Data Science and Engineering (2018) 3:150–165
https://doi.org/10.1007/s41019-018-0065-5

Representing Graphs as Bag of Vertices and Partitions for Graph Classification

Mansurul Bhuiyan[1] · Mohammad Al Hasan[2]

Received: 15 October 2017 / Revised: 25 April 2018 / Accepted: 25 May 2018 / Published online: 28 June 2018

Abstract
Graph classification is a difficult task because finding a good feature representation for graphs is challenging. Existing methods use topological metrics or local subgraphs as features, but the time complexity for finding discriminatory subgraphs or computing some of the crucial topological metrics (such as diameter and shortest path) is high, so existing methods do not scale well when the graphs to be classified are large. Another issue of graph classification is that the number of distinct graphs for each class that are available for training a classification model is generally limited. Such scarcity of graph data resources yields models that have much fewer instances than the model parameters, which leads to poor classification performance. In this work, we propose a novel approach for solving graph classification by using two alternative graph representations: the bag of vertices and the bag of partitions. For the first representation, we use representation learning-based node features and for the second, we use traditional metric-based features. Our experiments with 43 real-life graphs from seven different domains show that the bag representation of a graph improves the performance of graph classification significantly. We have shown 4–75% improvement on the vertex-based and 4–36% improvement on partition-based approach over the existing best methods. Besides, our vertex and partition multi-instance methods are on average 75 and 11 times faster in feature construction time than the current best, respectively.

Keywords Graph classification · Graph embedding · Frequent subgraph patterns

1 Introduction

Graph classification is an important research task which is used for solving various real-life classification problems. For example, in the domain of software engineering, softwares are represented as *program flow graphs* and graph classification is used for discriminating between correct and faulty software [6]. In the domain of mobile security, classification of *function call graphs* is used to categorize between malicious (malware) and benign Android application [12]. The most well-known example of graph classification probably comes from the cheminformatics domain, where graphs are used for representing chemical compounds, and molecular property descriptors are developed to classify these graphs for performing structure–activity relationship (SAR) analysis [51]. Formally, the graph classification problem is to develop a mapping function $f(\mathbf{x}) : X \rightarrow \{1, \ldots, c\}$, given a set of training instances $X = \langle \mathbf{x}_i, y_i \rangle_{i=1}^{N}$, where an instance $\mathbf{x}_i \in X$ is the representation of a graph in a chosen feature space and $y_i \in \{1, \ldots, c\}$ is the class label associated with $\mathbf{x}_i$.

A graph does not have a natural embedding in a metric space, but the majority of supervised classification methods require that the graph is represented as a point in a metric space. So, a critical prerequisite of a graph classification task is selecting features for embedding a graph in a metric space. Existing works on graph classification use two kinds of feature representation for a graph: topological features [1, 26] and local substructure-based features [11, 50]. Methods belonging to the former group compute different local or global topology-based graph metrics such as centralities, eccentricity, egonet degree, egonet size and

✉ Mohammad Al Hasan
alhasan@iupui.edu

Mansurul Bhuiyan
Mansurul.Bhuiyan@ibm.com

[1] IBM Almaden Research Center, San Jose, CA, USA

[2] Indiana University Purdue University Indianapolis, 723 W. Michigan Street, Indianapolis, IN 46202, USA

diameter and use these as features of the graphs. Methods belonging to the latter group extract local topologies such as frequent subgraphs [20, 25], discriminative subgraphs [37, 52] or graphlets [36] from the input graphs and use binary features representing their occurrence or lack of occurrences in a given graph. Besides these, some graph kernels have also been developed [4, 41] that encode the similarity between a pair of graphs in a Gram matrix. A specialized graph classification platform, gBoost [39], is proposed, which unifies the subgraph extraction and graph classification in an integrative process.

Much development has been made in graph classification, yet feature selection for graph data still remains as a challenge, specifically for the case when the graph to be classified is large in size. For such graphs, the task of feature value computation does not scale with the size of the graphs. For instance, for the topological feature-based methods, the computation of a number of metrics, such as the shortest path and diameter, has at least a quadratic complexity, which is not quite feasible for many real-life graphs. We experiment with a few coauthorship networks (average number of vertices in these graphs is around 96 thousands) and find that for these graphs, the execution time for constructing metrics that are used in two most-recent graph classification methods ([1, 26]) is around 40 hours. (Experimental results are available in Sect. 4.5.) Likewise, extracting frequent subgraph features is also not a scalable task. For example, we ran Gaston [32] (current state-of-the-art frequent subgraph mining algorithm) on several animal and human contact graphs (list of datasets is given in Table 1) with 30% support, but the mining task did not finish in 2 whole days of running. As we can see in Table 1, most of the graphs that we use in our experiments are of moderate size, yet existing methods for graph feature extraction are already infeasible for many of these graphs. Feature extraction tasks for even larger graphs, such as Facebook or Wikipedia, will be nearly impossible! Besides the high computational cost of feature extraction, selecting a small number of good features is another challenge for the graph classification task. More often, the domain knowledge from the analyst is critical for finding good features for graph classification, as different kinds of features work well for graphs appearing in various domains.

In recent years, unsupervised feature learning using neural networks has become popular. These methods help an analyst discover features automatically, thus obviating the necessity of feature engineering using domain knowledge. Researchers achieve excellent performance with these learning techniques for extraction of features form text [2, 29], speech [43] and images [45]. Perozzi et al. [27, 35] have shown the potential of neural network models for learning feature representation of the vertices of an input graph for solving vertex classification [35] and link prediction [27] tasks. However, none of the existing methods find feature representation for a graph in a graph database for graph classification. In addition to these neural network-based techniques, there exist a few works [38, 40] that find optimal embedding of a graph in the Euclidean space while preserving the shortest path distance. Nevertheless, due to their high computation cost, these embedding-based methods work well for small graphs only. Besides, the metric representation using these methods performs poorly in graph classification setting, as we will show in experiment section.

Existing graph classification methods also suffer from an issue that arises from the application of graph mining in real-life settings. The issue is that the number of available graphs for training a classification model is generally limited, even though each of the graphs can be very large. This causes poor performance to the methods that extract features using a supervised approach. The poor performance also propagates to the classification model building phase as the number of features is generally much larger than the number of training instances. This leads to the *fat matrix phenomenon* where the number of model parameters is considerably larger than the number of instances, and it is well known that such models are prone to over-fitting. In many of the existing works on machine learning, specifically in deep learning, artificial random noises are inserted to create distorted samples [7, 42], which greatly increases the training set size, and thus can alleviate the over-fitting problem. In [42], authors proposed a technique called "elastic distortion" to increase the number of image training data by applying simple distortions such as translations, rotations and skewing, which yield superior classification performance. However, in the existing works, no such mechanism is available for training data inflation for a graph classification task.

In this work, we propose a novel graph data representation for facilitating graph classification. Our feature representation is different from the existing works, and it solves the limitations of the existing graph classification methodologies that we have discussed in the earlier paragraphs. The first novelty of our feature representation is that we consider a graph as a bag of a uniform random subset of vertices of that graph such that each vertex in this set becomes an independent instance (a row) of a graph classification data set. Conceptually, each vertex in this feature representation is a distorted sample of the original graph from which the vertex is taken. This inflates the size of training data and substantially improves the performance of graph classification. One can also view such a data representation as multi-instance learning (MIL) [53]. However unlike MIL, for our data representation, during training phase each of the instances (in this case the vertices) in a bag assumes the same label which is the label of

Table 1 Real-life graphs

Domain	Datasets	(Vertex; edge)	Description
Animal	Bison	(26; 314)	Dominance between American bisons
	Hen	(32; 496)	Dominance between White Leghorn hens
	Dolphin	(62; 159)	Social network of bottlenose dolphins
	Kangaroo	(17; 91)	Interactions between free-ranging gray kangaroos
	Cattle	(28; 217)	Dominance behaviors observed between dairy cattles
	Zebra	(27; 111)	Interactions between Grevy's zebras
	sheep	(28; 250)	Dominance behavior between bighorn sheeps
	Macaques	(62; 1,187)	Dominance behavior between female Japanese macaques
Communication	UC Irvine messages	(1,899; 20,296)	Messages between students of UC, Irvine
	Enron	(87,273; 321,918)	Email communication between employees of Enron
	Digg	(30,398; 86,404)	Reply network of the social news website Digg
	FB Wall Post	(46,952; 274,086)	Subset of posts to other user's wall on Facebook
	LKML	(63,399; 242,976)	Communication network of the Linux kernel Mailing List
	EU institution	(265,214; 420,045)	Email communication of the undisclosed European institution
	U. Rovira email	(1,133; 5,451)	Email communication at the University Rovira i Virgili
Human Contact	Train bombing	(64; 143)	Contacts between terrorists involved in the train bombing
	Windsurfers	(43; 336)	Interpersonal contacts between windsurfers
	Infectious	(410; 2,765)	Face-to-face behavior of people during the exhibition
	Conference	(113; 2196)	Face-to-face contacts of the attendees in a conference
Coauthorship	arXiv hep-th	(22,908; 2,673,133)	Collaboration graph of arXiv's High Energy Physics—Theory
	arXiv astro-ph	(18,771; 198,050)	Collaboration graph of arXiv's Astrophysics section
	DBLP coauthorship	(317,080; 1,049,866)	Collaboration graph from DBLP computer science bibliography
	arXiv hep-ph	(28,093; 3,148,447)	Collaboration graph of arXiv's High Energy Physics—Phenomenology
Citation	arXiv hep-ph Cit.	(34,546; 421,578)	Citation graph of the arXiv's High Energy Physics—Phenomenology
	arXiv hep-th Cit.	(27,770; 352,807)	Citation graph of the arXiv's High Energy Physics—Theory
	Cora citation	(23,166; 91,500)	Cora citation network
	DBLP	(12,591; 49,743)	Citation graph of DBLP
Human Social	Jazz	(198; 2,742)	Collaboration network between Jazz musicians
	HighSchool	(70; 366)	Network contains friendships between boys highschool
	Residence hall	(217; 2,672)	Friendship between residents living at a residence hall
	Taro exchange	(22; 78)	Gift-givings (taro) between households in a Papuan village
	Dutch college	(32; 3,062)	Network contains friendships between university freshmen
	Sampson	(18; 188)	Network contains ratings between monks related to a crisis
	Zachary karate	(34; 78)	Network contains interaction between members of a karate club
	Seventh graders	(29; 376)	Network contains ratings between students from seventh grade
	Adolescent health	(2,539; 12,969)	Network was created from a adolescent health survey
	Tribes	(16; 58)	Social network of tribes of the Gahuku-Gama
Infrastucture	US-Airports	(1,574; 28,236)	Network of flights between US airports in 2010
	Air traffic control	(1,226; 2,615)	Network of preferred routes recommendations
	OpenFlights	(2,939; 30,501)	Network contains flights between airports of the world
	US power grid	(4,941; 6,594)	Network of power supply line between US power grids
	EuroRoad	(1,174; 1,417)	Road Network in Europe

the graph, whereas in MIL the instances in a bag can have different labels. Another difference from MIL is that for our case, the classification of a graph is determined by the majority voting of the vertex instances in the corresponding bag, but for MIL as long as one of the instances is of positive class, the bag is labeled as positive. The second novelty of our method is an unsupervised feature representation of each vertex utilizing deep neural network, similar to the one that is used for language modeling [29, 34]. Computing such features for a graph is faster than all the existing metric-based or subgraph-based feature representation, and they perform substantially better than the existing methodologies.

The idea of training data inflation using many distorted samples, in isolation, provides a significant performance boost for graph classification. To demonstrate this, we also propose another graph classification framework, which considers a graph as a bag of subgraphs. Given a graph, we partition the vertices of the graph; each of the partition-induced subgraphs then becomes an instance in a bag corresponding to that graph. To find features of these partition-induced subgraphs, we do not use language model-based feature embedding. Rather, we use existing metric-based approaches, which compute local/global topological features. In this work, we use seven topology-based (egonet, degree and clustering coefficient) features presented in [1] to represent each of the partition-induced subgraphs. Finding such topological metrics is costly on the entire graph, but it is cheap when it run over the partition-induced subgraphs yielding significant reduction in the execution time. Empirical evaluation over a large number of real-life graphs shows that training data inflation using graph partitioning is fast and robust, and it is substantially more accurate than the existing state-of-the-art graph classification methods.

We claim the following contributions:

- We propose two novel approaches for graph classification by training data inflation. In one approach, each sample in the inflated data is a randomly chosen vertex whose feature representation is obtained using a neural network-based language model. In another approach, each sample is a partition subgraph, whose feature representation is obtained through traditional graph topology metrics. Both the proposed methods are substantially better in terms of feature computation time and classification performance, specifically for large graphs.
- We empirically evaluate the performance of our proposed classification algorithms on multiple real-world datasets. To be precise, we use 43 real-life graphs from 7 different domains and classify these graphs into their respective domains.

2 Related Works

We discuss the related works in two different categories.

2.1 Graph Classification

In the area of data mining, Gonzalez et al. [13] are probably the first to address the problem of supervised graph classification. They propose an algorithm called `SubdueCL`, which finds discriminatory subgraphs from a set of graphs and uses these subgraphs as features for graph classification. Deshpande et al. [9] also use a similar approach for subgraph feature extraction for classifying chemical compounds using SVM. [31] proposed `DT-CLGBI`, which uses a custom-made decision tree for graph classification such that each node of the tree represents a mined subgraph. In all these works, features extraction is isolated from the classification task. A collection of follow-up works integrates the subgraph mining and graph classification in a unified framework. gBoost [39] is one of the earliest among these which uses mathematical programming. [11] use boosting decision stumps where a decision stump is associated with a subgraph. Other recent methods that use similar approach are `gActive` [23], `RgMiner` [22], `cogboost` [33], `GAIA` [21] and `Cork` [49].

Besides discriminating subgraphs, topological metrics are also used as features for graph classification. Li et al. [26] use 20 topological and label features, which include the degree, clustering coefficient, eccentricity, giant connected ratio, eigenvalues, label entropy and trace. Rahman et al. [36], use graphlet frequency distribution (GFD) to cluster graphs from various domains. In [28], authors compare graphs by using three metrics, called `Leadership` (it measures the extent to which the edge connectivity of a graph is dominated by a single vertex), `Bonding` a.k.a clustering coefficient and `Diversity` (its measurement is based on the number of edges, which share no common end points, and hence are disjoint.) In recent years, Berlingerio et al. [1] propose an algorithm called `NetSimile`, which computes features of a graph motivated from different social theories.

Kernel-based approaches are also popular for graph classification. Graph kernels are designed to exploit the shortest path [3], cyclic patterns [19], random walks [4], subgraphs [41, 50] and topological and vertex attributes [26]. Graph kernels compute the similarity between a pair of graphs, but this computation generally has high computational complexity; for example, the complexity of random walk kernel, a popular graph kernel, is $O(n^3)$, where n is the number of vertices. In summary, graph kernel-based methods do not scale for classification of large graphs.

2.2 Vertex Representation

There also exists a handful of solutions for the problem of "node classification" or "within-network classification." Some of these works use effective feature representation of the vertices for solving this task. Neville et al. [30] propose `ICA`, which is an iterative method based on constructing feature vectors for vertices from the information about them and their neighborhood. Henderson et al.'s method, called `ReFeX` [17], captures the behavioral feature of a vertex in the graph by recursively combining each node's local feature with their neighborhood (egonet-based) features. Koutra et al. [24] compare and contrast several guilt-by-association approaches for vertex representation. Tang et al. [47] propose to extract latent social dimensions based on network information and then use them as features for discriminative learning. In a follow-up work [48], they propose a framework called `SocioDim` which extracts social dimensions based on the network structure to capture prominent interaction patterns between nodes in order to learn a discriminative classifier. Han et al.[15] suggest that frequent neighborhood patterns can be used for constructing strong structure-aware features, which are effective for within-network classification task. Recently, Perozzi et al. [35] propose an algorithm for finding neighborhood-based feature representation of a vertex in a graph.

In the literature, another line of works exists to find feature representation of the vertices of graph [38, 40] that computes optimal embedding of a graph in Euclidean space that preserves topological properties, i.e., k-nearest neighbors. In [40], authors proposed an algorithm called "structure-preserving embedding" (SPE) that creates a low-dimensional set of coordinates for each vertex while perfectly encoding graph's connectivity. One of the crucial drawbacks of these embedding algorithms is that these methods are extremely memory intensive, and hence, they work well for small graphs only. We ran SPE [40] with several moderate size graphs (number of vertices is in 3 digits) in a 16 GB memory machine, but the processes were terminated due to insufficient memory error.

3 Method

Consider a graph database $\mathcal{G} = \{G_i\}_{1 \le i \le n}$. Each graph G_i is associated with a category label $L(G_i)$. For a graph G_i, we use $G_i.V$ and $G_i.P_k$ to denote the set of vertices, and the set of k-partitions of that graph, respectively. The task of supervised graph classification is to learn a classification model from a set of training graph instances. The main challenge of a graph classification task is to obtain a good feature representation for the graphs in $\mathcal{G}$ for solving this classification task.

Our solution to the feature representation for graph classification is to map a graph to a bag of multiple vertices or a bag of multiple subgraph instances, such that each of the instances in a bag becomes a distinct row in the classification training data. The instances in a bag inherit the label from the parent graph which they represent. Thus, if $v \in G_i.V$ is used as a bag instance for the graph G_i, the label of v is $L(G_i)$. If for a graph G_i all the vertices are used in the bag, one row of a traditional graph classification dataset becomes $G_i.V$ rows in our data representation each sharing the same label $L(G_i)$. Instead of vertices, we can also use partition-induced subgraphs as the bag instances. In this case, one row in a traditional graph classification dataset becomes $G_i.P_k$ rows in our data representation each sharing the same label $L(G_i)$. For a large input graph, we do not need to fill the bag with all the vertices of that graph, rather we can include only a random subset of vertices in the bag. In experiment section, we will show that the number of vertex instance that we take in a bag does not affect the classification performance significantly. For the case of partition-induced subgraph representation, we usually take all the partitions after choosing a reasonable number of partitions based on the size of the graph.

The immediate benefit of the above multi-instance feature representation is that such representation increases the number of rows in a classification dataset by providing multiple instances for each input graph. Given that most of the graphs are large with many vertices, multi-instance representation provides many-fold increase in the number of instances. This solves the fat matrix problem and thus obtains a robust graph classification model with higher accuracy. Furthermore, making many instances for one graph instance enables a learning algorithm to learn topological variances among different parts of the network, which also contributes to the model's accuracy. Many recent research works [7, 42] in deep learning community show the importance of training data inflation using the distorted copy of the input data sample. Our approach of multi-instance feature representation is a demonstration of such an endeavor for the task of graph classification.

Below we describe the feature representation of each instance in the bag. The next subsection will describe the vertex-based feature representation. In the subsection after that, we will discuss the feature representation of a partition-induced subgraph.

3.1 Vertex Feature Representation Using Random Walk

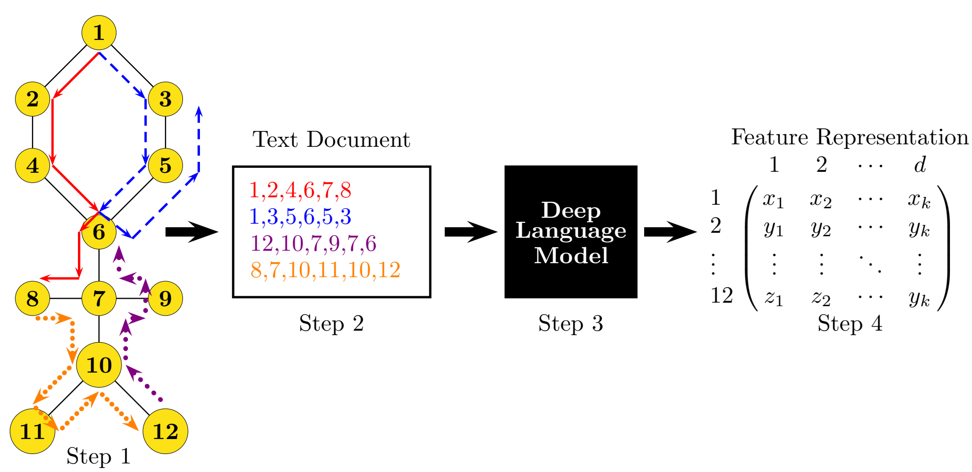
For our graph classification task, we assume that the nodes do not have a label or any other satellite data associated with them. So, the feature representation of a node v requires to capture the local topology around v. Following the DeepWalk method [35], we use a fixed length (say, l) random walk starting from the given node v (which we call root node) to build a sequence of nodes capturing the local topology around v. The method uniformly chooses an outgoing edge of currently visiting vertex until it makes l steps and builds a sequence of vertices that it visits through this walk. For each root node, the method performs t number of l length random walks. One can view each sequence of vertices as a sentence in a language, where the vertices are the words in that sentence. Given a set of sentences that are derived from a given root node, DeepWalk uses Word2Vec [29] embedding method to find a metric embedding of the given vertex in an appropriately chosen vector space. It finds the d-dimensional (d is user defined) feature representation of all vertices in the document. As we obtain the feature representation of each of the vertices in $\mathbb{R}^d$ using the above method, the embedding vectors become an instance of the bag of the given graph, G_i. The label of each of the vectors in this bag is $L(G_i)$.

The motivation of using random walk is twofold; first, the random walk is a cheap and efficient method for capturing local community structure around a graph node [44]. In [35], authors show that for power-law graphs, the frequency of vertices appearing in the random walks (constructed from the same graph) also follows a power-law distribution. Word frequency also follows similar distribution in the natural language context, so the language model is an appropriate choice for finding a metric representation of a node by considering the neighborhood around that node. Once the feature representation of the vertices is available, by using the vertices as a bag instance of a given graph we solve the graph classification problem.

In Fig. 1, we illustrate how we compute the feature representation of the vertices of an input graph $G(V, E)$. The toy graph that we use in this example has 12 vertices and 12 edges. At first, we uniformly select a set of target vertices; for each of these vertices, we will perform random walk of length l for t times. Suppose, we pick vertex 1, 8 and 12 and l is 5 and t is 2. The figure shows two random walks for vertex 1 only. We use different colors (red, blue, violate and orange) to illustrate different random walks. Once we have the random walks, we treat them as sentences in a document (step 2 in Fig. 1). In the third step, we pass the document in the text modeler (shown as black box in Fig. 1). As an output (step 4 in Fig. 1), text modeler produces feature representation of a given length (d-dimension) for all words, i.e., vertices $(1, 2, \ldots 12)$ in this case.

To give some perspective of adapting deep language modeler to model and find feature representation of the vertices of a graph, assume a sequence of words $W = \{w_0, w_1, \ldots, w_n\}$, where $w_i \in V$ (V is the vocabulary), a language model maximizes $\Pr[\mathrm{w_n}|\mathrm{w_0}, \mathrm{w_1}, \ldots, \mathrm{w_{n-1}}]$ over all the training corpus. Similarly, when we map random walks as sentences, estimated likelihood can be written as $\Pr[\mathrm{v_i}|\mathrm{v_0}, \mathrm{v_1}, \ldots, \mathrm{v_{i-1}}]$, which is the likelihood of observing a vertex v_i in a walk given all the previously visited vertices. This helps us to learn a mapping function $\mathcal{K}$, where $\mathcal{K} : v \in V \to \mathbb{R}^d$. Such mapping $\mathcal{K}$ embodies the latent topological representation associated with each vertex v in the graph. So the likelihood function becomes $\Pr[\mathrm{v_i}|\mathcal{K}(\mathrm{v_0}), \mathcal{K}(\mathrm{v_1}), \ldots, \mathcal{K}(\mathrm{v_{i-1}})]$. The deep language model (Word2Vec [29]) we used in this work adopts some of the recent relaxation to model the likelihood function. In our case, v_i in the likelihood function does not necessary be at the end of the context $(v_0, v_1, \ldots, v_{i-1})$, rather the context of a vertex consists of vertices appearing to the right of the given vertex in the random walk.

Fig. 1 Framework of vertex feature representation (best viewed in color)



3.2 Partition-Induced Subgraph Feature Representation

In another feature representation, we build the bag instances of a graph by partitioning the graph into different parts and then considering each part as a bag instance corresponding to that graph. Like the case of vertex multi-instance representation, each partition has the same label as the label of the parent graph. Since each partition is still a graph (but with a smaller size), we can use the existing metric-based approaches for its feature representation. Our main intention of using a partition-induced feature representation is to measure the effectiveness of training dataset inflation, irrespective of feature representation.

We choose seven features inspired by four social theories: (1) Coleman's Social Capital [8], (2) Burt's Structural Holes [5], (3) Heider's Balance [16], and (4) Homan's Social Exchange [18] according to the discussion in [1]. These theories, respectively, capture the connectivity of vertices and their neighborhoods, control of information flow, transitivity among the vertices, and reciprocity among the vertices. In the following, for a graph $G(V, E)$, we describe the seven features that we use in this work.

- d_u, degree of vertex u of G
- $d_{\text{nei}(u)} = \frac{1}{d_u}\sum_{v\in\text{nei}(u)} d_v$, average neighbor's degree of vertex u
- $|E_{\text{ego}(u)}|$, number of edges in node u's egonet.[1] $\text{ego}(u)$ returns node u's egonet.
- CC_u, clustering coefficient of node u which is defined as the number of triangles connected to vertex u over the number of connected triples centered at vertex u.
- $CC_{\text{nei}(u)} = \frac{1}{d_u}\sum_{v\in\text{nei}(u)} CC_v$, average clustering coefficient of vertex u's neighbors.
- $|E^{in}_{\text{ego}(u)}|$ = number of edge incident to $\text{ego}(u)$.
- $d_{\text{ego}(u)}$= degree of $\text{ego}(u)$ i.e., number of neighbors.

Note that the above features are for a vertex, so after extracting the above-mentioned seven features, we aggregate the features to compute the final feature vector of a graph. We use median, mean, standard deviation, skewness and kurtosis as aggregators. Similar to vertex multi-instance approach, once we obtain the feature representation of each of the partition-induced subgraphs, for a subgraph we obtain a $\mathbb{R}^d$ vector, which becomes an instance of the bag of the given graph, G_i. The label of the bag is $L(G_i)$.

3.3 Classification Model

For each graph G_i in the database $\mathcal{G}$, after generating numeric feature representation of its bag instances, we label each bag instance by the category label $L(G_i)$ of the parent graph. Since we attempt to solve a multi-class classification problem, we use multinomial logistic regression as the classification model $H_{\boldsymbol{\theta}}$, which is known as softmax regression in the literature. In multinomial logistic regression, the probability that a data point $x^i \in \mathbb{R}^d$ belongs to class j can be written as,

$$H_{\boldsymbol{\theta}}(x^i) = p(y^i = j|x^i; \boldsymbol{\theta}) = \frac{exp(\boldsymbol{\theta}_j^T x^i)}{\sum_{l=1}^{c} exp(\boldsymbol{\theta}_l^T x^i)},$$

where $j \in \{1, \ldots, c\}$ is the set of class labels, $\boldsymbol{\theta}_j$ is the weight vector corresponding to class j. The prediction of the model can be computed as, $\hat{y}_i = \text{argmax}_j\, p(y_i = j|x_i, \boldsymbol{\theta})$. Given a set of m labeled training instances $X = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, the weight matrix $\boldsymbol{\theta} \in c \times d$ (c is the number of class and d is the size of a feature vector) is computed by solving the convex optimization problem as shown in Eq. 1 using gradient descent. Note that, each data point $x^i \in \mathbb{R}^d$ in the training set corresponds to an instance in the bag of vertices or subgraphs.

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{argmin}}\, J(\boldsymbol{\theta}) &= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{c} \mathbf{1}_{\{y^i=j\}} \cdot \log\, p(y^i \\ &= j|x^i; \boldsymbol{\theta}) + \frac{\lambda}{2}\sum_{j=1}^{c}\sum_{k=1}^{d} \theta_{jk} \end{aligned} \tag{1}$$

$J(\boldsymbol{\theta})$ denotes the cost function which the classification model minimizes to find the optimal model parameter θ. $\mathbf{1}_{\{y^i=j\}}$ is the indicator function, indicating that only the output of the classifier corresponding to the correct class label is included in the cost. $\frac{\lambda}{2}\sum_{j=1}^{c}\sum_{k=1}^{d}\theta_{jk}$ is the regularization term to avoid over-fitting.

3.4 Pseudo-Code

In Fig. 2, we present the pseudo-code of the vertex multi-instance-based approach of graph classification. The input of the algorithm is a graph database $\mathcal{G}$ where each graph G_i is associated with a category label $L(G_i)$. Algorithm starts by iterating over each graph G_i. For each graph after populating random walks, it executes Word2Vec over the collection of walks to find the feature representation of the vertices of the graph and stores in $\text{Bag}(G_i.V)$ (lines 2–4). Then, the algorithm labels each of the instances in the $\text{Bag}(G_j.V)$ by the graph G_i's category label $L(G_i)$. At the end, $\text{Bag}(G_j.V)$ of labeled data instances is stored in a list called *Data*. When the iteration finishes, the algorithm applies k-fold cross-validation to split *Data* in the Bag level to generate the train($Data_{\text{train}}$) and test($Data_{\text{test}}$) fold. Algorithm then executes the training phase using $Data_{\text{train}}$ to train model $H_{\boldsymbol{\theta}}$ (line 6). Finally, the algorithm predicts the label of data points from the test fold ($Data_{\text{test}}$) using

[1] A vertex's egonet is the induced subgraph of its neighboring nodes.

Fig. 2 Vertex multi-instance graph classification algorithm

Vertex Multi-Instance Graph Classification($\mathcal{G}$):
comment: $\mathcal{G} = \{G_1, G_2, \cdots, G_n\}$ is the database of graphs from different domains,
1. $Data = [\,]$
2. **foreach** graph G_i in $\{G_1, G_2, \cdots, G_n\}$:
3. $\quad Walks_i$ = generate random walks from G_i
4. $\quad$ Bag($G_i.V$) = Word2Vec ($Walk_i$).
5. $\quad$ Add category label $L(G_i)$ to each data instance in Bag($G_i.V$)
6. $\quad$ Append Bag($G_i.V$) to $Data$
7. Apply $k-$fold cross validation over $Data$, use $Data_{train}$ to train softmax classification model $H_{\boldsymbol{\theta}}$
8. $Pred$ = Predict($H_{\boldsymbol{\theta}}$,$Data_{test}$)
9. Apply majority voting over $Pred$ to output graph label

Fig. 3 Partition multi-instance graph classification algorithm

Partition Multi-Instance Graph Classification($\mathcal{G}$):
comment: $\mathcal{G} = \{G_1, G_2, \cdots, G_n\}$ is the database of graphs from different domains,
1. $Data = [\,]$
2. **foreach** graph G_i in $\{G_1, G_2, \cdots, G_n\}$:
3. $\quad G_i.P_k$ = `Graclus`(G_i, k)
4. $\quad$ **foreach** induced subgraph $SG \in G_i.P_k$
5. $\qquad$ Bag($G_i.P_k$) = find seven topological features (SG).
6. $\quad$ Add category label $L(G_i)$ to each data instance in Bag($G_i.P_k$)
7. $\quad$ Append Bag($G_i.P_k$) to $Data$
8. Apply $k-$fold cross validation over $Data$, use $Data_{train}$ to train softmax classification model $H_{\boldsymbol{\theta}}$
9. $Pred$ = Predict($H_{\boldsymbol{\theta}}$,$Data_{test}$)
10. Apply majority voting over $Pred$ to output graph label

H_{θ} and outputs label of the test graphs represented by the bags of vertices in the test folds using majority voting (lines 7–8).

Figure 3 shows the pseudo-code of the partition multi-instance-based graph classification algorithm. Steps of the partition multi-instance technique are similar to the vertex multi-instance, except we use the graph partition algorithm to partition a graph G_i (line 3), say into k partitions. Then for each partition-induced subgraph in $G_i.P_k$, the partition multi-instance algorithm computes seven topology-based features in Bag($G_i.P_k$) (lines 4–5). Once the algorithm labels each instance in Bag($G_i.P_k$) with the corresponding graph category label and stores in a global list *Data*, it performs k-fold train-test scheme for Softmax classification (lines 6–9) and output label of the test graphs using majority voting (line 10).

4 Experiments and Results

To validate our proposed vertex and partition multi-instance graph classification algorithm, we perform several experiments. We use real-world graph data [46] from different domains in all our experiments. We have collected 43 graphs from 7 domains. In Table 1, we present basic statistics and short description of the graphs. As we can see, `Animal`, `Human Social` and `Human Contact` domains have smaller size graphs, whereas `Citation`, `Coauthorship`, `Communication` and `Infrastructure` contain moderate and large size graphs.

4.1 Experimental Setup

To find feature representation of the vertices of a graph, we use `gensim` library (https://radimrehurek.com/gensim/), which contains an open-source python implementation of "Word2Vec" algorithm. We write our own random walk generator using python. We set the length of the random walk (l) and feature vector size (d) to 40 and 30 for small and moderate size graphs (Animal, Human Contact, Human Social). For large size graphs (Citation, Collaboration, Communication and Infrastructure), we set these numbers to 60 and 70, respectively. For both cases, we set the number of random walk parameter (t) to 10. In Sect. 4.8, we discuss in detail the effects of parameter values on the performance of the classification task.

In the partition multi-instance-based approach, we use `NetSimile` by Berlingerio et al. [1] to compute features of the partition-induced subgraph constructed from each partition of a graph. We implement our own version of `NetSimile` in Python where all topological features are computed using Networkx [14] package. To partition the graphs in the dataset, we use `GraClus` by Kulis et al. [10]. `Graclus` takes the number of partition k as a user-defined parameter. We set the value of k to a small number for smaller graphs and reasonably high number for larger graphs. In this work, we choose k to be 60, 20 and 5, for large, moderate and small size graphs, respectively. We implement our own softmax classifier in Python. We set regularize parameter λ to $1e{-}4$ for all executions. We perform fourfold cross-validation over the data and use threefold to train, and onefold to test. To measure classifier's performance, we use percentage accuracy and micro-F1 metrics. We run all experiments in 3 GHz Intel machine with 16GB memory.

For performance comparison, we choose three representative graph classification methods. Two of these methods are proposed by Li et al. [26] and Berlingerio et al. [1]. In the forthcoming discussion, we denote Li et al.'s method as `Li` and Berlingerio et al's method as `NetSimile` (which is the name of their algorithm). `Li` is a topological feature-based approach, which works for a graph classification setting having a large number of small graphs. On the other hand, `NetSimile` represents the class of algorithms that is able to handle a small number of large graphs, similar to our problem setup. Given a graph, `Li` computes several (20) topological metrics (closeness, average degree, clustering coefficient, etc.) and uses these as features of the graph. The `NetSimile` computes seven local/global topological features (clustering coefficient, egonet size, degree of egonets, etc.) to capture connectivity, transitivity, reciprocity among the nodes and their neighbors. The third method that we compare against is `RgMiner`, proposed by Keneshloo et al. [22]. It is a frequent subgraph-based graph classification algorithm, which mines discriminatory subgraphs from the set of input graphs and then uses these subgraphs as features for graph classification. Note that a frequent subgraph mining-based algorithm is not scalable to handle large graphs, but for the sake of completeness, we compare our method with `RgMiner`, whenever possible. Besides classification accuracy, we also compare the execution time of these algorithms with our proposed methods.

4.2 Experiment on Classification Performance

In this experiment, we evaluate our models' performance on classifying graphs from different domains for both vertex and partition multi-instance technique. We also show the percentage improvement of these methods over existing best approach. We perform overall 14 classification tasks (mixture of binary, 3-class and 4-class classification) among different domains as shown in Column 1 of Table 2. For example, `Citation` versus `Coauthorship`, `Animal` versus `Human Social`, etc. The vertex multi-instance classification is an excellent method as we can see in Column 3 of Table 2; for example, in the coauthorship-communication task, this method achieves 97.0% accuracy. In Column 3 of the same table, we show the percentage of improvement on accuracy over current best approach (either `NetSimile` or `Li` because we are unable to use `RgMiner` over all graphs from Table 1) for

Table 2 Percentage accuracy of graph classifier

Domains	No. of class	Vertex multi-instance-based		Partition multi-instance-based		Current best method
		Accuracy (in %)	Improvement (in %) w.r.t current best	Accuracy (in %)	Improvement (in %) w.r.t current best	Accuracy (in %)
A-C	2	**85.1**	13.4	81.6	8.8	75
B-C	2	**97.0**	16.4	87.5	5.0	83.3
A-B	2	81.8	13.2	**83.3**	15.3	72.2
A-D	2	**89.0**	11.2	80.0	0	80.0
C-D	2	83.6	4.5	**91.6**	14.5	80.0
B-D	2	88.4	17.8	**97.5**	30.0	75.0
A-B-C	3	**85.2**	11.8	80.0	4.9	76.2
A-B-D	3	**85.1**	6.3	83.3	4.1	80.0
A-C-D	3	81.2	4.5	**81.3**	4.6	77.7
B-C-D	3	**88.1**	22.1	87.5	21.5	72.0
A-B-C-D	4	**80.0**	12.0	77.5	8.5	71.4
E-F	2	**83.4**	25.2	70.0	5.1	66.6
G-F	2	**75.0**	75.2	65.0	51.8	42.8
E-G-F	3	**65.0**	32.7	61.5	28.4	47.9

All best performing setups are indicated by bold

A—Citation, B—Coauthorship, C—Communication, D—Infrastructure, E—Animal, F—Human Contact, G—Human Social

Table 3 Micro-F1 (%) score of graph classifier

Domains	No. of class	Vertex multi-instance-based micro-F1 (%)	Partition multi-instance-based micro-F1 (%)
A-C	2	**86.7**	84.7
B-C	2	**93.2**	82.5
A-B	2	80.0	**81.2**
A-D	2	**91.2**	80.5
C-D	2	78.2	**92.1**
B-D	2	80.0	**92.2**
A-B-C	3	**85.4**	75.2
A-B-D	3	84.4	**85.1**
A-C-D	3	**82.1**	79.1
B-C-D	3	85.9	**88.1**
A-B-C-D	4	77.6	**77.9**
E-F	2	**78.8**	69.3
G-F	2	**71.4**	62.6
E-G-F	3	**61.4**	59.1

All best performing setups are indicated by bold

A—Citation, B—Coauthorship, C—Communication, D—Infrastructure, E—Animal, F—Human Contact, G—Human Social

the vertex multi-instance method. As we can see, vertex multi-instance approach achieves 4% to 75% improvement over the current best method. In Columns 4 and 5, we report percentage accuracy and percentage of improvement for the partition multi-instance method. This method also delivers excellent classification performance; for example, in the Coauthorship-Infrastructure task, it achieves 97.5% accuracy. The partition multi-instance approach achieves 4–51.8% improvement over current best method. In all 14 classification tasks, vertex multi-instance and Partition multi-instance approaches show superior performance over the existing methods. In the last column of Table 2, we show the performance of the best of the existing methods, either `NetSimile` or `Li`, whichever is better. In Table 3, we also report the average performance of the Vertex and Partition Multi-Instance-based graph classifier in terms of Micro-F1 score. (It is the harmonic mean of F1-scores for each class label.)

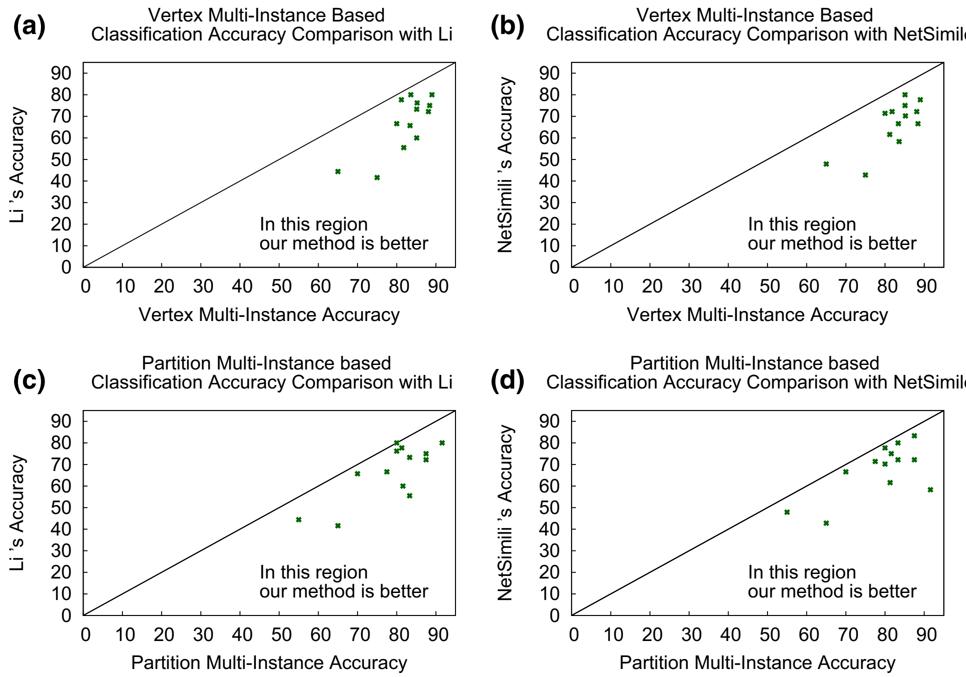
4.3 Comparison with the Existing Algorithms

In this experiment, we perform accuracy comparison with the existing algorithms (`Li` and `NetSimile`) of graph classification for all the graphs mentioned in Table 1. In order to make comparisons with a recent state-of-the-art frequent subgraph-based graph classification algorithm `RgMiner`[22], we pick animal, human social and human contact domains (Table 1) because these domains have smaller size graphs.

To illustrate the comparisons with `Li` and `NetSimile`, we create a scatter plot (Fig. 4) by placing the accuracy of our method in x-axis and the same of competitor's method in the y-axis for each classification task shown in Column 1 of Table 2. We also place a diagonal line in the plotting area. The number of points that lie in the lower triangle represents the number of tasks our methods are better than existing ones. Figure 4a, b shows performance comparison of the vertex multi-instance approach with `Li` and `NetSimile`. As we can see in all tasks, vertex multi-instance approach performs better than both `Li` and `NetSimile`. In Fig. 4b, d, we compare the accuracy of the partition multi-instance approach with `Li` and `NetSimile`. In this case, among 14 classification tasks our method performs better in 13, ties in 1. Superior performance of our proposed methods is essentially triggered by the training data inflation technique. Such technique helps to alleviate the fat-matrix phenomena (discussed in Sect. 1) and improves graph classification accuracy.

As discussed in Sect. 1, the execution of Gaston [32] over different combination of animal, human-social and human-contact settings for 30% support took more than 2 days. The same also holds true for the other subgraph feature-based methods, including GAIA and Cork. For subgraph mining using the above methods, we assume that the edges are unlabeled and all the vertices have the same label. This is a valid assumption because the edges of the graphs that we have used in this work do not possess any label except indicating a relationship; for example, bison graphs from animal domain portrait dominance relation between different bisons. We execute Gaston for 30% support over different combination of animal, human-social and human-contact settings. Once we have the frequent subgraphs, we apply `RgMiner` [22] to perform frequent subgraph-based graph classification. Average accuracies

Fig. 4 Classification accuracy comparison between vertex multi-instance with **a** `Li` and **b** `NetSimile`, partition multi-instance with **c** `Li` and **d** `NetSimile`



we got for "animal–human contact," "human contact–human social" and "animal–human contact–human social" settings are 52.3, 50.0 and 43.7%, respectively, which are lower than the accuracies reported by both of our proposed methods (see last 3 rows in Table 2) in the corresponding setups.

Finally, we want to show how the structure-preserving embedding (SPE) methods perform on smaller size graph classification. We use [40]'s algorithm for feature construction of each node in the graph. We choose animal-humanSocial setting for the experiment. We have to remove jazz, health and hall (Table 1) datasets from human-social category because SPE was unable to work for these graphs. Using SPE we gather 2-, 5- and 10-dimensional feature representation for each vertices of the graphs in animal and human-social category. Then using the classification algorithm discussed in Sect. 3.4, we compute the percentage of accuracy. The percentage of accuracies we got are 50.3, 46.4 and 41.6% for 2, 5 and 10 dimension, respectively. The reason for poor classification performance of the SPE method is that low-dimensional representation computed by the SPE method can only capture immediate (1-hop) neighborhood information of a node, whereas in order to classify a graph we want to have topological information of a node span over more than one hop.

4.4 Experiment with Vertically Scaled Dataset

In earlier experiments, we have shown that our proposed methods are particularly suitable for a horizontally scaled dataset—in such a dataset, the number of graphs is small, but each of the graphs is large in size. For example, the average number of vertices in the Citation, Communication and Coauthorship networks is 24K, 62K and 96K, respectively; for these graphs, our methods perform the best over all the existing methods. Note that for these datasets the frequent/discriminative subgraph-based methods are not able to run due to their excessive computation cost. On the other hand, if the graph dataset is vertically scaled, i.e., if there are many graphs in the dataset but each of the graphs is small in size, then the existing subgraph-based methods generally work well. To show this, we consider a well-known discriminative subgraph-based approach, namely *gboost*[2] and run it on breast cancer (MCF-7) dataset in the National Cancer Institute (NCI) graph data repository.[3] On this dataset, *gboost* has an F-Score value of 0.75, whereas the best F-score among our proposed methods is 0.65. This inferior performance of our proposed methods in MCF-7 dataset is expected. In this dataset, the graphs, on average, have 26 nodes and 28 edges. On such small graphs, the random walk-based vertex embedding or partition-based multi-instance learning is unable to capture the topological properties of the graphs which are suitable for classification.

[2] Matlab implementation of gboost is publicly available from http://www.nowozin.net/sebastian/gboost/.

[3] MCF-7 dataset is available to download from https://www.cs.ucsb.edu/~xyan/dataset.htm.

Table 4 Average running time in second (*VML* vertex multi-instance, *PML* partition multi-instance)

Domain	Avg. vertex size	Time [26]'s (s)	Time [1]'s (s)	Time VML (s)	Time PML (s)
Animal	35	**0.05**	0.135	0.09	0.08
Human contact	157	1.71	0.69	0.45	**0.17**
Human social	317	2.73	0.38	1.1	**0.32**
Infrastucture	2K	27.7	18.2	8.2	**1.5**
Citation	24K	8634.7	1142.8	121.6	**89.1**
Communication	62K	41217.8	2136.4	289.1	**254.6**
Coauthorship	96K	137106.4	117811.5	**562.4**	8367.1

All best performing setups are indicated by bold

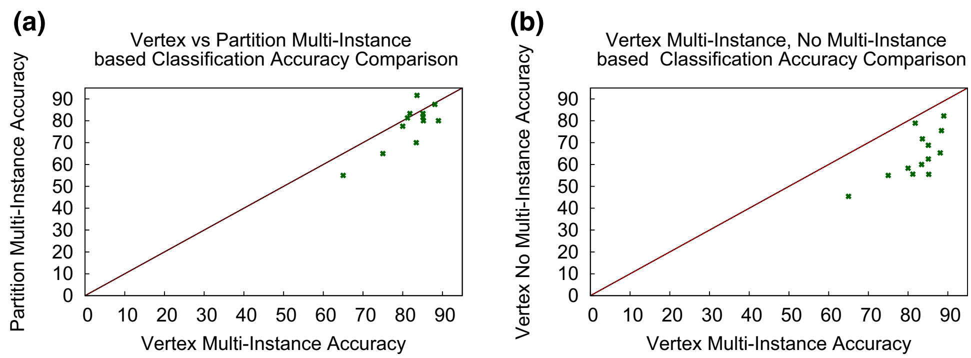


Fig. 5 Accuracy comparison between **a** vertex versus partition multi-instance and **b** vertex multi-instance versus no multi-instance-based approach

4.5 Timing Analysis

In this section, we perform the timing analysis of our proposed algorithms and compare with the existing ones. To report runtime performance of an algorithm, we group execution times of the algorithm over graphs by its (graphs) category, i.e., citation, collaboration and report average running time. In Table 4, we show average running time in seconds. For the sake of comparison, we only report running time of finding feature representation for all the algorithms except for the partition multi-instance method, where we incorporate running time of partitioning as well. As we can see that for smaller graphs, all the methods finish within a reasonable time frame. However, for large graphs specially in the Citation, Coauthorship and Communication category, running time is very high for `Li` and `NetSimile`. For communication and Coauthorship category graphs, the vertex multi-instance method achieves 142- and 243-fold improvement over `Li` and 7 and 209 time improvement over `NetSimile`, respectively. The partition multi-instance approach achieves 162- and 16-fold improvement over `Li` and `NetSimile` for communication but only 16 and 14 for Coauthorship domain. Note that we could execute `RgMiner` just for the smaller size graphs from animal, human contact and human-social category, and such executions solely will not be enough to portray the complete picture on the comparisons over the running time. So, we decide not to report `RgMiner`'s running time in Table 4. Nevertheless, it takes 17.2, 15.8 and 21.3 s for `RgMiner` to mine frequent subgraphs using Gaston for 30% support and finding feature representation for "animal–human contact," "human contact–human social" and animal–human contact–human social" setting, respectively.

4.6 Effectiveness of Training Data Inflation

In earlier experiments, we see that both the vertex and partition multi-instance methods perform better than the existing algorithms. The vertex multi-instance method incorporates training data inflation along with the deep learning-based feature representation, and the partition multi-instance uses training data inflation with the existing metric-based feature representation. In this experiment, we want to investigate whether the superior performance of our proposed methods can be attributed to training data inflation or deep learning-based representation of vertices. To do this, we populate scatter plot (Fig. 5) similar to Fig. 4, by placing the vertex multi-instance method's accuracy in x-axis, and the partition multi-instance method's accuracy in y-axis. As shown in Fig. 5, all the points in the plotting area are very close to the diagonal line, which establishes competitive performance between these two methods. Training data inflation is the common part between these two approaches. Moreover, the partition multi-instance method shows that improved graph classification performance is achievable without deep learning-based techniques. So training data inflation-based paradigm is attributed more toward better performance of our methods of graph classification. However, for large graphs vertex multi instance-based classification may be more attractive due to its smaller running time for constructing the feature representation of a graph.

To investigate further, we perform a graph classification experiment without leveraging the training data inflation. After obtaining deep learning-based feature representation of the vertices of a graph, we apply five aggregator functions: mean, median, kurtosis, standard deviation and variance over each feature and derive a single feature vector for the bag. Then, we use these single feature vectors per graph, i.e., bag to train the classification model. We perform this experiment for all classification settings mentioned in Table 2. In Fig. 5, we compare the classification performance for Multi-Instance (x-axis)- and No Multi-Instance (y-axis)-based approach using scatter plot. As we can see, for all cases, all the points reside in the lower triangle of the plotting area, hence establishing superior performance of Multi-Instance-based approach over No Multi-Instance.

4.7 Experiment on Bag size

As we mentioned in Sect. 3 that for large graphs, the number of vertex instances in a bag does not affect much on the classification accuracy. To empirically validate this claim, in the citation-communication setting, we measure classification accuracy for bag size of 1000, 2000, 3000 and 4000 vertices. As we can see in Fig. 6a, accuracy does not change after 2000. In case of the partition multi-instance approach, to increase the number of instances (subgraphs) in a bag, we have to create more partitions of a graph. Large number of partition causes smaller size partition-induced subgraphs and face potential chance of not capturing important local neighborhood topologies. In Fig. 6b, we can see that in the citation-communication setting, prediction accuracy decreases slightly as the number of partition increases beyond an optimal value. Also when we set the number of partition to a small value performance decreases because for small partition, number of training data (row instances) remains small.

4.8 Parameter Selection for Vertex Multi-instance Approach

In the vertex multi-instance approach, to find a better feature representation of the vertices of an input graph, we have to tune three important parameters. One is the dimension of the feature vector (d), second one is the length of the random walk (l) and third one is the number of random walks per vertex (t). In this experiment, we analyze the sensitivity of these three parameters. We choose animal–humanContact from small/moderate size graphs and Communication Infrastructure from large graphs. First, we want to see how the dimension size affects the overall performance of the classifier for smaller graphs on “animal–humanContact” setting. We execute the vertex multi-instance algorithm for different d values ranging from 20 to 60 and record classifier’s performance (percentage of accuracy). We set the length of random walk to 40. We perform the same experiment for large graphs on “citation-communication” setting while varying d values from 20 to 90. We fix the random walk length to 60 in this case. The number of random walk parameter is set to 10 for both small and large size graphs. In Fig. 7a, c, we plot the percentage of accuracy for different d values. As we can see for lower and higher values of d, the classifier’s performance is not satisfactory. Smaller d is unable to capture topological information that is local to a node, whereas larger d causes feature representation of the vertices be more generalized, which in turn decreases discriminative property hence downfall in overall accuracy. The best performance we get is $d = 30$ for small graphs and $d = 70$ for large graphs.

To see the effect of the length of random walk (l), we perform the same experiment as above but with different lengths of random walk ranging from 20 to 60, while keeping the dimension size to $d = 30$ for small/moderate graphs. For large graphs, we set l from 20 to 90 while fixing d to 70. For both cases, the number of random walk (t) is set to 10. In Fig. 7b, d, we plot the percentage of accuracy across different random walk lengths. Walk length parameter l has a similar effect as d. For small l, local topological information of a node is less captured and

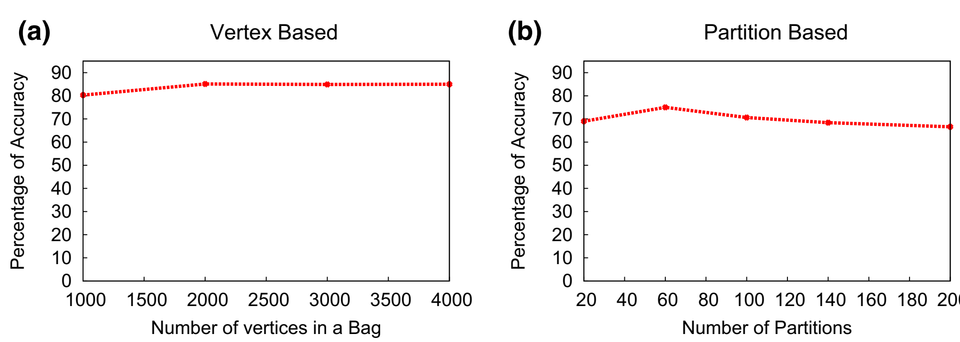


Fig. 6 Effects of bag size over classification accuracy **a** vertex multi-instance **b** partition multi-instance

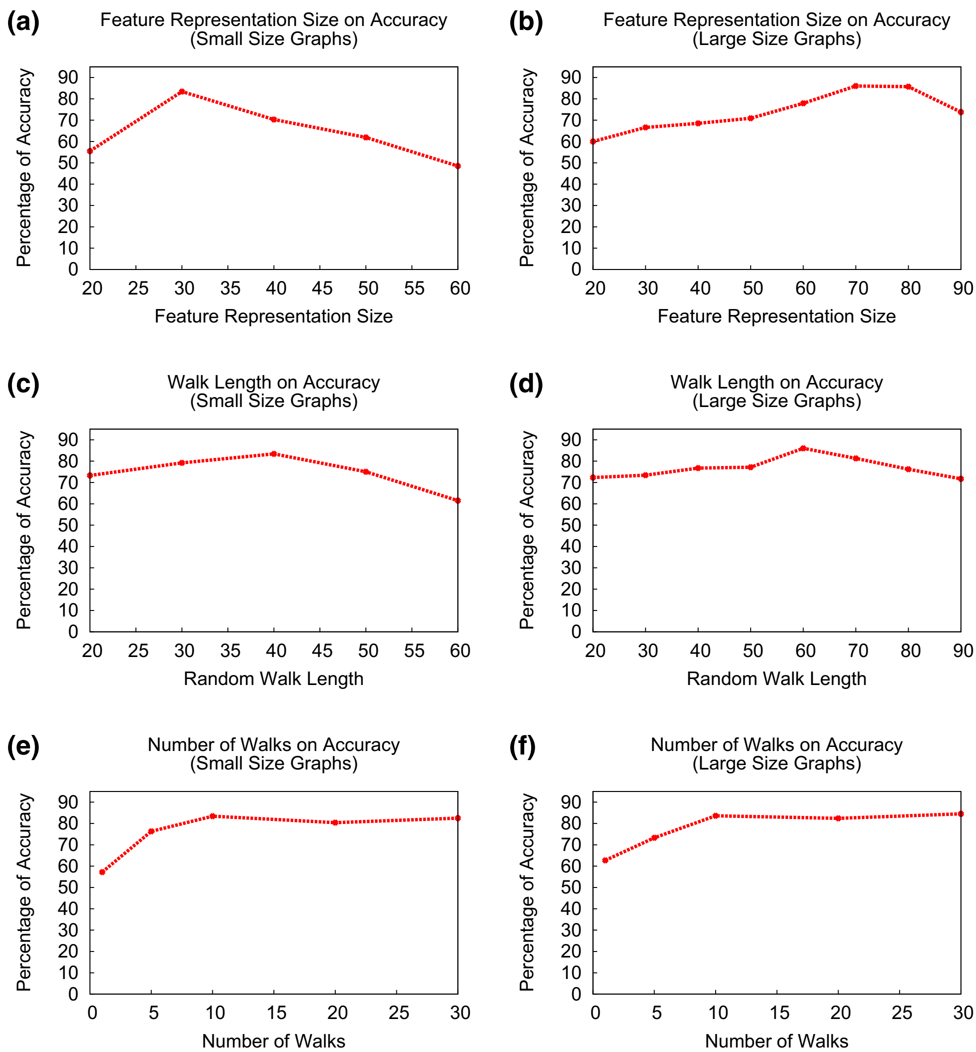


Fig. 7 Effects of dimension size (**a**, **b**), random walk length (**c**, **d**) and number of walks (**e**, **f**) parameter over classification accuracy for vertex multi-instance approach

large l capture more topological information with respect to the entire graph than a node. We get best performance for $l = 40$ and $l = 60$ for small and large graphs, respectively.

Finally, we perform the same experiment as above for the different numbers of random walk (t) parameter ranging from 1 to 30 for both small and large size graphs. In Fig. 7e, f, we plot the percentage of accuracy across different numbers of random walk. As we can see, after $t = 10$, the classifier's performance does not vary for both small and large size graphs. Moreover, when we set the number of walks to a higher value, overall training time of the model increases. So, we set the number of random walk parameter t to 10 for all experiments we perform in this research.

5 Conclusions

In this work, we propose two novel solutions of the graph classification problem. In the vertex multi-instance solution, we map a graph into a bag of vertices and leverage neural network-based representation learning technique to find feature representation of the vertices. In the partition multi-instance solution, we map a graph into a bag of subgraphs and use traditional metric-based feature representation technique to construct features of the partition-induced subgraphs. We perform extensive empirical evaluations of our proposed methods over several real-world graph data from different domains. We compare our algorithms with the existing methods of graph classification and show that our methods perform significantly better on classification accuracy as well as running time.

Acknowledgements Funding for this work is provided by United States National Science Foundation (Grant No. IIS-1149851).

References

1. Berlingerio M, Koutra D, Eliassi-Rad T, Faloutsos C (2013) Network similarity via multiple social theories. In: Proceedings of the 2013 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM'13), pp 1439–1440
2. Bordes A, Glorot X, Weston J, Bengio Y (2012) Joint learning of words and meaning representations for open-text semantic parsing. In: International conference on artificial intelligence and statistics, pp 127–135
3. Borgwardt KM, Kriegel HP (2005) Shortest-path kernels on graphs. In: Proceedings of the fifth IEEE international conference on data mining (ICDM'05), pp 74–81
4. Borgwardt KM, Schraudolph NN, Vishwanathan S (2007) Fast computation of graph kernels. In: Schölkopf B, Platt J, Hoffman T (eds) Advances in neural information processing systems, vol 19, pp 1449–1456
5. Burt RS (2009) Structural holes: the social structure of competition. Harvard University Press, Harvard
6. Cheng H, Lo D, Zhou Y, Wang X, Yan X (2009) Identifying bug signatures using discriminative graph mining. In: Proceedings of the eighteenth international symposium on software testing and analysis, pp 141–152
7. Ciresan D, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: 2012 IEEE conference on computer vision and pattern recognition (CVPR), pp 3642–3649
8. Coleman JS (1986) Individual interests and collective action: selected essays. Cambridge University Press, Cambridge
9. Deshpande M, Kuramochi M, Wale N, Karypis G (2005) Frequent substructure-based approaches for classifying chemical compounds. IEEE Trans Knowl Data Eng 17(8):1036–1050
10. Dhillon I, Guan Y, Kulis B (2005) A fast kernel-based multilevel algorithm for graph clustering. In: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining, pp 629–634
11. Fei H, Huan J (2014) Structured sparse boosting for graph classification. ACM Trans Knowl Discov Data 9(1):4:1–4:22
12. Gascon H, Yamaguchi F, Arp D, Rieck K (2013) Structural detection of android malware using embedded call graphs. In: Proceedings of the 2013 ACM workshop on artificial intelligence and security, pp 45–54
13. Gonzalez JA, Holder LB, Cook DJ (2002) Graph-based relational concept learning. In: Proceedings of the nineteenth international conference on machine learning (ICML'02), pp 219–226
14. Hagberg AA, Schult DA, Swart PJ (2008) Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th python in science conference (SciPy2008). Pasadena, CA, USA, pp 11–15
15. Han J, Wen JR, Pei J (2014) Within-network classification using radius-constrained neighborhood patterns. In: Proceedings of the 23rd ACM CIKM, pp 1539–1548
16. Heider F (2013) The psychology of interpersonal relations. Wiley, London
17. Henderson K, Gallagher B, Li L, Akoglu L, Eliassi-Rad T, Tong H, Faloutsos C (2011) It's who you know: graph mining using recursive structural features. In: Proceedings of the 17th ACM SIGKDD, KDD'11
18. Homans GC (1958) Social behavior as exchange. Am J Sociol 63(6):597–606
19. Horváth T, Gärtner T, Wrobel S (2004) Cyclic pattern kernels for predictive graph mining. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD'04), pp 158–167
20. Jiang C, Coenen F, Zito M (2013) A survey of frequent subgraph mining algorithms. Knowl Eng Rev 28(01):75–105
21. Jin N, Young C, Wang W (2010) Gaia: graph classification using evolutionary computation. In: Proceedings of the 2010 ACM SIGMOD international conference on management of data. ACM, pp 879–890
22. Keneshloo Y, Yazdani S (2013) A relative feature selection algorithm for graph classification. In: Advances in databases and information systems, advances in intelligent systems and computing, vol 186, pp 137–148
23. Kong X, Fan W, Yu PS (2011) Dual active feature and sample selection for graph classification. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'11), pp 654–662
24. Koutra D, Ke TY, Kang U, Chau DH, Pao HKK, Faloutsos C (2011) Unifying guilt-by-association approaches: theorems and fast algorithms. In: Proceedings of the 2011 European conference on machine learning and knowledge discovery in databases—Volume Part II, pp 245–260
25. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Proceedings of the IEEE international conference on data mining, 2001 (ICDM 2001). IEEE, pp 313–320
26. Li G, Semerci M, Yener B, Zaki MJ (2012) Effective graph classification based on topological and label attributes. Stat Anal Data Min 5(4):265–283
27. Liu F, Liu B, Sun C, Liu M, Wang X (2013) Deep learning approaches for link prediction in social network services. In: Lee M, Hirose A, Hou Z-H, Kil RM (eds) Neural information processing, vol 8227. Springer, Berlin, pp 425–432
28. Macindoe O, Richards W (2010) Graph comparison using fine structure analysis. In: Proceedings of the 2010 IEEE second international conference on social computing, pp 193–200
29. Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality. Adv Neural Inf Process Syst 26:3111–3119
30. Neville J, Jensen D (2000) Iterative classification in relational data. In: Proceedings of the AAAI, pp 13–20
31. Nguyen PC, Ohara K, Mogi A, Motoda H, Washio T (2006) Constructing decision trees for graph-structured data by chunkingless graph-based induction. In: Proceedings of the 10th Pacific-Asia conference on advances in knowledge discovery and data mining (PAKDD'06), pp 390–399
32. Nijssen S, Kok J (2004) A quickstart in frequent structure mining can make a difference. In: Proceedings of the ACM SIGKDD
33. Pan S, Wu J, Zhu X (2015) Cogboost: boosting for fast cost-sensitive graph classification. IEEE Trans Knowl Data Eng 27(11):2933–2946
34. Pennington J, Socher R, Manning CD (2014) Glove: global vectors for word representation. In: Proceedings of the empirical methods in natural language processing (EMNLP 2014), vol 12, pp 1532–1543
35. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD, pp 701–710
36. Rahman M, Bhuiyan MA, Al Hasan M (2014) Graft: an efficient graphlet counting method for large graph analysis. IEEE Trans Knowl Data Eng 26(10):2466–2478
37. Ranu S, Hoang M, Singh A (2013) Mining discriminative subgraphs from global-state networks. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining, pp 509–517
38. Reiterman J, Rödl V, Šiňajová E (1992) On embedding of graphs into Euclidean spaces of small dimension. J Comb Theory Ser B 56(1):1–8

39. Saigo H, Nowozin S, Kadowaki T, Kudo T, Tsuda K (2009) gboost: a mathematical programming approach to graph classification and regression. Mach Learn 75(1):69–89
40. Shaw B, Jebara T (2009) Structure preserving embedding. In: Proceedings of the 26th annual international conference on machine learning. ACM, pp 937–944
41. Shervashidze N, Petri T, Mehlhorn K, Borgwardt KM, Vishwanathan S (2009) Efficient graphlet kernels for large graph comparison. In: Proceedings of the twelfth international conference on artificial intelligence and statistics (AISTATS-09), vol 5, pp 488–495
42. Simard P, Steinkraus D, Platt JC (2003) Best practices for convolutional neural networks applied to visual document analysis. In: Proceedings of the seventh international conference on document analysis and recognition, 2003, pp 958–963
43. Socher R, Huang EH, Pennin J, Manning CD, Ng AY (2011) Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In: Advances in neural information processing systems, pp 801–809
44. Spielman DA, Teng SH (2004) Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of the thirty-sixth annual ACM symposium on theory of computing, STOC'04, pp 81–90
45. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2014) Going deeper with convolutions. CoRR arXiv:1409.4842
46. The Koblenz Network Collection (konect) (2015) http://konect.uni-koblenz.de/networks/
47. Tang L, Liu H (2009) Scalable learning of collective behavior based on sparse social dimensions. In: Proceedings of the 18th ACM conference on information and knowledge management (CIKM'09), pp 1107–1116
48. Tang L, Liu H (2011) Leveraging social media networks for classification. Data Min Knowl Discov 23(3):447–478
49. Thoma M, Cheng H, Gretton A, Han J, Kriegel HP, Smola A, Song L, Yu PS, Yan X, Borgwardt K (2009) Near-optimal supervised feature selection among frequent subgraphs. In: Proceedings of the 2009 SIAM international conference on data mining. SIAM, pp 1076–1087
50. Thoma M, Cheng H, Gretton A, Han J, Kriegel HP, Smola A, Song L, Yu PS, Yan X, Borgwardt KM (2010) Discriminative frequent subgraph mining with optimality guarantees. Stat Anal Data Min 3(5):302–318
51. Wawer M, Peltason L, Weskamp N, Teckentrup A, Bajorath J (2008) Structure–activity relationship anatomy by network-like similarity graphs and local structure–activity relationship indices. J Med Chem 51(19):6075–6084
52. Yan X, Cheng H, Han J, Yu PS (2008) Mining significant graph patterns by leap search. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM, pp 433–444
53. Zhou ZH, Zhang ML, Huang SJ, Li YF (2012) Multi-instance multi-label learning. Artif Intell 176(1):2291–2320