on the basis of vertex representations learned by GraRep. Compared with NC-VE, PVECB-G obtains almost the same performance in predicting labels of vertices without content. On the other hand, the performance of PVECB-G in predicting labels of vertices with content is much better than that of NC-VE. Specifically, the differences between the average Micro-F_1 scores of PVECB-G and NC-VE are 3.70%(↑) and −0.45%(↓) for predicting labels of vertices with and without content respectively on *RSRnet-S*. The corresponding values are 7.08%(↑) and −0.83%(↓) on *RSRnet-M*, and 8.45%(↑) and −1.27%(↓) on *RSRnet-L*. The empirical results demonstrate that PVECB-G can effectively make use of available vertex and edge content to improve embedding vectors of vertices with content, with little loss of qualities of representations of vertices without content.

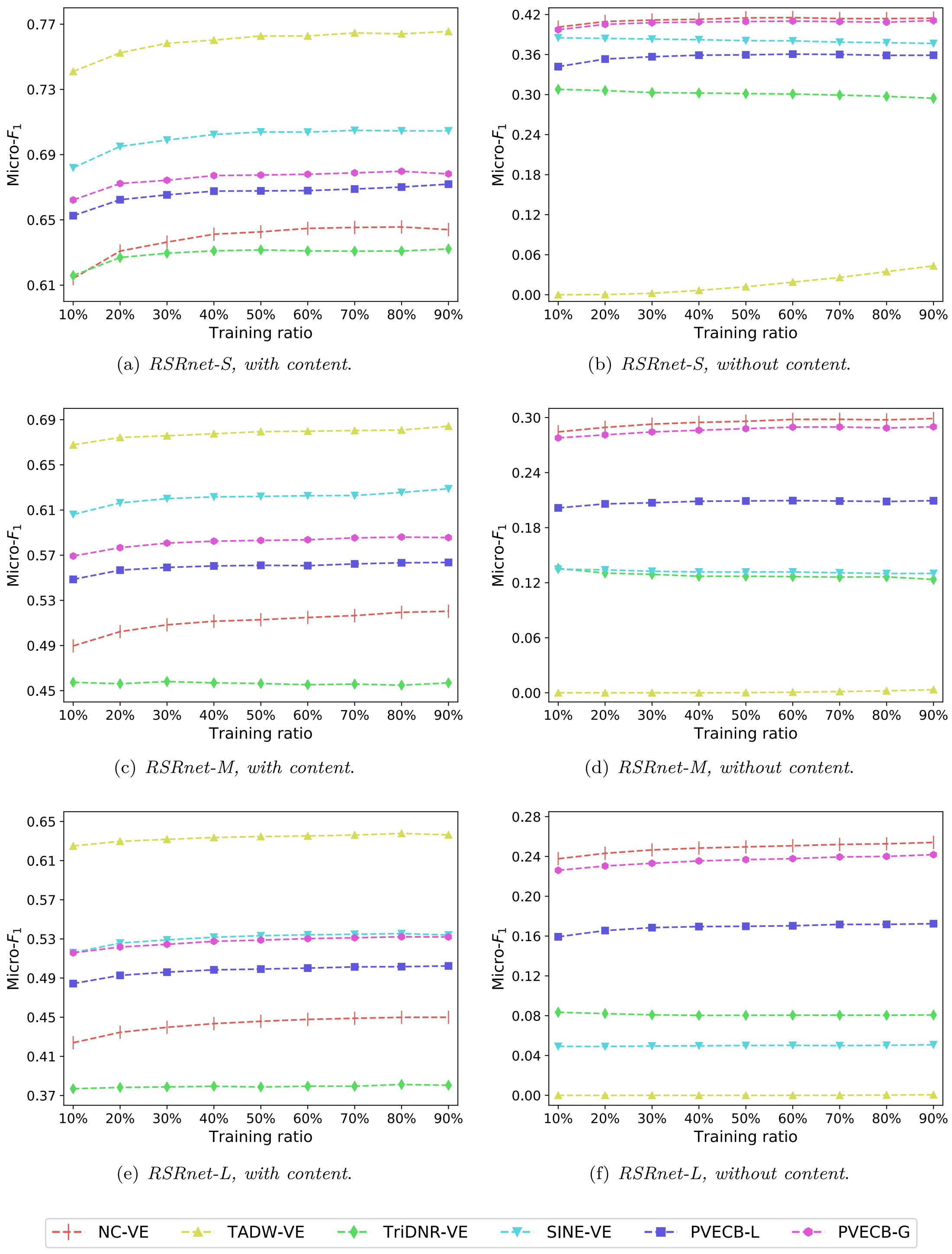
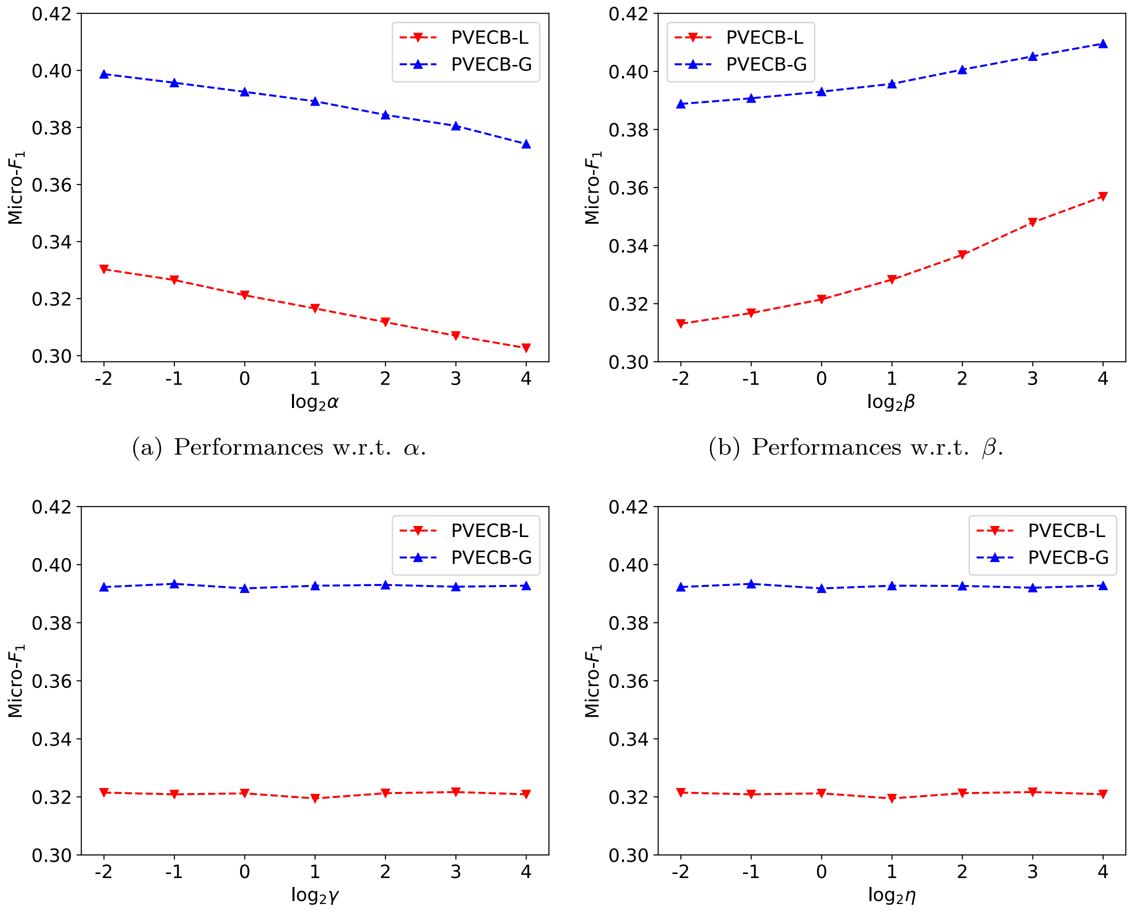


Fig. 4. The Micro-F_1 scores of multi-label vertex classification on vertices with and without content.

On the other hand, NC-VE achieves good performance gains on vertices with content, while maintaining the performance on vertices without content. However, although existing content-enhanced network embedding methods have better performance than NC-VE on vertices with content, their performance on vertices without content dramatically degrades, especially for TADW (as we discussed in Section 3.2). These results account for why NC-VE is almost always among the best performing baseline methods in Section 5.3.



(a) Performances w.r.t. α.

(b) Performances w.r.t. β.

(c) Performances w.r.t. γ.

(d) Performances w.r.t. η.

Fig. 5. Parameter sensitivity.

5.5. Parameter sensitivity

In what follows, we discuss the parameter sensitivity with respect to four hyper-parameters: the weight of the aspect (b) of fine-tuning with vertex content α, the weight of fine-tuning with edge content β, the weight of reconstruction of vertex content γ, and the weight of reconstruction of edge content η. We fix the training ratio to 50% and investigate how different choices of parameters affect the performance of our proposed method on the dataset *RSRnet-M*. Except for the examined parameter, all other hyper-parameters are set to 1. Fig. 5 shows the Micro-F_1 scores over different choices of parameters.

From Figs. 5(a) and 5(b), we observe that the hyper-parameters α and β have opposite influences on the performance of our method. Specifically, the performance of PVECB improves as α decreases and β increases. It is reasonable because edge content represents specific pair-wise interactions and can be used to guide the learning of embedding vectors more effectively. On the other hand, the Micro-F_1 scores of our method fluctuate in a very small range (less than 1%) with γ and η varying from 0.25 to 16 as shown in Figs. 5(c) and 5(d). This demonstrates that PVECB can stay relatively stable when the weights of reconstruction terms vary within a reasonable range, which benefits from the powerful capability of SDAE as we pre-train two SDAEs for extracting content embedding vectors before fine-tuning.

6. Conclusion

In this paper, we propose a novel method PVECB to boost the existing structure-only network embedding method with available vertex and edge content. PVECB employs two kinds of mechanisms to fine-tune structure-only embedding vectors with vertex content and edge content respectively. Its attractive property is to allow us to implicitly diffuse content information over the network and learn effective vertex embedding vectors even when only a small portion of vertices and edges have content information. Experiments on four real-world datasets demonstrate the capability of PVECB to boost structure-only network embedding methods. Recent research has shown that deep convolutional neural networks are powerful for learning features from multimedia data (e.g., images and speeches), which is prevalent in networks such as online social networks. In the future, we plan to extend PVECB with deep learning techniques for learning networks with multimedia data on vertices and edges.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The research presented in this paper is supported in part by Shenzhen Basic Research Grant (JCYJ20170816100819428), National Key R&D Program of China (2018YFC0830500), National Natural Science Foundation of China (61922067, U1736205, 61603290), Natural Science Basic Research Plan in Shaanxi Province of China (2019JM-159), and Natural Science Basic Research Plan in Zhejiang Province of China (LGG18F020016).

References

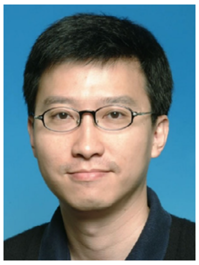
[1] T. Baldwin, P. Cook, M. Lui, A. MacKinlay, L. Wang, How noisy social media text, how different social media sources? in: Proceedings of the Sixth International Joint Conference on Natural Language Processing, 2013, pp. 356–364.
[2] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in: Advances in Neural Information Processing Systems, 2002, pp. 585–591.
[3] S. Cao, W. Lu, Q. Xu, Grarep: Learning graph representations with global structural information, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 891–900.
[4] H. Chen, X. Sun, Y. Tian, B. Perozzi, M. Chen, S. Skiena, Enhanced network embeddings via exploiting edge labels, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, 2018, pp. 1579–1582.
[5] T.F. Cox, M.A. Cox, Multidimensional Scaling, Chapman and hall/CRC, 2000.
[6] Y. Dong, J. Tang, S. Wu, J. Tian, N.V. Chawla, J. Rao, H. Cao, Link prediction and recommendation across heterogeneous social networks, in: IEEE 12th International Conference on Data Mining, IEEE, 2012, pp. 181–190.
[7] A. Grover, J. Leskovec, Node2vec: scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 855–864.
[8] D.P. Kingma, M. Welling, Stochastic gradient vb and the variational auto-encoder, in: Second International Conference on Learning Representations, ICLR, 2014.
[9] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: Proceedings of the 31st International Conference on Machine Learning, 2014, pp. 1188–1196.
[10] O. Levy, Y. Goldberg, Neural word embedding as implicit matrix factorization, in: Advances in Neural Information Processing Systems, 2014, pp. 2177–2185.
[11] H. Li, H. Wang, Z. Yang, M. Odagaki, Variation autoencoder based network representation learning for classification, in: Proceedings of ACL 2017, Student Research Workshop, 2017, pp. 56–61.
[12] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, J. Am. Soc. Inf.Sci. Technol. 58 (7) (2007) 1019–1031.
[13] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: Advances in Neural Information Processing Systems, 2013, pp. 3111–3119.
[14] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 1105–1114.
[15] S. Pan, J. Wu, X. Zhu, C. Zhang, Y. Wang, Tri-party deep network representation, Network 11 (9) (2016) 12.
[16] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 701–710.
[17] L.F. Ribeiro, P.H. Saverese, D.R. Figueiredo, Struc2vec: learning node representations from structural identity, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 385–394.
[18] S.T. Roweis, L.K. Saul, Nonlinear dimensionality reduction by locally linear embedding, Science 290 (5500) (2000) 2323–2326.
[19] R. Salakhutdinov, G. Hinton, Semantic hashing, Int. J. Approx. Reason. 50 (7) (2009) 969–978.
[20] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: large-scale information network embedding, in: Proceedings of the 24th International Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.
[21] J. Tang, L. Yao, D. Zhang, J. Zhang, A combination approach to web user profiling, ACM Trans. Knowl. Discov. Data (TKDD) 5 (1) (2010) 2.
[22] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, Z. Su, Arnetminer: extraction and mining of academic social networks, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2008, pp. 990–998.
[23] J.B. Tenenbaum, V. De Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323.
[24] C. Tu, Z. Zhang, Z. Liu, M. Sun, Transnet: translation-based network representation learning for social relation extraction, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, 2017, pp. 19–25.
[25] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (Dec) (2010) 3371–3408.
[26] M.D. Vose, A linear algorithm for generating random numbers with a given distribution, IEEE Trans. Softw. Eng. 17 (9) (1991) 972–975.
[27] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 1225–1234.
[28] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, S. Yang, Community preserving network embedding, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.
[29] X. Wang, L. Tang, H. Liu, L. Wang, Learning with multi-resolution overlapping communities, Knowl. Inf. Syst. (KAIS) (2012), doi:10.1007/s10115-012-0555-0.
[30] C. Yang, Z. Liu, D. Zhao, M. Sun, E. Chang, Network representation learning with rich text information, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.
[31] D. Zhang, J. Yin, X. Zhu, C. Zhang, Sine: scalable incomplete network embedding, in: 2018 IEEE International Conference on Data Mining, IEEE, 2018, pp. 737–746.
[32] J. Zhang, P.S. Yu, Z.-H. Zhou, Meta-path based multi-network collective link prediction, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 1286–1295.

Lin Lan received the B.S. in automation engineering from Xi'an Jiaotong University, Xi'an, P.R. China, in 2016. He is currently a Ph.D. student with NSKEYLAB at Xi'an Jiaotong University. His research interests include network analysis, network representation learning, and deep learning and its applications.

Pinghui Wang received the Ph.D. degree in control science and engineering from Xi'an Jiaotong University, Xi'an, P.R. China. From April 2012 to October 2012, he was a postdoctoral researcher with the Department of Computer Science and Engineering at The Chinese University of Hong Kong. From October 2012 to July 2013, he was a postdoctoral researcher with the School of Computer Science at McGill University, QC, Canada. He is currently an associate professor with the department of automation at Xi'an Jiaotong University. His research interests include Internet traffic measurement and modeling, traffic classification, abnormal detection, and online social network measurement.

Junzhou Zhao received the Ph.D. degree in control science and engineering from Xi'an Jiaotong University, Xi'an, P.R. China. He is currently a postdoctoral researcher at King Abdullah University of Science and Technology, Thuwal, Saudi Arabia. His research focuses on mining and measuring massive large scale networks/graphs, with a particular interest in online social networks, a.k.a. the network science.

Jing Tao received the B.S and M.S degrees in automatic control from Xi'an Jiaotong University, Xi'an, China, in 2001 and 2006 respectively. He is currently a teacher in Xi'an Jiaotong University and on-the-job Ph.D. candidate with the Systems Engineering Institute and SKLMS Laboratory, Xi'an Jiaotong University under the supervision of Prof. Xiaohong Guan. His research interests include Internet traffic measurement and modeling, traffic classification, abnormal detection, and botnet.



John C.S. Lui received the Ph.D. degree in computer science from UCLA. He is currently a professor in the Department of Computer Science and Engineering at The Chinese University of Hong Kong. His current research interests include communication networks, network/system security (e.g., cloud security, mobile security, etc.), network economics, network sciences (e.g., online social networks, information spreading, etc.), cloud computing, large-scale distributed systems and performance evaluation theory. He serves in the editorial board of IEEE/ACM Transactions on Networking, IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, Journal of Performance Evaluation and International Journal of Network Security. He was the chairman of the CSE Department from 2005 to 2011. He received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award. He is also a corecipient of the IFIP WG 7.3 Performance 2005 and IEEE/IFIP NOMS 2006 Best Student Paper Awards. He is an elected member of the IFIP WG 7.3, fellow of the ACM, fellow of the IEEE, and croucher senior research fellow.

Xiaohong Guan received the B.S. and M.S. in automatic control from Tsinghua University, Beijing, China, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from the University of Connecticut, Storrs, US, in 1993. From 1993 to 1995, he was a consulting engineer at PG&E. From 1985 to 1988, he was with the Systems Engineering Institute, Xi'an Jiaotong University, Xi'an, China. From January 1999 to February 2000, he was with the Division of Engineering and Applied Science, Harvard University, Cambridge, MA. Since 1995, he has been with the Systems Engineering Institute, Xi'an Jiaotong University, and was appointed Cheung Kong Professor of Systems Engineering in 1999, and dean of the School of Electronic and Information Engineering in 2008. Since 2001 he has been the director of the Center for Intelligent and Networked Systems, Tsinghua University, and served as head of the Department of Automation, 2003–2008. He is an Editor of IEEE Transactions on Power Systems and an Associate Editor of Automata. His research interests include allocation and scheduling of complex networked resources, network security, and sensor networks. He has been elected Fellow of IEEE.