THE UNIVERSITY OF EDINBURGH

DOCTORAL THESIS

Graph Mining on Static, Multiplex and Attributed Networks

Benedek Rózemberczki

Supervisor:
Dr. Rik Sarkar

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Center for Doctoral Training in Data Science
School of Informatics

June 3, 2021

Declaration of Authorship

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. Part of the materials in this dissertation have been published in the following publications, where the contribution from fellow PhD students and researchers was limited to help with proofs and running experiments:

1. **Benedek Rózemberczki**, Ryan Davies, Rik Sarkar, and Charles Sutton. *GEMSEC: Graph Embedding with Self Clustering.* Published in the Proceedings of the ACM International Conference on Advances in Social Network Analysis and Mining (ASONAM '19).
2. **Benedek Rózemberczki**, Carl Allen, and Rik Sarkar. *Multi-Scale Attributed Node Embedding.* Published in the Journal of Complex Networks.
3. **Benedek Rózemberczki** and Rik Sarkar. *Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models.* Published in the Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20).
4. **Benedek Rózemberczki**, Péter Englert, Amol Kapoor, Martin Blais, and Bryan Perozzi. *Pathfinder Discovery Networks for Neural Message Passing.* Published in the Proceedings of the 2021 ACM Worldwide Web Conference (WWW '21).
5. **Benedek Rózemberczki** and Rik Sarkar. *The Shapley Value of Classifiers in Ensemble Games.* Under Review in CIKM '21.
6. **Benedek Rózemberczki**, Olivér Kiss, and Rik Sarkar. *Karate Club: An API Oriented Open-Source Python Framework for Unsupervised Learning on Graphs.* Published in the Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20).
7. **Benedek Rózemberczki**, Olivér Kiss, and Rik Sarkar. *Little Ball of Fur: A Python Library for Graph Sampling.* Published in the Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20).

(Benedek Rózemberczki)

Signed:

_____

The University of Edinburgh

Abstract

School of Informatics

Doctor of Philosophy

Graph Mining on Static, Multiplex and Attributed Networks

by Benedek Rózemberczki

Graph structured data is pervasive and generated by online human interactions at an unprecedented velocity. Sophisticated features encoded by relational data, such as structure and relative proximity have unparalleled expressiveness to describe the complex human systems that generated the data. Relational data poses challenges for information extraction and knowledge discovery due to its web scale size, extreme sparsity, multimodality, the presence of spatial autocorrelation and heterogeneity. The main goal of this thesis is to introduce principled and applicable graph mining algorithms and software packages which can tackle these challenges. Novel techniques introduced in this thesis were influenced by ideas in unsupervised machine learning, statistics and game theory. These algorithms exploit structural and proximity-based context by decomposing feature matrices, defining novel classes of characteristic functions and differentiable parametric statistical models. Proposed methods are designed to be scalable and memory efficient. The utility of machine learning methods and software packages introduced in the thesis is established by mathematical proofs and experiments that use real world networks and synthetic graphs.

KEYWORDS: node embedding, graph embedding, model pruning, ensemble learning

Lay Summary

Our world is increasingly interrelated and complex connections are being weaved around us. These relations can describe the way people interact, how goods are being transported, the journey of financial transactions and the reason why ecological chains can collapse. Graph structured data describes these important real world relations. The process of extracting information and knowledge from graph structured data is a particularly challenging task for multiple reasons. The main challenge is the amount of data and the extreme sparsity of relations. For example, in a social network billions of people are connected, yet the majority of people has no connections with each other. Other challenging aspects of dealing with graph structured data include the time dimension and heterogeneity of relationships which all add layers of complexity to the task at hand. In this dissertation we discuss novel principled algorithms that can solve a large number of important graph data processing problems in an efficient way with these challenges in mind.

Nodes in real world networks frequently have generic attributes. For instance in a social network attributes of users can include the content they liked, their gender and age. These node attributes can be numerous and the distribution of these attributes in the neighbourhood of nodes is of prime interest for analysts. Dealing with node features is not a trivial task. First, we introduce an algorithm which can extract node features that describe the global location of nodes in the graph while the natural community structure is respected. Second, we describe an algorithm which describes the presence of vertex attributes at local and global scale with learned features. Third, we define a novel technique which can characterize the distribution of continuous features in neighbourhoods such as age in a principled way.

Edge features and the presence of multiple types of edges between vertices is also a fairly common phenomenon in real world networks. For example in an online social network the users can interact with each other in a number of fundamentally different ways such as liking the same content, following and messaging each other. The fourth contribution of this thesis is a machine learning model which can learn to combine these edges and edge features in a meaningful way in order to predict interesting properties of nodes.

Statistical models such as the graph mining techniques which we introduce in the thesis are commonly combined together to make decisions. However, the contribution of various statistical models to the decisions of a combined model is hard to quantify. As a fifth contribution we developed a game theory based method to attribute the decisions to specific models and we propose an efficient algorithm to estimate this model importance. We demonstrate the effectiveness of the introduced technique on various real world problems. The last two contributions in the thesis are the design of software libraries which foster research on graph structured data. The first software package provides a range of graph mining techniques in a single unified framework. The second library covers graph sampling algorithms which can extract representative subgraphs.

Acknowledgements

First, I would like to express my eternal gratitude to my primary supervisor *Rik Sarkar*, who supported and guided me during my doctoral studies. Our discussions with *Rik*, his patience and constructive criticism helped me to become a better researcher. I feel fortunate and privileged that we worked together with him. I am also grateful to *He Sun*, *Charles Sutton* and *Valerio Restocchi*, who participated in my PhD reviews and gave me detailed feedback about research directions and scientific communication skills.

I was privileged to collaborate with *Carl Allen*, *Ryan Davies* and *Olivér Kiss*. They have all contributed to the quality, depth and merits of this thesis. I am thankful to *Carl* for his contributions to the theory of attributed node embedding and its relations to proximity-preserving node representation learning. His expertise on implicit factorization techniques was a major influence on my thinking. Our discussions with *Ryan* about community detection considerably improved my understanding of latent space clustering. Multiple chapters in thesis are based on collaborations with *Olivér* who suggested remarkable experiments and his ideas shaped the texture of this thesis.

During my doctoral training I was fortunate enough to do a research visit in New York and collaborate with the Google AI Research graph mining group. Working with *Martin Blais, Péter Englert, Amol Kapoor* and *Bryan Perozzi* helped me in conceptualizing research ideas about multiplex graph representation learning which are integral part of this thesis. I am especially grateful for our discussions with *Bryan* about multi-scale graph and node representations which greatly influenced my thinking and the research presented in this thesis.

The *School of Informatics* created a vibrant and lively environment for discussing research and provided opportunities to interact with brilliant and kind human beings. The members of our research group *Abhirup Ghosh*, *Maria Astefanoaei*, *Lauren Watson*, *Panagiota Katsikouli* and *Rayna Andreeva* had a tremendous impact on my thinking and creativity. I would like to express my special gratitude to *Lauren* who gave critical and succinct feedback on my manuscripts. As a member of the *Centre for Doctoral Training in Data Science* I was surrounded by friends like *Carl Allen*, *Ivana Balazevic*, *Artur Bekasov*, *Conor Durkan*, *Simao Eduardo*, *Maria Gorinova*, *Zack Hodari*, *Jonathan Mallinson* and *James Owers* who encouraged me and provided much needed emotional stamina throughout these years. Our office always a felt like a home thanks to the presence of *Paul Piho*, *Michael Schlichtkrull*, *Philippa Shoemark* and *Ludovica Vissat*.

Finally, I am thankful to my *family* for their continuing support and encouragement in this wonderful endeavour. I must express my gratitude and love to my wife *Enikő*. She is my greatest emotional support and she made enormous sacrifices that helped me to succeed in this journey. Her contribution to this thesis cannot be appropriately quantified and appreciated.

Dedication

To my family.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A graph is a mathematical construct which can compactly model real world networks including social interactions, food chains, the texture of materials, molecules, financial transactions and the web. Graphs consist of vertices and edges that serve as links between the vertices. For instance, in a social network vertices can model human beings and the existence of edges can describe that two people have some form of interaction. By augmenting the basic model of graphs with node and edge attributes, multiplexity, heterogeneity, temporal dimension, and dynamics one can provide rich descriptions of a real world network. The analysis, processing and understanding of such graphs requires the rigorous and principled design of appropriate algorithmic tools. *Graph mining*, which is a subfield of data mining, considers the design of efficient and explainable algorithms that can extract knowledge and information from graphs that describe complex real world networks [86].

Traditional *network science* and graph mining are interrelated, but differentiated from each other as the former focuses on the understanding and analysis of the complex systems which generate the observed graph [16, 42, 48]. In a social network setting a network scientist would investigate: why does an individual's age affect their location in the social network? Comparatively a graph mining researcher in the same setting would focus on a research question such as: how can we design an efficient and precise algorithm for predicting the age of users based on their location in the social network? The answer to this question would be influenced by factors such as the characteristics of the available data, potential computational and hardware constraints and the exact definitions of efficiency and precision. This alludes to the multidisciplinary and intersectional nature of graph mining research.

The design and theoretical foundation of algorithms that distill applicable knowledge from graph structured data adopts ideas from intersecting scientific fields. For example, the graph mining algorithms developed as part of this dissertation expand and generalize fundamental concepts from Fourier analysis, natural language processing, machine learning and game theory. However, the main results and ideas presented in this dissertation are also interpretable in the context of the intersecting scientific domain not just in the sole context of graph mining. It is worth emphasizing that the potential spillover of graph mining research reaches beyond these intersecting domains. The pervasive nature of graph structured data [16] and the high practical applicability of graph mining research [86] helped the progress of other areas such as cheminformatics [58], biology [41], quantitative finance [235], marketing [161], fraud analysis [201] and malware detection [49].

The first section of this chapter discusses the main challenges that arise when processing graph structured data. The next section gives an overview of the key research topics in graph mining theory and describes the main characteristics of software tools utilized for graph analytics. The last section outlines the structure and contributions of this dissertation.

1.1 Challenges for graph mining

In this section we give an overview of the main technical, conceptual and operational challenges that novel graph mining techniques and algorithms have to tackle. This thesis proposes methods to tackle these challenges.

Web-scale data. Graph data can represent small molecules with a few vertices and edges but also large online social networks which have billions of nodes and links. Storing data at this scale itself presents challenges and designing graph mining techniques which can efficiently extract knowledge from datasets at web scale is even more challenging. In practical terms this means that graph mining algorithms have to be efficient – their time and space complexity should not be worse than linear in the number of vertices and edges when it comes to big data. The algorithms proposed in this thesis solve this challenge, as our methods are all time and space efficient.

Extreme sparsity. Real world graph structured data exhibits a high level of sparsity in a number of aspects. Let us consider the example of an online social network. First, the users of a social network only have connections with a limited number of other users. Second, the users interact frequently with a limited number of their neighbours. Finally, these interactions are temporally concentrated and the users behave in a bursty manner in which high intensity periods of frequent interactions are followed by sparse cool off cycles. Graph mining techniques which work well in practice must consider the challenges caused by this extreme level of data sparsity.

Multimodal data. Graph data is commonly used to represent social networks which are rich in multimodal metadata such as sound, images and text. Social media websites such as Facebook and Twitter collate a vast amount of non-structured data about the users (nodes) and different types of edges. Incorporating information from these data sources is crucial for practical applications of machine learning, which means that modern graph mining techniques have to be able to integrate these data sources efficiently. Another aspect of multimodality is the creation of graphs based on non-graph structured data. Novel graph mining techniques must be able to create similarity graphs from large scale multimodal data in a principled way. This raises numerous non-trivial computational and conceptual challenges. For instance: How can we integrate multiple data sources into a single similarity metric to create edge weights?

Autocorrelation. Attributed graphs created from real world data commonly exhibit the so-called *birds of a feather* phenomenon with respect to generic node attributes and structural features of vertices. In statistical terms this phenomenon is spatial autocorrelation with respect to the variable at question. This spatial autocorrelation can be incorporated into a supervised model and used to predict the attribute which exhibits autocorrelation. At the same time modeling a spatially autocorrelated target variable inconsiderably can result in residual autocorrelation and a misspecified statistical model.

Inductive learning. Extracting node features based on proximity contextualizes the vertices in relation to a landmark. This feature engineering approach has practical appeal as it is intuitive, explainable, scalable and allows for the extraction of expressive and highly predictive node features when generic vertex attributes are not available. However, it falls short if the graph of interest consists of multiple disconnected components. Transferring knowledge from one graph to another in an inductive manner is also impossible when contextualization is grounded on a landmark like proximity. Modern graph mining techniques try to solve this challenge by defining universal contexts such as structural properties or generic vertex attributes.

Attribution of information. It is fairly common that the topological properties of vertices such as centrality and embeddedness are correlated with the relative location of nodes [9]. For example, in a chain graph with an even number of nodes the vertex in the middle has the highest centrality. Moreover structural and proximal characteristics are potentially correlated with generic vertex attributes such as the age of people in a social network [149]. Hence, a predictive model which predicts a node feature and uses correlated proximal, structural or generic features might misattribute the role of features in an endogeneous manner when correlated features are left out. Disentangling the individual role of features and attribution is therefore a primary challenge for predictive models that operate on graph structured data.

Multiplexity, heterogeneity, and dynamics. Traditional models of graph mining assume that vertices and edges in a graph are untyped and static. However, in real world settings the assumption about a *homogeneous graph* does not hold. In a social network users can interact in a number of ways which results in a multiplex graph. Moreover, interactions are temporal which makes links dynamic and nodes themselves can appear and disappear. Users can also interact with non-users such as pages and posts which can be seen as a vertex with a different type, which alludes that the underlying graph in fact is heterogeneous. Graph mining techniques must adapt to the challenges that come with the relaxed assumptions about a static homogeneous graph.

1.2 Key topics in graph mining

We will now discuss the most important recent scientific developments in the field of graph mining. Our goal is to position the algorithmic and applied contributions of the thesis with relation to current state-of-the-art graph mining research. We have a focus on missing representation capabilities of current graph mining algorithms which would be important contributions to the field.

Node embedding

A central issue in graph mining is the extraction of expressive node features [72, 86] which can be used by downstream machine learning algorithms to predict missing properties of nodes. Generic attributes of nodes might not be available or the node feature space could be high dimensional which would render the use of a complex downstream model infeasible. Node embedding techniques solve this challenge by mapping nodes into a low dimensional space without any manual feature engineering [9]. The coordinates of nodes from the embedding space are then used as input features to the downstream model.

Proximity preserving node embedding

Proximity preserving node embedding techniques [8, 199, 200] learn the mapping from feature space to the embedding space based on using other nodes in the graph as landmark features. Precisely, the relative proximities of neighbouring vertices in the graph are utilized as features to contextualize the global location of nodes. Defining proximity based features helps when the target attribute of interest exhibits a strong spatial autocorrelation. In simple terms this means that nodes that are connected are likely to have a similar value of the outcome variable. For example users with a large number of common friends in a social network are likely to have a similar age [149].

A proximity preserving node embedding is learned by decomposing a node-node proximity matrix which might describe adjacency relationships [8], neighbourhood overlap [139] or the modularity of the underlying community structure [199, 200]. This poses a modeling question that has important practical implications: how do we define a matrix decomposition problem that can be solved efficiently? This question is crucial as we might want to factorize dense matrices that describe higher order proximity, not just simple first order relations which can be described by a sparse adjacency matrix. The answer is the use of *implicit matrix factorization* [129, 130, 154] a method that exploits negative sampling, which is a form of noise contrastive estimation [67], to learn the node embedding in a space and time efficient manner.

The flagship implicit factorization based proximity preserving node embedding technique *DeepWalk* [147, 154] decomposes a target matrix which is a weighted sum of normalized adjacency matrix powers up to a given order. The approximate factorization technique described in *DeepWalk* can incorporate information from arbitrary order neighbourhoods, yet learn the node embedding in linear time. This inspired novel node embedding techniques such as *Diff2Vec*, *Node2Vec*, and others [24, 66, 150, 170] which also use a similar implicit decomposition technique.

Multi-scaling and the order of proximity. The node embedding technique proposed in *DeepWalk* has multiple shortcomings. First, proximity information coming from direct neighbours and higher order relations cannot be disentangled. It is reasonable to create node features which describe the relative location of a node in the graph at multiple scales [22, 148, 198]. The simplest way is the explicit factorization of normalized adjacency matrix powers one by one [22], while a more nuanced approach is doing the same but with an implicit technique. As we mentioned *DeepWalk* sums the normalized adjacency matrix powers and the weights used in this sum are uniform – information coming from different proximities receives the same weight. This assumption can be relaxed [4, 167] and a distribution can be learned over the various orders of proximities to aggregate information in a less restricted way.

Explicit clustering in the embedding space. Proximity preserving node embedding algorithms inherently assume that the nodes in the graph are clustered. In fact a number of community detection procedures are based on matrix factorization [99, 193, 232] – community memberships of vertices are associated with factor vector dimensions. Hence, the implicit assumption about clustering can be made explicit by a custom tailored loss function which forces nodes to group in the embedding space [24, 218, 238]. Our contribution *Graph Embedding with Self-Clustering* described in Chapter 2 goes one step further and allows for community-aware node embeddings which have a different number of clusters and embedding dimensions [166].

Structural node embedding

Using the proximity of nodes to landmark vertices in the graph was one way to describe nodes. Another approach could be describing the structural roles [9] that nodes play in the network. For example, vertices might serve as information brokers, bridges or exist on the fringe of the network. Distilling features which describe the structural role of vertices in the network is crucial when it is reasonable to assume that the outcome variable can be explained by those. Such targets could be the number of views on a webpage [165], users who churn from a platform [173] or the role of computer servers on the physical internet [76].

An intuitive way to describe the structural role of vertices is the computation of descriptive statistics such as degree, clustering coefficient, embeddedness and centrality. However, using these raw node level metrics is not sufficient as the structural role of a node is characterized by context. In order to appropriately model this structural role identification techniques such as *ReFex* [76] try to characterize the distribution of these statistics in the neighbourhood of vertices by recursively aggregating them. Indiscriminate generation of descriptive statistics is costly and the aggregation results in high dimensional feature spaces and features which are strongly correlated, which leads to the need for structural node embedding algorithms. Structural node embedding algorithms perform dimensionality reduction on a node – structural feature matrix [11, 66, 75]. The result is an embedding where nodes which share similar structural features are located close in the embedding space. However, this closeness does not necessarily coincide with spatial proximity in the graph itself [41, 171]. It is worth emphasizing that scalable techniques utilize implicit matrix factorization to learn the node embedding.

Attributed node embedding

Proximity preserving and structural role-based node embedding techniques [11, 75, 147, 148] perform dimensionality reduction on vertex features extracted purely using the graph topology. Attributed node embedding algorithms [81, 116, 165, 171, 227, 231, 234, 236] use supplementary vertex features when the mapping to the embedding space is learned. The rationale behind exploiting generic vertex attributes is simple: nodes that share a large number of connections and properties might be similar with respect to a potential target feature.

The majority of the existing algorithms formulates the attributed node embedding problem as a joint matrix factorization problem with first order information [81, 116, 234, 236]. This is mainly due to computational reasons, since procedures which use higher order proximity information have limited scalability and practical usability [227, 231]. Our works *Multi-Scale Attributed Node Embedding* in Chapter 3 and *Characteristic Functions on Graphs* in Chapter 4 make multiple important contributions to the theory and application of attributed node embeddings: (i) we are able to include information from higher order proximity efficiently (ii) we contextualize nodes based purely on generic vertex features.

Graph kernels

Node embedding techniques look at the graph from a micro-level view to make nodes comparable based on their location, structural role and attributes. For example, these methods are useful

when we want to solve predictive problems on a social network which require a fine-grained perspective such as the prediction of user age, churn and uplift [148, 149]. At the same time making macro-level comparisons of graphs is also a reasonable goal for graph mining. Graph kernels are similarity functions designed to measure the structural similarity of graphs using a set of handcrafted topological features [97, 192]. Building on graph kernels, supervised kernelized machine learning models can be defined to predict properties at the graph level such as the solubility of molecules [182].

The design of graph kernels utilizes a pairwise comparison of vertices [74] based on structural features to create a single scalar similarity measure. The pre-defined topological properties used for measuring similarity can be frequent subtrees [31, 182], cyclic patterns [78, 95], paths [157, 197] or truncated attributed random walks [53, 89, 191] among others. A major direction of research focuses on restricting the number of pairwise comparisons [96, 177] using optimal assignments. In these graph kernels only the structural similarity of matched nodes contributes to the graph level similarity metric.

Graph kernels were a crucial step towards whole graph embedding [27, 134] techniques and graph neural networks, but they have considerable computational shortcomings. First, the extraction of graph level topological features such as cycles and shortest paths in itself is computationally intensive [53, 78]. Second, calculating the kernel function is costly when an optimal assignment problem has to be solved for all pairs of graphs. Finally, given a dataset of graphs one has to calculate all of the pairwise similarities between the graphs which can be prohibitively space and time expensive.

Graph level embedding and statistical descriptors

Graph level embedding and sketching techniques map graphs to a low dimensional metric space in such way that those which have similar structural properties are close to each other [134, 171, 202, 226]. The rational behind this idea is to design a two step framework - first learn a graph level embedding to extract structural features then apply standard machine learning tools to the embedding features to solve arbitrary graph level predictive tasks. Potential applications include molecular property estimation [134], malware identification [49] and fraud detection [201].

Graph level embedding

A graph level embedding is learned by implicitly factorizing a graph - structural feature matrix. The structural features used to learn the graph level embedding can vary from computationally cheap subtrees [134, 182] to hard to compute cyclic patterns [78, 225, 226]. A considerable advantage of the most popular subtree based graph level embedding techniques [27, 134] is the linear time and space complexity and easy-to-use publicly available implementations [168]. In practice these techniques can easily learn representations for millions of graphs with thousands of nodes [49].

Statistical descriptors

Graph level statistical descriptors extract structural attributes of the graph which are not interpretable compared to tree features and cyclic patterns. A class of these methods distills features from the eigendecomposition of graph Laplacian [34, 213] such as the eigenspectrum or the heat trace [202, 204]. These techniques extract highly predictive features but at the same characterized by unfavorable runtime complexity. A separate class of statistical descriptor methods considers scattering [51] and characteristic function transforms [171] to create expressive sets of graph level features. Compared to eigendecomposition based methods these algorithms have linear time requirement, excellent scalability and predictive performance [49].

Graph neural networks

Most of the node embedding techniques which we described [147, 148] are *transductive* learning algorithms [71, 92] which is a very limiting shortcoming. In practice it means that we cannot create representations of new nodes in a dynamic graph setting when those are added to the vertex set incrementally. This limitation can be also present in a static setting – because nodes from two unconnected graph components do not necessarily have a shared proximal or structural context [9]. Graph neural networks create node embeddings in an inductive manner by contextualizing the nodes with the distribution of exogenous attributes in their neighbourhood. These vertex attributes have to be universally present for all of the graphs used for the learning process and inference.

Inductive node representation learning with graph neural networks

Graph neural networks which create node level representations perform *neural message passing* with the learned node representations [58]. This process consists two steps: (*i*) encoding the vertex features as hidden representations for every node by a neural network; (*ii*) followed by a neighbourhood based propagation of these hidden representations. These hidden node representations can be learnt in a supervised [2, 92] or unsupervised manner [71, 211]. Model architectures are mainly differentiated by how the feature encoding is parametrized and what message aggregation function is used for the message propagation.

The majority of graph neural network architectures encodes the vertex features with a *single hidden layer* feedforward neural network [2, 3, 28, 71, 92]. Using a deep neural network for message compression is also a frequent architecture design choice [93, 211] while generalized spatial regression like linear models are rarely used for learning node representations [222]. The weights used in the message passing phase are a major driving force behind predictive performance. Common non-adaptive message passing aggregation weights are defined by the graph Laplacian matrix [92], graph wavelet functions [224], powers of the normalized adjacency matrix [5, 71, 72], approximate personalized PageRank scores [18, 105] and the higher-order diffusion matrix [94, 115]. Adaptive message passing schemes are defined by incorporating node and edge features which are fed to a neural network layer which parametrizes the message-passing edge weights [167, 210].

Pooling node representations

Node representations generated by graph neural network layers can be aggregated to create graph level representations. These graph level representations are obtained by using a *pooling function* which aggregates the node representations [72]. By utilizing these pooled graph level features one can solve analytical problems which involve inference about the whole graph such as molecular toxicity prediction. One common approach to perform node feature pooling is to select the strongest node level activations [52, 247]. Another possibility is to define the weights used for the node representation aggregation with an attention mechanism [102, 240]. This way those node representations which help the most with the graph level downstream problem will get the highest aggregation weight during the pooling.

TABLE 1.1: Multi-aspect summary of open-source graph mining libraries. We provide an exhaustive coverage of analytics and machine learning libraries that operate on graph structured data. Libraries are compared based on year of release, support for dynamic graphs, functionalities provided and programming language.

Reference	Year	Time		Analytics						Machine Learning						Languages					
		Static	Temporal	Shortest Path	Centrality	Transitivity	k-cores	Cascades	Sampling	Community Detection	Node Embedding	Graph Embedding	Link Prediction	Graph Neural Networks	Graph Kernels	Python	R	Julia	C++	Java	Scala
IGraph [30]	2006	•		•	•	•	•			•						•	•		•		
NetworkX [68]	2008	•		•	•	•	•			•						•					
SNAP [109]	2014	•		•	•	•	•		•	•	•		•			•			•		
GraphTool [146]	2014	•		•	•	•	•	•		•						•			•		
NetworKit [186]	2016	•	•	•	•	•	•		•	•			•			•			•		
DyNetX [162]	2016		•	•												•					
LightGraphs [19]	2017	•		•	•	•												•			
ND-Lib [164]	2018		•					•								•					
Little Ball of Fur [169]	2020		•						•							•					
Lynx Kite [32]	2020	•		•	•	•	•		•	•	•		•	•		•					•
DLG [217]	2019	•									•			•		•					
Euler [233]	2019	•									•			•		•					
Google AI GCNN [26]	2020	•									•			•		•					
Spektral [64]	2020	•									•			•		•					
Torch Geometric [47]	2019	•									•			•		•					
Torch Geometric Temporal [174]	2020		•											•		•					
Karate Club [168]	2020	•								•	•	•				•					
OpenNE [205]	2018	•									•			•		•					
OpenKE [73]	2018	•									•			•		•					
EvalNE [124]	2019	•									•		•			•					
GraphKit-Learn [83]	2020	•													•	•					
GraKel [184]	2020	•													•	•					
GraphKernels [192]	2017	•													•	•	•		•		
CD-Lib [163]	2019	•								•						•					

Graph mining software

Open-source graph mining software packages can be categorized into two main groups: analytics frameworks and machine learning libraries. We summarized the functionalities of publicly available open-source graph mining software in Table 1.1 based on multiple aspects such as year of release, programming language, available descriptive statistics and machine learning tools.

Analytics libraries [30, 68, 109, 146, 186] provide simple tools such as clustering coefficient and centrality calculation that allow the descriptive analysis of large graphs. These tools are relatively older, only provide basic machine learning capabilities (community detection and link prediction) and frequently have a high performance C++ backend.

Libraries designed for machine learning on graphs mostly provide a breadth of algorithms in niche domains. One large group focuses on supervised and unsupervised node level representation learning [47, 64, 73, 168, 174, 205]. Another large group focuses on the calculation of graph kernel functions [83, 184, 192] for kernelized machine learning. Machine learning libraries which have the same functionalities are mostly differentiated by the linear algebra and automatic differentiation backend [1, 144, 214, 215] that serves the modeling functionalities. These frameworks are relatively new, have a limited scope and the support for dynamic problems is nearly non-existent [174].

1.3 Contributions and structure of the thesis

This section overviews the main contributions of the thesis which has two major interrelated parts. In Part I we describe novel unsupervised and supervised machine learning techniques which operate on graph structured data.

- Chapter 2, *Graph Embedding with Self-Clustering* proposes *GEMSEC* an implicit factorization machine which learns a proximity-preserving parametric node embedding and graph clustering jointly. We achieve this by implicitly decomposing the empirically approximated pointwise mutual information matrix calculated from co-occurrence statistics of first- and second-order truncated random walks with a k-means like clustering constraint. Our formulation results in a linear time and space algorithm which incorporates higher order proximity information to create latent space clusters and embeddings. Elaborate experiments on social network data from Facebook and Deezer validate that *GEMSEC* extracts high quality vertex features without supervision for node classification and community detection.

- Chapter 3, *Multi-Scale Attributed Node Embedding* introduces *MUSAE* the first attributed node embedding algorithm that can learn multi-scale vertex representation contextualized by vertex attributes in an unsupervised way. We design an implicit factorization machine which decomposes the empirically approximated node - vertex attribute pointwise mutual information matrix calculated from attributed random walks. We theoretically prove that with the random walks *MUSAE* asymptotically factorizes adjacency matrix power - vertex attribute matrix products in linear time and space. Our empirical evaluation which focuses on node classification, node level regression, link prediction and transfer learning tasks demonstrates that *MUSAE* is competitive even with state of the art graph neural networks.

- Chapter 4, *Characteristic Functions on Graphs* generalizes characteristic functions (the Fourier transform of probability distributions) to describe generic vertex feature distributions in the neighbourhood of nodes. We argue that characteristic functions defined on neighbourhoods serve as succinct structural embeddings of nodes with the appropriate proximity

based parametrization. A truncated random walk based conceptualization of characteristic functions is introduced along with *FEATHER*, an efficient algorithm which computes these functions in linear time for all of the vertices in a graph. We prove the robustness and permutation invariance of characteristic functions and we introduce the parametric statistical models *FEATHER-L* and *FEATHER-N* which build on this new class of characteristic functions. Comprehensive experiments validate the predictive performance and efficacy of the *FEATHER* model variants on graph classification, node classification and transfer learning problems.

- Chapter 5, *Pathfinder Discovery Networks for Neural Message Passing* describes the *PDN* a graph neural network layer which can aggregate the edge weights from the layers of a multiplex graph to define an optimal message passing graph tailored for a downstream task. We discuss how existing graph neural network architectures can be integrated with our parametric edge weight aggregation scheme to define this task specific propagation graph. The flexible design of the *PDN* allows us to formulate edge convolutions and multi-scale graph neural networks as a *PDN*. Theoretical results show that a *PDN* would not suffer from oversmoothing and that our model can learn to separate inter- and intra-class edges from expressive edge features. A series of experiments establish that the basic *PDN* model and its variants are competitive with the predictive performance of node embeddings and graph neural networks. The theoretical result about the separation of inter- and intra-class edges is validated with synthetically generated data.

- Chapter 6, *The Shapley Value of Classifiers in Ensemble Games* defines the concept of *ensemble games*, a special type of weighted voting games, where machine learning models in an ensemble collaborate to classify a data point correctly. We derive a dual class of games to characterize incorrect classification decisions. An approximate Shapley value based solution of these games is introduced to quantify the role of individual classifiers in the decision of the ensemble. Using this solution of the games we propose *Troupe*, an algorithm to estimate the average Shapley value of models based on a labeled dataset. We argue that these scores describe the power of classifiers in the ensemble and the entropy of the Shapley values is a measure of model heterogeneity. Our proofs establish the time and space complexity required to run *Troupe* and the sample size needed to obtain precise estimates of the average Shapley value. Our evaluation of *Troupe* on graph classification tasks verifies that we are able to get high quality estimates of the Shapley value and also that the Shapley values of classifiers in ensemble games are suitable for identifying high quality models in the ensemble.

In Part II, we discuss the design and use cases of graph mining software which were developed as part of this thesis.

- Chapter 7, *Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs* discusses the design of *Karate Club*, an open-source Python library which provides community detection, node embedding and whole graph embedding functionalities. This framework was created with a user friendly interface, includes a limited

number of public class methods, has encapsulated default hyperparameters, standardized dataset ingestion and output generation. The main features of the library are showcased by practical graph mining code snippets. We demonstrate the predictive performance of the implemented methods on various practical graph mining tasks such as graph and node classification.

- Chapter 8, *Little Ball of Fur: A Python Library for Graph Sampling* describes the design of *Little Ball of Fur*, a publicly available Python framework which provides graph sampling algorithms. *Little Ball of Fur* is the first open-source library which provides techniques which can extract representative subgraphs from a larger graph using node, edge and exploration-based sampling algorithms. This framework is built with an API oriented design which entails simple sampler constructors, a single public sample method, default hyperparameters and standardized graph imputation. Capabilities of the framework are highlighted with illustrative examples of code. We demonstrate with experiments that the samples extracted with the algorithms available in *Little Ball of Fur* are representative subgraphs and that our framework can be used to accelerate downstream graph mining techniques.

The thesis ends with Chapter 9 which summarizes the most important theoretical and empirical contributions of the thesis and points out directions for future research.

Part I

Modern Techniques for Graph Mining

Chapter 2

Graph Embedding with Self-Clustering

Modern graph embedding procedures can efficiently process graphs with millions of nodes. In this chapter, we propose *GEMSEC* – a graph embedding algorithm which learns a clustering of the nodes simultaneously with computing their embedding. *GEMSEC* is a general extension of earlier work in the domain of sequence-based graph embedding. *GEMSEC* places nodes in an abstract feature space where the vertex features minimize the negative log-likelihood of preserving sampled vertex neighborhoods, and it incorporates known social network properties through a machine learning regularization. We present two new social network datasets and show that by simultaneously considering the embedding and clustering problems with respect to social network properties, *GEMSEC* extracts high-quality clusters competitive with or superior to other community detection algorithms. In experiments, the method is found to be computationally efficient and robust to the choice of hyperparameters.

2.1 Introduction

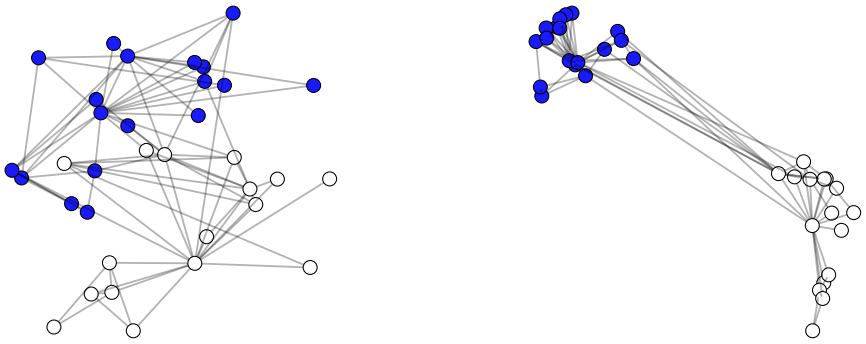
Community detection is one of the most important problems in network analysis due to its wide applications ranging from the analysis of collaboration networks to image segmentation, the study of protein-protein interaction networks in biology, and many others [14, 57, 141, 173, 175, 209, 219]. Communities are usually defined as groups of nodes that are connected to each other more densely than to the rest of the network. Classical approaches to community detection depend on properties such as graph metrics, spectral properties and density of shortest paths [108]. Random walks and randomized label propagation [65, 143] also have been investigated.

Embedding the nodes in a low dimensional Euclidean space enables us to apply standard machine learning techniques. This space is sometimes called the *feature space* – implying that it represents abstract structural features of the network. Embeddings have been used for machine learning tasks such as labeling nodes, regression, link prediction, and graph visualization, see [63] for a survey. Graph embedding processes usually aim to preserve certain predefined differences between nodes encoded in their embedding distances. For social network embedding, a natural priority is to preserve community membership and enable community detection.

Recently, sequence-based methods have been developed as a way to convert complex, non-linear network structures into formats more compatible with vector spaces. These methods sample sequences of nodes from the graph using a randomized mechanism (e.g. random walks), with the idea that nodes that are "close" in the graph connectivity will also frequently appear close in a sampling of random walks. The methods then proceed to use this random-walk-proximity

information as a basis to embed nodes such that socially close nodes are placed nearby. In this category, *Deepwalk* [147] and *Node2Vec* [66] are two popular methods.

While these methods preserve the proximity of nodes in the graph sense, they do not have an explicit preference for preserving social communities. Thus, in this chapter, we develop a machine learning approach that considers clustering when embedding the network and includes a parameter to control the closeness of nodes in the same community. Figure 2.1(a) shows the embedding obtained by the standard *Deepwalk* method, where communities are coherent, but not clearly separated in the embedding. The method described in this chapter, called *GEMSEC*, is able to produce clusters that are tightly embedded and separated from each other (Fig. 2.1(b)).



(A) DeepWalk (B) GEMSEC

FIGURE 2.1: Zachary's Karate club graph [242]. White nodes: instructor's group; blue nodes: president's group. GEMSEC produces embedding with more tightly clustered communities.

Our contributions

GEMSEC is an algorithm that considers the two problems of embedding and community detection simultaneously, and as a result, the two solutions of embedding and clustering can inform and improve each other. Through iterations, the embedding converges toward one where nodes are placed close to their neighbors in the network, while at the same time clusters in the embedding space are well separated.

The algorithm is based on the paradigm of sequence-based node embedding procedures that create d dimensional feature representations of nodes in an abstract feature space. Sequence-based node embeddings embed pairs of nodes close to each other if they occur frequently within a small window of each other in a random walk. This problem can be formulated as minimizing the negative log-likelihood of observed neighborhood samples (Sec. 3.3) and is called the skip-gram optimization [129]. We extend this objective function to include a clustering cost. The formal description is presented in Subsection 2.3. The resulting optimization problem is solved with a variant of mini-batch gradient descent [91].

The detailed algorithm is presented in Subsection 2.3. By enforcing clustering on the embedding, *GEMSEC* reveals the natural community structure (e.g. Figure 2.1). Our approach improves over existing methods of simultaneous embedding and clustering [24, 218, 238] and shows that community sensitivity can be directly incorporated into the skip-gram style optimization to obtain greater accuracy and efficiency.

In social networks, nodes in the same community tend to have similar groups of friends, which is expressed as high neighborhood overlap. This fact can be leveraged to produce clusters that are better aligned with the underlying communities. We achieve this effect using a regularization procedure – a smoothness regularization added to the basic optimization achieves more coherent community detection. The effect can be seen in Figure 2.3, where a somewhat uncertain community affiliation suggested by the randomized sampling is sharpened by the smoothness regularization. This technique is described in Subsection 3.3.

In experimental evaluation we demonstrate that *GEMSEC* outperforms – in clustering quality – the state of the art neighborhood based [66, 147], multi-scale [148, 198] and community aware embedding methods [24, 218, 238]. We present new social datasets from the streaming service Deezer and show that the clustering can improve music recommendations. The clustering performance of *GEMSEC* is found to be robust to hyperparameter changes, and the runtime complexity of our method is linear in the size of the graphs.

To summarize, the main contributions of our work are:

1. *GEMSEC*: a sequence sampling-based learning model which learns an embedding of the nodes at the same time as it learns a clustering of the nodes.
2. Clustering in *GEMSEC* can be aligned to network neighborhoods by a smoothness regularization added to the optimization. This enhances the algorithm's sensitivity to natural communities.
3. Two new large social network datasets are introduced – from Facebook and Deezer data.
4. Experimental results show that the embedding process runs linearly in the input size. It generally performs well in quality of embedding and in particular outperforms existing methods on cluster quality measured by modularity and subsequent recommendation tasks.

We start with reviewing related work in the area and relation to our approach in the next section. A high-performance Tensorflow reference implementation of *GEMSEC* and the datasets that we collected can be accessed online `https://github.com/benedekrozemberczki/GEMSEC`.

2.2 Related work

There is a long line of research in metric embedding – for example, embedding discrete metrics into trees [45] and into vector spaces [126]. Optimization-based representation of networks has been used for routing and navigation in domains such as sensor networks and robotics [80, 241]. Representations in hyperbolic spaces have emerged as a technique to preserve richer network structures [35, 176, 243].

A general overview

Recent advances in node embedding procedures have made it possible to learn vector features for large real-world graphs [66, 147, 198]. Features extracted with these *sequence-based node embedding*

procedures can be used for predicting social network users' missing age [63], the category of scientific papers in citation networks [148] and the function of proteins in protein-protein interaction networks [66]. Besides supervised learning tasks on nodes the extracted features can be used for graph visualization [63], link prediction [66] and community detection [24].

Sequence based embedding commonly considers variations in the sampling strategy that is used to obtain vertex sequences – truncated random walks being the simplest strategy [147]. More involved methods include second-order random walks [66], skips in random walks [148] and diffusion graphs [170]. It is worth noting that these models implicitly approximate matrix factorizations for different matrices that are expensive to factorize explicitly [154].

Our work extends the literature of node embedding algorithms which are community aware. Earlier works in this category did not directly extend the skip-gram embedding framework. *M-NMF* [218] applies computationally expensive non-negative matrix factorization with a modularity constraint term. The procedure *DANMF* [238] uses hierarchical non-negative matrix factorization to create community-aware node embeddings. *ComE* [24] is a more scalable approach, but it assumes that in the embedding space the communities fit a gaussian structure, and aims to model them by a mixture of Gaussians. In comparison to these methods, *GEMSEC* provides greater control over community sensitivity of the embedding process, it is independent of the specific neighborhood sampling methods and is computationally efficient.

A desiderata based comparison

A proximity preserving node embedding procedure which satisfies community awareness and preserves the macro-level neighbourhood structure must possess certain amicable properties. We summarized these important properties for various well known proximity preserving node embedding algorithms in Table 2.1 with the respective space and time complexities.

- **Non-linearity:** The reconstructed proximity matrix is not a linear function of the node embedding matrices. This allows for learning more expressive representations.
- **Multi-scaleness:** A multi-scale node embedding algorithm is able to encode neighbourhood information from different orders of proximities [22, 148, 198] and the learned node representations do not mix information from separate scales.
- **Implicit:** The proximity matrix of interest is decomposed implicitly when the node embeddings are learned. Scalable modern node embedding methods [147, 148, 198] do not use an exact proximity matrix calculation and decomposition. An exact decomposition approach results in worse than linear space and time complexity in the number of nodes.
- **Proximity agnostic:** A proximity agnostic node embedding algorithm is capable of encoding different proximity matrices, not just a single hand crafted one such as the pointwise mutual information matrix decomposed by *DeepWalk* [147].
- **Cluster awareness:** Learning the node embedding involves a cost function that is being optimized. A cluster (community) aware node embedding technique [24, 218, 238] has a term in the cost function which ensure the cluster awareness.

- **Decoupled cluster dimensions:** The number of clusters should not be the same as the number of node embedding dimensions. Such constraints seriously limit the expressiveness of the node embedding features [99, 193, 218, 232].
- **Space and time efficiency:** A well designed node embedding algorithm has linear space and time complexity in the number of nodes. Anything that has super-linear time or space complexity (e.g. *GraRep* [22]) in the number of vertices is not an efficient procedure for learning a node embedding.

Looking at the summary results presented in Table 2.1 it is evident that our proposed Algorithm satisfies all the desired properties and has efficient time and space complexities.

TABLE 2.1: A summary of existing proximity preserving and community aware node embedding techniques with respect to having (✔) and missing (✘) desired properties. The time an space complexity are reported as a function of vertex count $|V|$, embedding dimensions d, window size w, number of negative samples k, and number of clusters centers $|C|$.

	Non-linear	Multi Scale	Implicit	Proximity Agnostic	Cluster Awareness	Decoupled Cluster Dimensions	Space Complexity	Time Complexity
DeepWalk [147]	✔	✘	✔	✘	✘	✘	$\mathcal{O}(\|V\| \, d)$	$\mathcal{O}(\|V\| \, d \, k \, w)$
LINE$_2$ [198]	✔	✔	✔	✘	✘	✘	$\mathcal{O}(\|V\| \, d)$	$\mathcal{O}(\|V\| \, d \, k)$
Node2Vec [66]	✔	✘	✔	✔	✘	✘	$\mathcal{O}(\|V\|^3)$	$\mathcal{O}(\|V\| \, d \, k \, w)$
Walklets [148]	✔	✔	✔	✘	✘	✘	$\mathcal{O}(\|V\| \, d \, w)$	$\mathcal{O}(\|V\| \, d \, k \, w)$
GraRep [22]	✔	✔	✘	✘	✘	✘	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^3 \, d \, w)$
NetMF [154]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^3 \, d)$
BoostNE [85]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^3 \, d)$
Diff2Vec [170]	✔	✘	✔	✘	✘	✘	$\mathcal{O}(\|V\| \, d)$	$\mathcal{O}(\|V\| \, d \, k \, w)$
HOPE [139]	✘	✔	✘	✔	✘	✘	$\mathcal{O}(\|V\|^2 \, d)$	$\mathcal{O}(\|V\|^2 \, d)$
VERSE [203]	✔	✘	✔	✔	✘	✘	$\mathcal{O}(\|V\| \, d)$	$\mathcal{O}(\|V\|^2 \, d \, k)$
BigClam [232]	✔	✘	✔	✘	✘	✘	$\mathcal{O}(\|V\| \, d)$	$\mathcal{O}(\|V\| \, d)$
NNSED [193]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^2 \, d)$
SymmNMF [99]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^2 \, d)$
M-NMF [218]	✘	✘	✘	✘	✔	✘	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^2 \, d)$
ComE [24]	✔	✘	✔	✘	✔	✘	$\mathcal{O}(\|V\| \, d)$	$\mathcal{O}(\|V\| \, d^2 \, k \, w)$
DANMF [238]	✔	✘	✘	✘	✔	✘	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\|^2 \, d)$
GEMSEC (ours)	✔	✔	✔	✔	✔	✔	$\mathcal{O}(\|V\| \, d)$	$\mathcal{O}(\|V\| \, d \, \|C\| \, k \, w)$

2.3 Graph embedding with self-clustering

For a graph $G = (V, E)$, a node embedding is a mapping $f : V \to \mathbb{R}^d$ where d is the dimensionality of the embedding space. For each node $v \in V$ we create a d dimensional representation. Alternatively, the embedding is a $|V| \times d$ real-valued matrix. In sequence-based embedding, sequences of neighboring nodes are sampled from the graph. Within a sequence, a node v occurs in the *context* of a window ω within the sequence. Given a sample S of sequences, we refer to the collection of windows containing v as $N_S(v)$. Earlier works have proposed random walks, second-order random walks or branching processes to obtain $N_S(v)$. In our experiments, we used unweighted first and second-order random walks for node sampling [66, 147].

Our goal is to minimize the negative log-likelihood of observing neighborhoods of source nodes conditional on feature vectors that describe the position of nodes in the embedding space.

Formally, the optimization objective is:

$$\min_{f} \quad \sum_{v \in V} -\log P(N_S(v)|f(v)) \tag{2.1}$$

for a suitable probability function $P(\cdot|\cdot)$. To define this P, we consider two standard properties (see [66]) expected of the embedding f in relation to N_S. First, it should be possible to factorize $P(N_S(v)|f(v))$ in line with *conditional independence* with respect to $f(v)$. Formally:

$$P(N_S(v)|f(v)) = \prod_{n_i \in N_S(v)} P(n_i \in N_S(v) \mid f(v), f(n_i)). \tag{2.2}$$

Second, it should satisfy *symmetry in the feature space*, meaning that source and neighboring nodes have a symmetric effect on each other in the embedding space. A softmax function on the pairwise dot products of node representations with $f(v)$ to get $P(n_i \in N_S(v) \mid f(v), f(n_i))$ express such a property:

$$P(n_i \in N_S(v) \mid f(v), f(n_i)) = \frac{\exp(f(n_i) \cdot f(v))}{\sum_{u \in V} \exp(f(u) \cdot (f(v))}. \tag{2.3}$$

Substituting 2.2 and 2.3 into the optimization function, we get:

$$\min_{f} \sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]. \tag{2.4}$$

The partition function in Equation 2.4 enforces nodes to be embedded in a low volume space around the origin, while the second term forces nodes with similar sampled neighborhoods to be embedded close to each other.

Learning to cluster without overlaps

Next, we extend the optimization to pay attention to the clusters it forms. We include a clustering cost similar to k-means, measuring the distance from nodes to their cluster centers. This augmented optimization problem is described by minimizing a loss function over the embedding f and position of cluster centers μ, that is, $\min_{f,\mu} \mathcal{L}$, where:

$$\mathcal{L} = \underbrace{\sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]}_{\text{Embedding cost}} + \underbrace{\gamma \cdot \sum_{v \in V} \min_{c \in C} \|f(v) - \mu_c\|_2}_{\text{Clustering cost}}. \tag{2.5}$$

In Equation 2.5 we have C the set of cluster centers – the c^{th} cluster mean is denoted by μ_c. Each of these cluster centers is a d-dimensional vector in the embedding space. The idea is to minimize the distance from each node to its nearest cluster center. The weight coefficient of the clustering cost is given by the hyperparameter γ. Evaluating the partition function in the

proposed objective function for all of the source nodes has a $\mathcal{O}(|V|^2)$ runtime complexity. Because of this, we approximate the partition function term with negative sampling which is a form of noise contrastive estimation [67, 129].

(A) Node capture. (B) Empty initialization.

FIGURE 2.2: Potential issues with cluster cost weighting and cluster initialization. Different node colors denote different ground truth community memberships and the computed cluster boundary is denoted by the dashed line. In Subfigure 2.2a a single white node is captured in a cluster with the blue nodes due to clustering weight γ being high. In Subfigure 2.2b an empty cluster is initialized with no nodes in it. It is plausible that the cluster center remains empty throughout the optimization process.

$$\frac{\partial \mathcal{L}}{\partial f(v^*)} = \underbrace{\frac{\sum_{u \in V} \exp(f(v^*) \cdot f(u)) \cdot f(u)}{\sum_{u \in V} \exp(f(v^*) \cdot f(u))}}_{\text{Partition function gradient}} - \underbrace{\sum_{n_i \in N_S(v^*)} f(n_i)}_{\text{Neighbor direction}} + \underbrace{\gamma \cdot \frac{f(v^*) - \mu_c}{\|f(v^*) - \mu_c\|_2}}_{\text{Closest cluster direction}} \tag{2.6}$$

The gradients of the loss function in Equation 2.5 are important in solving the minimization problem. As a result we can obtain the gradients for node representations and cluster centers. Examining in more detail, the gradient of the objective function $\mathcal{L}$ with respect to the representation of node $v^* \in V$ is described by Equation 2.6 if μ_c is the closest cluster center to $f(v^*)$.

The gradient of the partition function pulls the representation of v^* towards the origin. The second term moves the representation of v^* closer to the representations of its neighbors in the embedding space while the third term moves the node closer to the closest cluster center. If we set a high γ value the third term dominates the gradient. This will cause the node to gravitate towards the closest cluster center which might not contain the neighbors of v^*. An example is shown in Figure 2.2a. If the set of nodes that belong to cluster center c is V_c, then the gradient of the objective function with respect to μ_c is described by

$$\frac{\partial \mathcal{L}}{\partial \mu_c} = -\gamma \cdot \sum_{v \in V_c} \frac{f(v) - \mu_c}{\|f(v) - \mu_c\|_2}. \tag{2.7}$$

In Equation 2.7 we see that the gradient moves the cluster center by the sum of coordinates of nodes in the embedding space that belong to cluster c. Second, if a cluster ends up empty it will not be updated as elements of the gradient would be zero. Because of this, cluster centers and embedding weights are initialized with the same uniform distribution. A wrong initialization just like the one with an empty cluster in Subfigure 2.2b can affect clustering performance considerably.

Learning to cluster with overlaps

The design presented in Subsection assumes that a node only belongs to a single cluster center. We cam reformulate the optimization problem defined by Equation 2.5 to accommodate the presence of overlapping clusters. We will have the same setup with a fixed set of cluster center points. The probability of node $v \in V$ belonging to cluster $c \in C$ is parametrized by Equation 2.8. This way we define a probability distribution over the cluster centers.

$$P_{v,c} = \frac{\exp(f(v) \cdot \mu_c)}{\sum\limits_{c \in C} \exp(f(v) \cdot \mu_c)} \tag{2.8}$$

We can add the entropy of these cluster membership distributions to the original embedding loss. By doing so we define a new optimization problem described by Equation 2.9. This way a node can belong to multiple cluster centers at the same time.

$$\mathcal{L} = \underbrace{\sum_{v \in V} \left[\ln \left(\sum_{u \in V} \exp(f(v) \cdot f(u)) \right) - \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right]}_{\text{Embedding cost}} - \underbrace{\gamma \cdot \sum_{v \in V} \sum_{c \in C} P_{v,c} \cdot \log(P_{v,c})}_{\text{Clustering cost}}. \tag{2.9}$$

The gradient of $\mathcal{L}$ with respect to $f(v^*)$ is given by Equation 2.10. Compared to the the non-overlapping case we see that the gradient depends on all cluster center vectors not just the one which is closest in the embedding space. Other parts of the gradient are unaffected by this modification to the model – the gradient of the partition function and the direction of the neighbors is unaffected.

$$\frac{\partial \mathcal{L}}{\partial f(v^*)} = \underbrace{\frac{\sum\limits_{u \in V} \exp(f(v^*) \cdot f(u)) \cdot f(u)}{\sum\limits_{u \in V} \exp(f(v^*) \cdot f(u))}}_{\text{Partition function gradient}} - \underbrace{\sum_{n_i \in N_S(v^*)} f(n_i)}_{\text{Neighbor direction}} + \underbrace{\gamma \cdot \frac{f(v^*) - \mu_c}{\|f(v^*) - \mu_c\|_2}}_{\text{Closest cluster direction}} \tag{2.10}$$

Similarly one can get an analytical solution for the gradient of a $\mathcal{L}$ with respect to μ_c which is given by Equation 2.11. This gradient is also a function of the other cluster center's position in the embedding space.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mu_c} = & \underbrace{\gamma \cdot \sum_{v \in V} \left(1 - \frac{\exp(\mu_c \cdot f(v))}{\sum\limits_{c^* \in C} \exp(\mu_{c^*} \cdot f(v))} \right) \cdot \frac{\exp(\mu_c \cdot f(v))}{\sum\limits_{c^* \in C} \exp(\mu_{c^*} \cdot f(v))} \cdot \left[1 + \log \left(\frac{\exp(\mu_c \cdot f(v))}{\sum\limits_{c^* \in C} \exp(\mu_{c^*} \cdot f(v))} \right) \right] \cdot f(v)}_{\text{Effect of node representations}} \\ & \underbrace{-\gamma \cdot \sum_{v \in V} \sum_{c' \in \{C \setminus \{c\}\}} \frac{\exp(\mu_{c'} \cdot f(v)) \cdot \exp(\mu_c \cdot f(v))}{(\sum\limits_{c^* \in C} \exp(\mu_c^* \cdot f(v)))^2} \cdot \left[1 + \log \left(\frac{\exp(\mu_{c'} \cdot f(v))}{\sum\limits_{c^* \in C} \exp(\mu_{c^*} \cdot f(v))} \right) \right] \cdot f(v)}_{\text{Interaction with other cluster centers.}}. \end{aligned} \tag{2.11}$$

The self-clustering embedding algorithm

We propose an efficient learning method to create *GEMSEC* embeddings which is described with pseudo-code by Algorithm 1. The main idea behind our procedure is the following. To avoid the

Data: $\mathcal{G} = (V, E)$ – Graph to be embedded.
N – Number of sequence samples per node.
l – Length of sequences.
ω – Context size.
d – Number of embedding dimensions.
$|C|$ – Number of clusters.
k – Number of noise samples.
γ_0 – Initial clustering weight coefficient.
α_0, α_F – Initial and final learning rate.
Result: $f(v)$, where $v \in V$
μ_c, where $c \in C$

1 Model $\leftarrow$ **Initialize Model**$(|V|, d, |C|)$
2 $t \leftarrow 0$
3 **for** n in *1:N* **do**
4 $\quad \widehat{V} \leftarrow$ **Shuffle**(V)
5 $\quad$ **for** v in $\widehat{V}$ **do**
6 $\quad\quad t \leftarrow t + 1$
7 $\quad\quad \gamma \leftarrow$ **Update** γ $(\gamma_0, t, w, l, N, |V|)$
8 $\quad\quad \alpha \leftarrow$ **Update** α $(\alpha_0, \alpha_F, t, w, l, N, |V|)$
9 $\quad\quad$ Sequence $\leftarrow$ **Sample Nodes**$(\mathcal{G}, v, l)$
10 $\quad\quad$ Features $\leftarrow$ **Extract Features**$($ Sequence$, \omega)$
11 $\quad\quad$ **Update Weights**$(\text{Model}, \text{Features}, \gamma, \alpha, k)$
12 $\quad$ **end**
13 **end**

Algorithm 1: Graph embedding with self-clustering training procedure.

clustering cost overpowering the graph information (as in Fig. 2.2a), we initialize the system with a low weight $\gamma_0 \in [0, 1]$ for clustering, and through iterations anneal it to 1.

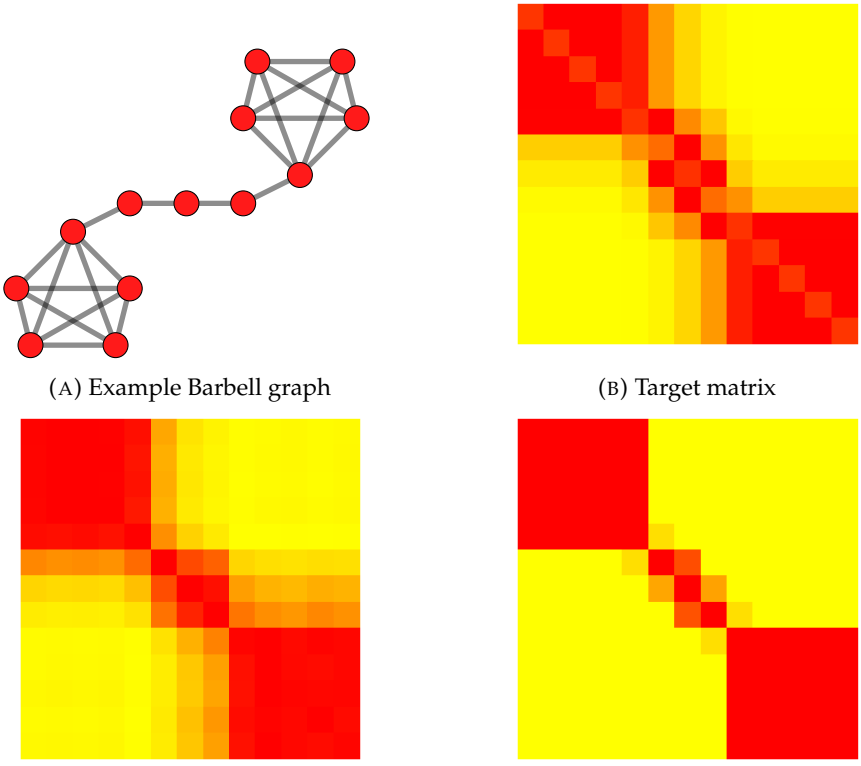
The embedding computation proceeds as follows. The weights in the model are initialized based on the number of vertices, embedding dimensions and clusters. After this, the algorithm makes N sampling repetitions in order to generate vertex sequences from every source node. Before starting a sampling epoch, it shuffles the set of vertices. We set the clustering cost coefficient γ (line 7) according to an exponential annealing rule described by Equation 2.12. The learning rate is set to α (line 8) with a linear annealing rule (Equation 2.13).

$$\gamma = \gamma_0 \cdot \left(10^{\frac{-t \cdot \log_{10} \gamma_0}{w \cdot l \cdot |V| \cdot N}}\right) \tag{2.12}$$

$$\alpha = \alpha_0 - (\alpha_0 - \alpha_F) \cdot \frac{t}{w \cdot l \cdot |V| \cdot N} \tag{2.13}$$

The sampling process reads sequences of length l (line 9) and extracts features using the context window size ω (line 10). The extracted features, gradient, current learning rate and clustering cost coefficient determine the update to model weights by the optimizer (line 11). In the implementation we utilized a variant of stochastic gradient descent – the *Adam* optimizer [91]. We approximate the first cost term with noise contrastive estimation to make the gradient descent

tractable, drawing k noise samples for each positive sample. If the node sampling is done by first-order random walks the runtime complexity of this procedure will be $\mathcal{O}((\omega \cdot k + |C|) \cdot l \cdot d \cdot |V| \cdot N)$ while *DeepWalk* with noise contrastive estimation has a $\mathcal{O}(\omega \cdot k \cdot l \cdot d \cdot |V| \cdot N)$ runtime complexity.



(A) Example Barbell graph (B) Target matrix

(C) DeepWalk Approximation (D) Smooth DeepWalk Approximation

FIGURE 2.3: An example Barbell graph with the corresponding target matrix factorized (window size of 3) by *DeepWalk* [154] and the reconstructed target matrices obtained with standard *DeepWalk* and *Smooth DeepWalk*. Regularized optimization produces more well defined communities. While the standard *DeepWalk* model has less well defined clusters.

Smoothness regularization for coherent community detection

We have seen in Subsection 2.3 that there is a tension between what the clustering objective considers to be clusters and what the real communities are in the underlying social network. We can incorporate additional knowledge of social network communities using a machine learning technique called regularization.

We observe that social networks have natural local properties such as homophily, strong ties between members of a community, etc. Thus, we can incorporate such social network-specific properties in the form of regularization to find more natural embeddings and clusters.

This regularization effect can be achieved by adding a term Λ to the loss function:

$$\Lambda = \lambda \cdot \sum_{(v,u) \in E_S} w_{(v,u)} \cdot \|f(v) - f(u)\|_2 \,, \tag{2.14}$$

where the weight function w determines the *social network cost* of the embedding with respect to properties of the edges traversed in the sampling. We use the neighborhood overlap of an

edge – defined as the fraction of neighbors common to two nodes of the edge relative to the union of the two neighbor sets[1]. In experiments on real data, neighborhood overlap is known to be a strong indicator of the strength of relation between members of a social network [136]. Thus, by treating neighborhood overlap as the weight $w_{v,u}$ of edge (v, u), we can get effective social network clustering, which is confirmed by experiments in the next section. The coefficient λ lets us tune the contribution of the social network cost in the embedding process. In experiments, the regularized version of the algorithms is found to be more robust to changes in hyperparameters.

The effect of the regularization can be understood intuitively through an example. For this exposition, let us consider matrix representations of the social network describing closeness of nodes. In fact, other skip-gram style learning processes like [66, 147] are known to approximate the factorization of a similarity matrix M such as [154]:

$$M_{u,v} = \log\left(\frac{\mathrm{vol}(G)}{\omega}\sum_{r=1}^{\omega}\frac{\sum\limits_{P\in\mathcal{P}^r_{v,u}}\prod\limits_{a\in P\setminus\{v\}}\frac{1}{\deg(a)}}{\deg(v)}\right) - \log(k)$$

where $P^r_{v,u}$ is the set of paths going from v to u with length r. Elements of the target matrix M grow with number of paths of length at most ω between the corresponding nodes. Thus M is intended to represent level of connectivity between nodes in terms of a raw graph feature like number of paths.

The Barbell graph in Figure 2.3a is a typical example with an obvious community structure we can use to analyze the matter. The optimization procedure used by Deepwalk [147] aims to converge to a target matrix $M_{u,v}$ shown in Figure 2.3b. Observe that this matrix has fuzzy edges around the communities of the graph, showing a degree of uncertainty. An actual approximation by running the Deepwalk is shown in Figure 2.3c, which naturally incorporates further uncertainty due to sampling. A much more clear output with sharp communities can be obtained by applying a regularized optimization. This can be seen in Figure 2.3d.

2.4 Experimental evaluation

In this section we evaluate the cluster quality obtained by the *GEMSEC* variants, their scalability, robustness and predictive performance on a downstream supervised task. Results show that *GEMSEC* outperforms or is at par with existing methods in all measures.

The social networks used for evaluation

For the evaluation of *GEMSEC* real-world social network datasets are used which we collected from public APIs specifically for this work. Table 2.2 shows these social networks have a variety of size, density, and level of clustering. We used graphs from two sources:

[1]Neighbor sets $N(a)$ and $N(b)$ of nodes a and b, the neighborhood overlap of (a, b) is defined as the Jaccard similarity $\frac{N(a)\cap N(b)}{N(a)\cup N(b)}$.

TABLE 2.2: Descriptive statistics of social networks and web graphs used for the evaluation of graph embedding with self-clustering.

Source	Dataset	|V|	Density	Transitivity
Facebook	Politicians	5,908	0.0024	0.3011
	Companies	14,113	0.0005	0.1532
	Athletes	13,866	0.0009	0.1292
	Media	27,917	0.0005	0.1140
	Celebrities	11,565	0.0010	0.1666
	Artists	50,515	0.0006	0.1140
	Government	7,057	0.0036	0.2238
	TV Shows	3,892	0.0023	0.5906
Deezer	Croatia	54,573	0.0004	0.1146
	Hungary	47,538	0.0002	0.0929
	Romania	41,773	0.0001	0.0752

- *Facebook page networks:* These graphs represent mutual like networks among verified Facebook pages – the types of sites included TV shows, politicians, athletes, and artists among others.
- *Deezer user-user friendship networks:* We collected friendship networks from the music streaming site Deezer and included 3 European countries (Croatia, Hungary, and Romania). For each user, we curated the list of genres loved based on the songs liked by the user.

Standard hyperparameter settings

A fixed standard parameter setting is used our experiments, and we indicate any deviations. Models using first order random walk sampling strategy are referenced as *GEMSEC* and *Smooth GEMSEC*, second order random walk variants are named as *GEMSEC*$_2$ and *Smooth GEMSEC*$_2$. Random walks with length 80 are used and 5 truncated random walks per source node were used. Second-order random walk control hyperparameters [66] *return* and *in-out* were chosen from $\{2^{-2}, 2^{-1}, 1, 2, 4\}$. A window size of 5 is used for features. Each embedding has 16 dimensions and we extract 20 cluster centers. A parameter sweep over hyperparameters was used to obtain the highest average modularity. Initial learning rate values are chosen from $\{10^{-2}, 5 \cdot 10^{-3}, 10^{-3}\}$ and the final learning rate is chosen from $\{10^{-3}, 5 \cdot 10^{-4}, 10^{-4}\}$. Noise contrastive estimation uses 10 negative examples. The initial clustering cost coefficient is chosen from $\{10^{-1}, 10^{-2}, 10^{-3}\}$. The smoothness regularization term's hyperparameter is 0.0625 and Jaccard's coefficient is the penalty weight.

Detecting web graph communities

Using Facebook page networks we evaluate the clustering performance. Cluster quality is evaluated by modularity – we assume that a node belongs to a single community. Our results are summarized in Table 2.3 based on 10 experimental repetitions and errors in parentheses correspond to two standard deviations. The baselines use the hyperparameters from the respective

research papers. We used 16-dimensional embeddings throughout. The embeddings obtained with non-community-aware methods were clustered after the embedding by k-means clustering to extract 20 cluster centers. Specifically, comparisons are made with:

1. *Overlap Factorization* [8]: Factorizes the neighborhood overlap matrix to create features.
2. *DeepWalk* [147]: Approximates the sum of the adjacency matrix powers with first order random walks and implicitly factorizes it.
3. *LINE* [198]: Implicitly factorizes the sum of the first two powers for the normalized adjacency matrix and the resulting node representation vectors are concatenated together to form a multi-scale representation.
4. *Node2vec* [66]: Factorizes a neighbourhood matrix obtained with second order random walks. The *in-out* and *return* parameters of the second-order random walks were chosen from the $\{2^{-2}, 2^{-1}, 1, 2, 4\}$ set to maximize modularity.
5. *Walklets* [148]: Approximates with first order random walks each adjacency matrix power individually and implicitly factorizes the target matrix. These embeddings are concatenated to form a multi-scale representation of nodes.
6. *ComE* [24]: Uses a Gaussian mixture model to learn an embedding and clustering jointly using random walk features.
7. *M-NMF* [218]: Factorizes a matrix which is a weighted sum of the first two proximity matrices with a modularity based regularization constraint.
8. *DANMF* [238]: Decomposes a weighted sum of the first two proximity matrices hierarchically to obtain cluster memberships with an autoencoder-like non-negative matrix factorization model.

Smooth GEMSEC, *GEMSEC*$_2$ and *Smooth GEMSEC*$_2$ consistently outperform the neighborhood conserving node embedding methods and the competing community aware methods. The relative advantage of *Smooth GEMSEC*$_2$ over the benchmarks is highest on the Athletes dataset as the clustering's modularity is 3.44% higher than the best performing baseline. It is the worst on the Media dataset with a disadvantage of 0.35% compared to the strongest baseline. Use of smoothness regularization has sometimes non-significant, but definitely positive effect on the clustering performance of *Deepwalk*, *GEMSEC* and *GEMSEC*$_2$.

A social recommendation task for music genres

Node embeddings are often used for extracting features of nodes for downstream predictive tasks. In order to investigate this, we use social networks of Deezer users collected from European countries. We predict the genres (out of 84) of music liked by people. Following the embedding, we used logistic regression with ℓ_2 regularization to predict each of the labels and 90% of the nodes were randomly selected for training. We evaluated the performance of the remaining users. Numbers reported in Table 2.4 are F_1 scores calculated from 10 experimental repetitions.

TABLE 2.3: Mean modularity of clusterings on the Facebook datasets. Each embedding experiment was repeated ten times. Errors in the parentheses correspond to two standard deviations. In terms of modularity *Smooth GEMSEC*$_2$ outperforms the baselines.

	Politician	**Company**	**Athlete**	**Media**	**Celebrity**	**Artist**	**Government**	**TV Shows**
Overlap Factorization	.810 ±.008	.553 ±.010	.601 ±.020	.471 ±.016	.551 ±.01	.474 ±.018	.608 ±.024	.786 ±.008
DeepWalk	.840 ±.015	.637 ±.012	.649 ±.012	.481 ±.022	.631 ±.011	.508 ±.029	.686 ±.024	.811 ±.005
LINE	.841 ±.014	.651 ±.009	.665 ±.007	.558 ±.012	.642 ±.010	.557 ±.014	.690 ±.017	.813 ±.010
Node2Vec	.846 ±.012	.664 ±.008	.669 ±.007	.565 ±.011	.643 ±.013	.560 ±.010	.692 ±.017	.827 ±.016
Walklets	.843 ±.014	.655 ±.012	.664 ±.007	.562 ±.009	.621 ±.043	.548 ±.016	.689 ±.019	.819 ±.015
ComE	.830 ±.008	.654 ±.005	.665 ±.007	**.573** ±.005	.635 ±.010	.560 ±.011	.696 ±.010	.806 ±.011
M-NMF	.816 ±.014	.646 ±.007	.655 ±.008	.561 ±.004	.628 ±.006	.535 ±.021	.668 ±.011	.813 ±.008
DANMF	.810 ±.020	.648 ±.005	.650 ±.009	.560 ±.006	.628 ±.011	.532 ±.019	.673 ±.015	.812 ±.014
Smooth DeepWalk	.849 ±.017	.667 ±.007	.669 ±.007	.541 ±.006	.643 ±.008	.523 ±.020	.707 ±.008	.835 ±.008
GEMSEC	.851 ±.009	.662 ±.013	.674 ±.009	.536 ±.011	.636 ±.014	.528 ±.020	.705 ±.020	.833 ±.010
Smooth GEMSEC	.855 ±.006	.683 ±.009	**.692** ±.009	.567 ±.009	**.649** ±.008	.559 ±.011	.710 ±.008	.841 ±.004
GEMSEC$_2$	.852 ±.010	.667 ±.008	.683 ±.008	.551 ±.008	.638 ±.009	**.562** ±.020	**.712** ±.010	.838 ±.010
Smooth GEMSEC$_2$	**.859** ±.006	**.684** ±.009	**.692** ±.007	.571 ±.010	**.649** ±.011	**.562** ±.017	**.712** ±.010	**.847** ±.006

TABLE 2.4: Multi-label node classification performance of the embedding extracted features on the Deezer genre likes datasets. Performance is measured by average F_1 score values. Models were trained on 90% of the data and evaluated on the remaining 10%. Errors in the parentheses correspond to two standard deviations. *GEMSEC* models consistently have good performance.

	CROATIA			**HUNGARY**			**ROMANIA**		
	Micro	**Macro**	**Weighted**	**Micro**	**Macro**	**Weighted**	**Micro**	**Macro**	**Weighted**
Overlap Factorization	.319 ±.017	.026 ±.002	.208 ±.010	.361 ±.007	.029 ±.001	.227 ±.006	.275 ±.025	.020 ±.003	.167 ±.017
DeepWalk	.321 ±.006	.026 ±.002	.207 ±.004	.361 ±.004	.029 ±.002	.228 ±.002	.307 ±.008	.023 ±.002	.186 ±.006
LINE	.331 ±.013	.028 ±.002	.212 ±.010	.374 ±.007	.033 ±.002	.250 ±.005	.332 ±.007	.028 ±.002	.212 ±.006
Node2Vec	.348 ±.012	.032 ±.003	.235 ±.010	.393 ±.008	.037 ±.002	.267 ±.011	.346 ±.008	.031 ±.002	.229 ±.008
Walklets	.363 ±.013	.043 ±.003	.270 ±.012	.397 ±.007	.051 ±.001	.307 ±.006	.361 ±.011	**.050** ±.005	.281 ±.012
ComE	.326 ±.012	.028 ±.002	.217 ±.009	.363 ±.010	.033 ±.001	.246 ±.007	.323 ±.008	.028 ±.001	.212 ±.006
M-NMF	.336 ±.005	.028 ±.001	.217 ±.003	.369 ±.015	.032 ±.002	.239 ±.011	.330 ±.016	.028 ±.002	.209 ±.013
DANMF	.340 ±.007	.027 ±.002	.210 ±.002	.365 ±.011	.031 ±.002	.242 ±.008	.335 ±.009	.029 ±.002	.210 ±.012
Smooth DeepWalk	.329 ±.006	.028 ±.002	.215 ±.006	.375 ±.006	.032 ±.002	.244 ±.004	.321 ±.008	.026 ±.002	.204 ±.006
GEMSEC	.328 ±.006	.027 ±.002	.212 ±.004	.377 ±.004	.032 ±.002	.244 ±.004	.332 ±.008	.028 ±.002	.213 ±.006
Smooth GEMSEC	.333 ±.006	.028 ±.002	.215 ±.004	.379 ±.006	.034 ±.002	.250 ±.004	.334 ±.008	.029 ±.002	.215 ±.006
GEMSEC$_2$	**.381** ±.007	**.046** ±.003	**.287** ±.005	.407 ±.005	.050 ±.003	.310 ±.007	**.378** ±.009	.049 ±.003	**.289** ±.007
Smooth GEMSEC$_2$	.373 ±.005	.044 ±.002	.276 ±.006	**.409** ±.004	**.053** ±.002	**.314** ±.006	.376 ±.008	.049 ±.003	.287 ±.007

GEMSEC$_2$ significantly outperforms the other methods on all three countries' datasets. The performance advantage varies between 3.03% and 4.95%. We also see that *Smooth GEMSEC*$_2$ has lower accuracy, but it is able to outperform *DeepWalk*, *LINE*, *Node2Vec*, *Walklets*, *ComE*, *M-NMF* and *DANMF* on all datasets.

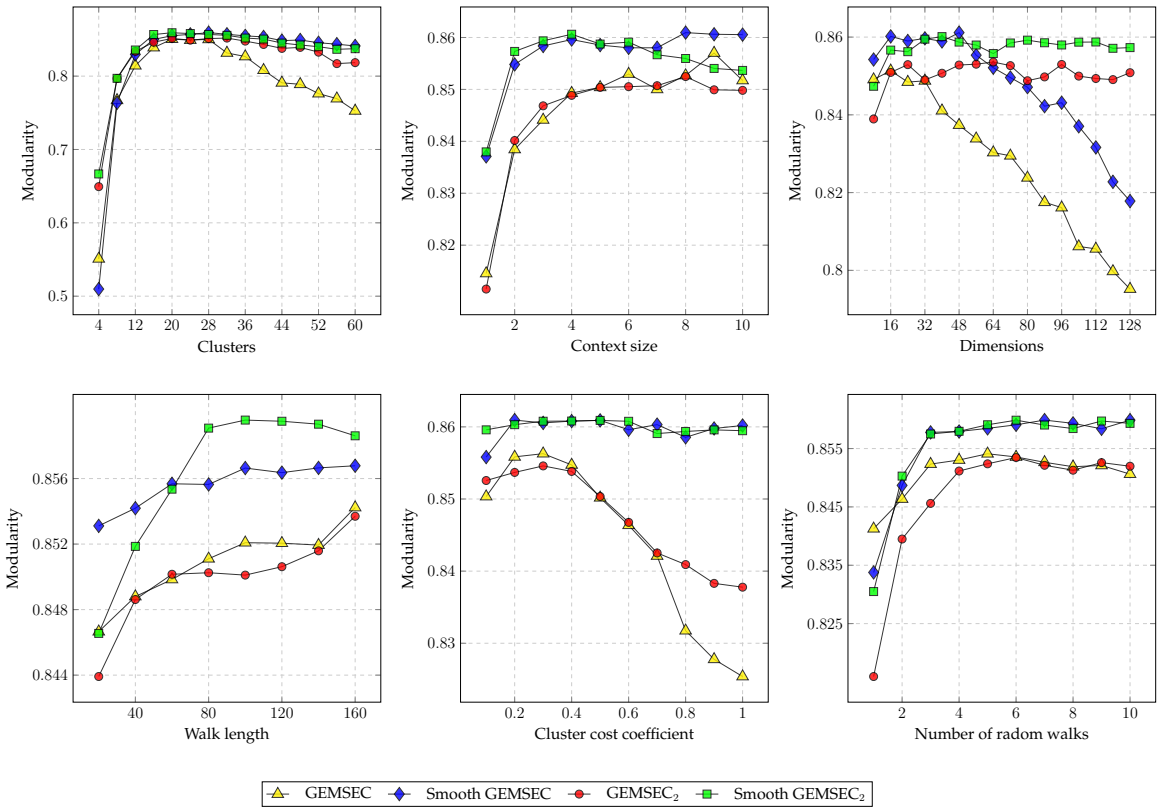


FIGURE 2.4: Sensitivity of cluster quality to parameter changes measured by modularity.

Sensitivity analysis

We tested the effect of hyperparameter changes to clustering performance. The Politicians Facebook graph is embedded with the standard parameter settings while the initial and final learning rates are set to be 10^{-2} and $5 \cdot 10^{-3}$ respectively, the clustering cost coefficient is 0.1 and we perturb certain hyperparameters. The second-order random walks used *in-out* and *return* parameters of 4. In Figure 2.4 each data point represents the mean modularity calculated from 10 experiments. Based on the experimental results we make two observations. First, *GEMSEC* model variants give high-quality clusters for a wide range of parameter settings. Second, introducing smoothness regularization makes *GEMSEC* models more robust to hyperparameter changes. This is particularly apparent across varying the number of clusters. The length of truncated random walks and the number of random walks per source node above a certain threshold has only a marginal effect on the community detection performance.

Scalability and computational efficiency

To create graphs of various sizes, we used the Erdos-Renyi model and with an average degree of 20. Figure 2.5 shows the log of mean runtime against the log of the number of nodes. Most importantly, we can conclude that doubling the size of the graph doubles the time needed for optimizing *GEMSEC*, thus the growth is linear. We also observe that embedding algorithms that incorporate clustering have a higher cost, and regularization also produces a higher cost, but similar growth.

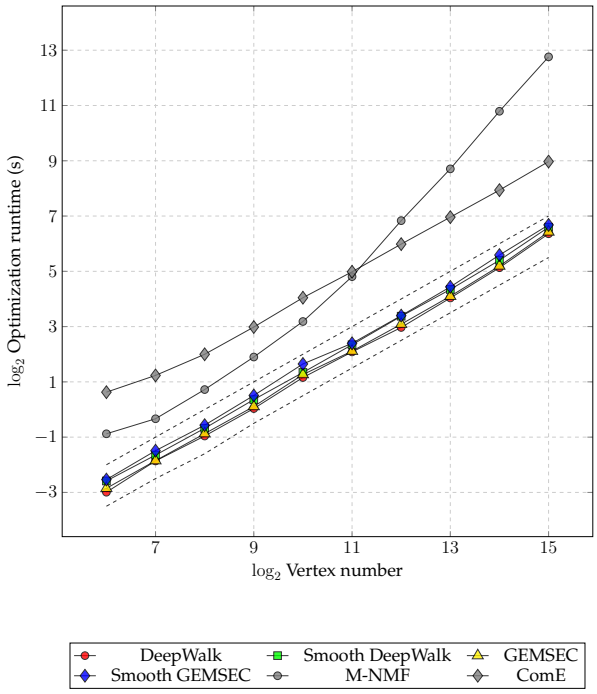


FIGURE 2.5: Sensitivity of optimization runtime to graph size measured by seconds. The dashed lines are linear references.

2.5 Conclusions

We described *GEMSEC* – a novel algorithm that learns a node embedding and a clustering of nodes jointly. It extends existing embedding modes. We showed that smoothness regularization is used to incorporate social network properties and produce natural embedding and clustering. We presented new social network datasets, and experimentally, our methods outperform a number of strong community aware node embedding baselines on node classification and community detection tasks.

Chapter 3

Multi-Scale Attributed Node Embedding

We present network embedding algorithms that capture information about a node from the local distribution over node attributes around it, as observed over random walks following an approach similar to Skip-gram. Observations from neighborhoods of different sizes are either pooled (*AE*) or encoded distinctly in a multi-scale approach (*MUSAE*). Capturing attribute-neighborhood relationships over multiple scales is useful for a diverse range of applications, including latent feature identification across disconnected networks with similar attributes. We prove theoretically that matrices of node-feature pointwise mutual information are implicitly factorized by the embeddings. Experiments show that our algorithms are robust, computationally efficient and outperform comparable models on social, web and citation network datasets.

3.1 Introduction

Node embedding is a fundamental technique in network analysis that serves as a precursor to numerous downstream machine learning and optimisation tasks, e.g. community detection, network visualization and link prediction [66, 147, 198]. Several recent network embedding methods, such as *Deepwalk* [147], *Node2Vec* [66] and *Walklets* [148], achieve impressive performance by learning the network structure following an approach similar to *Word2Vec Skip-gram* [130], originally designed for word embedding. In these works, sequences of neighboring nodes are generated from random walks over a network, and representations are distilled from extracted node-node proximity statistics that capture local neighbourhood information.

When the nodes of a network have attributes (or features), their embeddings can be used to capture information about the attributes in their local neighbourhood. For a social network, attributes might represent a person's interests, habits, history or preferences. The pattern of node attributes are often similar in a neighborhood, and conversely, nodes with similar attributes are more likely to be connected. This property is known as *homophily*. Attributed network embedding methods [81, 116, 227] leverage this additional information to supplement that of node neighbourhood structure, benefiting many applications, e.g. recommender systems, node classification and link prediction [231, 234, 244].

The neighborhood of a node can be considered at different path lengths, or *scales*. In a social network, near neighbors may correspond to classmates, whereas nodes separated by greater

scales may be in different cities or countries. Attributes of neighbors at different scales can be considered separately (*multi-scale*) or *pooled* in some way (e.g. weighted average). Figure 3.1 shows how the attribute distribution over neighbourhoods at different scales can indicate nodes with similar network *roles* even if they are distant in the network, or even in different networks. Methods that take attributes of nearby nodes into account generalizes those that do not, e.g. [148], for which feature vectors can be considered standard basis vectors.

Many embedding methods correspond to matrix factorization, indeed some attributed embedding methods (e.g. [231]) explicitly factorize a matrix of link-attribute information. Embeddings learned using Skip-gram are known to factorize a matrix of *pointwise mutual information* (PMI) of co-occurrences between each word and local *context* words [110]. Related network embedding methods [66, 147, 154, 198] also implicitly factorize PMI matrices based on the probability of encountering each (context) node on a random walk from each starting node [154].

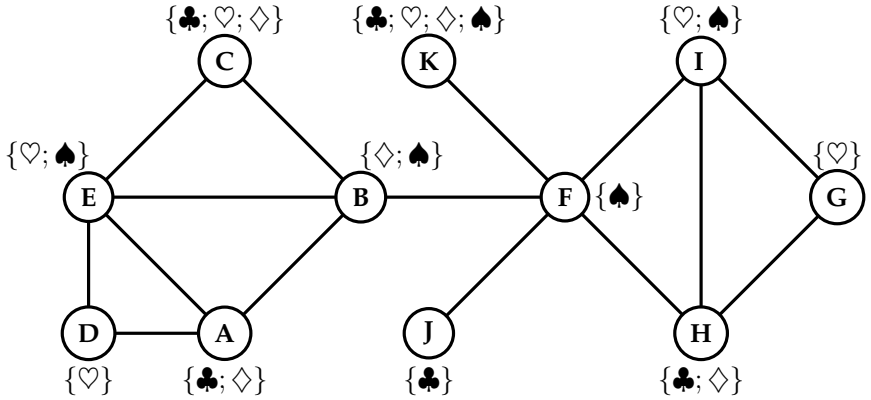


FIGURE 3.1: The phenomenon affecting and inspiring the design of the multi-scale attributed network embedding procedure. Attributed nodes D and G have the same feature set and their nearest neighbours also exhibit equivalent sets of features, whereas features at higher order neighbourhoods differ. A multi-scale attributed node embedding method is able to represent these differences and similarities in the embedding space.

Our key contributions are:

1. We introduce the first Skip-gram style embedding algorithms that consider attribute distributions over local neighborhoods, both pooled (*AE*) and multi-scale (*MUSAE*), and their counterparts that attribute distinct features to each node (*AE-EGO* and *MUSAE-EGO*).

2. We theoretically prove that the embeddings approximately factorize PMI matrices based on the product of an adjacency matrix power and node-feature matrix.

3. We show that popular network embedding methods *DeepWalk* [147] and *Walklets* [148] are special cases of our *AE* and *MUSAE*.

4. We show empirically that *AE* and *MUSAE* embeddings enable strong performance at regression, classification, and link prediction tasks for real-world networks (e.g. Wikipedia and Facebook), are computationally scalable and enable transfer learning between networks.

We provide reference implementations of *AE* and *MUSAE*, together with the datasets used for evaluation at `https://github.com/benedekrozemberczki/MUSAE`.

3.2 Related work

Efficient unsupervised learning of node embeddings for large networks has seen unprecedented development in recent years. The current paradigm focuses on learning latent space representations of nodes such that those that share neighbors [66, 147, 148, 198], structural roles [11, 75, 76, 159] or attributes are located close together in the embedding space. Our work falls under the last of these categories as our goal is to learn similar latent representations for nodes with similar sets of features in their neighborhoods, both on a pooled and multi-scale basis.

A general overview

Neighborhood preserving node embedding procedures place nodes with common first, second and higher order neighbors within close proximity in the embedding space. Recent works in the neighborhood preserving node embedding literature were inspired by the *Skip-gram* model [129, 130], which generates word embeddings by implicitly factorizing a shifted pointwise mutual information (PMI) matrix [110] obtained from a text corpus. This procedure inspired *DeepWalk* [147], a method which generates truncated random walks over a graph to obtain a "corpus" from which the *Skip-gram* model generates neighborhood preserving node embeddings. In doing so, *DeepWalk* implicitly factorizes a PMI matrix, which can be shown, based on the underlying first-order Markov process, to correspond to the mean of a set of normalized adjacency matrix powers up to a given order [154]. Such *pooling* of matrices can be suboptimal since neighbors over increasing path lengths (or scales) are treated equally or according to fixed weightings [66, 129]; whereas it has been found that an optimal weighting may be task or dataset specific [4]. In contrast, multi-scale node embedding methods such as *LINE* [198], *GraRep* [22] and *Walklets* [148] separately learn lower-dimensional node embedding components from each adjacency matrix power and concatenate them to form the full node representation. Such un-pooled representations, comprising distinct but less information at each scale, are found to give higher performance in a number of downstream settings, without increasing the overall number of free parameters [148].

Attributed node embedding procedures refine ideas from neighborhood based node embeddings to also incorporate node *attributes* (equivalently, features or labels) [81, 116, 227, 231, 234]. Similarities between both a node's neighborhood structure and features contribute to determining pairwise proximity in the node embedding space. These models follow quite different strategies to obtain such representations. The most elemental procedure, *TADW* [227], decomposes a convex combination of normalized adjacency matrix powers into a matrix product that includes the feature matrix. Several other models, such as *SINE* [244] and *ASNE* [116], implicitly factorize a matrix formed by concatenating the feature and adjacency matrices. Other approaches such as *TENE* [234], formulate the attributed node embedding task as a joint non-negative matrix factorization problem in which node representations obtained from sub-tasks are used to regularize one another. *AANE* [81] uses a similar network structure based regularization approach, in which a node feature similarity matrix is decomposed using the alternating direction method of multipliers. The method most similar to our own is *BANE* [231], in which the product of a normalized adjacency matrix power and a feature matrix is explicitly factorized to obtain

attributed node embeddings. Many other methods exist, but do not consider the attributes of higher order neighborhoods [81, 116, 116, 227, 244].

The relationship between our pooled (*AE*) and multi-scale (*MUSAE*) attributed node embedding methods mirrors that between graph convolutional neural networks (GCNNs) and multi-scale GCNNs. Widely used graph convolutional layers, such as *GCN* [92], *GraphSage* [71], *GAT* [210], *APPNP* [93], *SGCONV* [222] and *ClusterGCN* [28], create latent node representations that pool node attributes from arbitrary order neighborhoods, which are then inseparable and unrecoverable. In contrast, *MixHop* [5] learns latent features for each proximity.

A desiderata based comparison

A node representation learning technique must have certain desired properties in order to generate expressive vertex features and have scalability. We summarized these beneficial properties of node embedding techniques in Table 3.1 with the respective space and time complexities.

TABLE 3.1: A summary of existing node embedding techniques (proximity preserving and attributed) with respect to having (✔) and missing (✘) desired properties. The time and space complexities are reported as a function of vertex and edge counts ($|\mathbb{V}|$ and $|\mathbb{E}|$), unique feature count $|\mathbb{F}|$, average node feature count m, and embedding dimensions d.

	Generic Features	Multi Scale	Implicit	Proximal	Higher Order	Inductive	Non-linear	Space Complexity	Time Complexity
DeepWalk [147]	✘	✘	✔	✔	✔	✘	✔	$\mathcal{O}(\lvert\mathbb{V}\rvert\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ d)$
LINE$_2$ [198]	✘	✔	✔	✔	✔	✘	✔	$\mathcal{O}(\lvert\mathbb{V}\rvert\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ d)$
Node2Vec [66]	✘	✘	✔	✔	✔	✘	✔	$\mathcal{O}(\lvert\mathbb{V}\rvert^3)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ d)$
Walklets [148]	✘	✔	✔	✔	✔	✘	✔	$\mathcal{O}(\lvert\mathbb{V}\rvert\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ d)$
NetMF [154]	✘	✘	✘	✔	✔	✘	✘	$\mathcal{O}(\lvert\mathbb{V}\rvert^2\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert^3\ d)$
HOPE [139]	✘	✘	✘	✔	✔	✘	✘	$\mathcal{O}(\lvert\mathbb{V}\rvert^2\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert^3\ d)$
GraRep [22]	✘	✔	✘	✔	✔	✘	✘	$\mathcal{O}(\lvert\mathbb{V}\rvert^2\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert^3\ d)$
TADW [227]	✔	✘	✘	✘	✔	✘	✘	$\mathcal{O}((\lvert\mathbb{V}\rvert + \lvert\mathbb{F}\rvert)\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert^2\ \lvert\mathbb{F}\rvert\ d)$
ASNE [116]	✔	✘	✘	✔	✘	✘	✘	$\mathcal{O}((\lvert\mathbb{V}\rvert + \lvert\mathbb{F}\rvert)\ d)$	$\mathcal{O}(\lvert\mathbb{E}\rvert\ m\ d)$
AANE [81]	✔	✘	✘	✔	✘	✘	✔	$\mathcal{O}(\lvert\mathbb{V}\rvert^2\ m\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert^2\ m\ d)$
BANE [231]	✔	✘	✘	✘	✔	✘	✘	$\mathcal{O}(\lvert\mathbb{V}\rvert^2\ m\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert^3\ m\ d)$
TENE [234]	✔	✘	✘	✔	✘	✘	✘	$\mathcal{O}((\lvert\mathbb{V}\rvert + \lvert\mathbb{F}\rvert)\ d)$	$\mathcal{O}(\lvert\mathbb{E}\rvert\ m\ d)$
AE	✔	✘	✔	✘	✔	✔	✔	$\mathcal{O}((\lvert\mathbb{V}\rvert + \lvert\mathbb{F}\rvert)\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ m\ d)$
MUSAE	✔	✔	✔	✘	✔	✔	✔	$\mathcal{O}((\lvert\mathbb{V}\rvert + \lvert\mathbb{F}\rvert)\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ m\ d)$
AE-EGO	✔	✘	✔	✔	✔	✘	✔	$\mathcal{O}((\lvert\mathbb{V}\rvert + \lvert\mathbb{F}\rvert)\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ m\ d)$
MUSAE-EGO	✔	✔	✔	✔	✔	✘	✔	$\mathcal{O}((\lvert\mathbb{V}\rvert + \lvert\mathbb{F}\rvert)\ d)$	$\mathcal{O}(\lvert\mathbb{V}\rvert\ m\ d)$

- **Generic features:** Generic vertex properties such as the income of users in a social network are encoded when a node embedding is learned. These generic features are used to contextualize the location of the node in the embedding space universally.
- **Multi-scale:** Information obtained from distinct proximities (e.g. random walk hops, shortest path distance) is encoded by distinct groups of node embedding features. Using a multi-scale node representation the micro-, meso-, and macroscopic context of a node can be discerned.

- **Implicit:** The decomposed target matrix is not calculated explicitly. This reduces the required space and time complexity, which makes the embedding model applicable in practical large-scale industrial settings.
- **Proximal:** The contextual proximity information (location in comparison to other vertices) about the node is encoded when the node embedding is created. A proximal node embedding cannot be inductive, as the proximal context is not be meaningful in disjoint graphs.
- **Higher-order:** The embedding encodes information from nodes that are not adjacent to a node. For example, in random walk based contextualization the information is obtained from multiple hops, not just the first step.
- **Inductive:** The node embedding technique can map unseen nodes to the embedding space which are not connected to the graph used for training. An embedding technique which contextualizes the nodes based on the proximity to other nodes cannot be inductive.
- **Non-linear:** A node-node proximity score in the target matrix is not a linear function of the two node embedding vectors. This property allows for super and sub linear proximity score encoding.

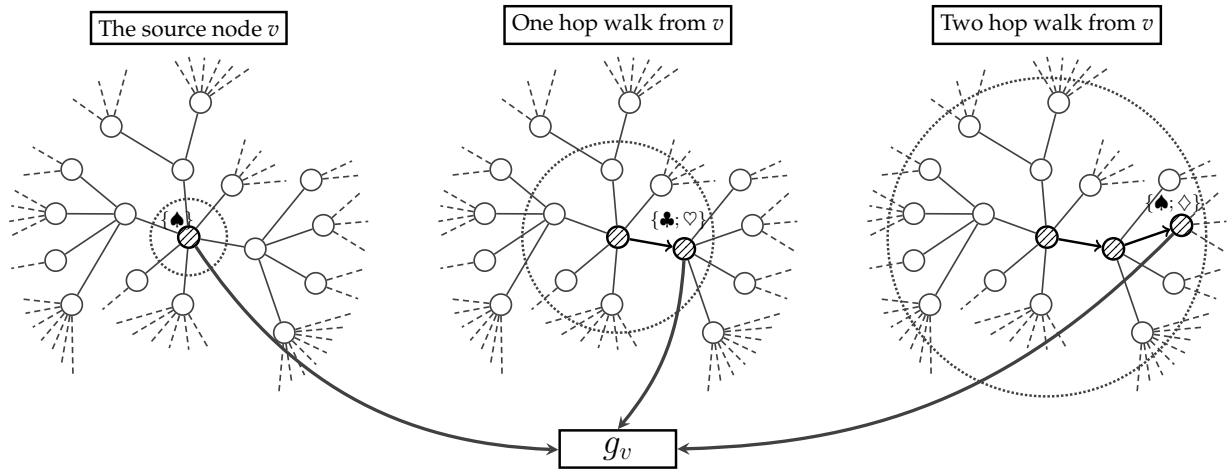
3.3 Attributed embedding algorithms

Here we define our algorithms to jointly learn embeddings of nodes and attributes based on the structure and attributes of local neighborhoods. The aim is to learn similar embeddings for nodes that occur in neighborhoods of similar attributes; and similar embeddings for attributes that occur in similar neighborhoods of nodes.

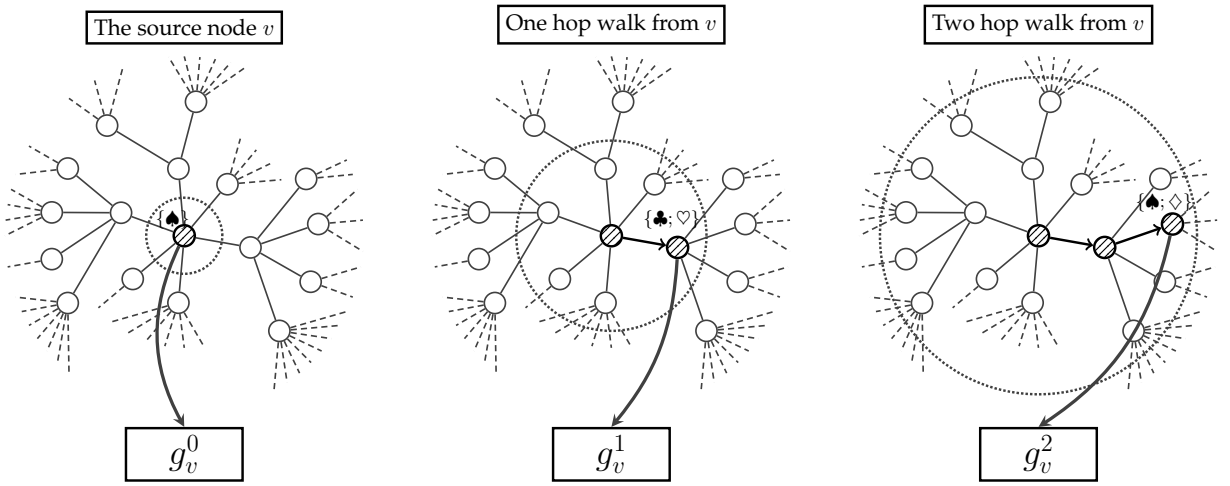
Let $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ be an undirected graph, where $\mathbb{V}$ and $\mathbb{E}$ are the sets of vertices and edges (or links), and let $\mathbb{F}$ be the set of all binary node features. For each node $v \in \mathbb{V}$, let $\mathbb{F}_v \subseteq \mathbb{F}$ be the subset of features belonging to v. An embedding of nodes is a mapping $g : \mathbb{V} \to \mathbb{R}^d$ that assigns a d-dimensional vector $g(v)$, or simply g_v, to each node v and is fully described by a matrix $\boldsymbol{G} \in \mathbb{R}^{|\mathbb{V}| \times d}$. An embedding of the features (to the same latent space) is a mapping $h : \mathbb{F} \to \mathbb{R}^d$ with embeddings denoted $h(f) \doteq h_f$, as summarised by a matrix $\boldsymbol{H} \in \mathbb{R}^{|\mathbb{F}| \times d}$.

Attributed embedding

The *Attributed Embedding* (*AE*) method is described by Algorithm 2 and the main idea is figuratively summarized in Figure 3.2a. With probability proportional to node degree, s starting nodes v_1 are sampled from $\mathbb{V}$ (line 2). From each starting node, a node sequence of length l is sampled over $\mathcal{G}$ following a first-order random walk (line 3). For a given window size t, iterate over the first $l-t$ nodes of the sequence v_j, termed *source* nodes (line 4). For each source node, the subsequent t nodes are considered *target* nodes (line 5). For each target node v_{j+r}, the tuple (v_j, f) is added to the corpus $\mathbb{D}$ for each feature $f \in \mathbb{F}_{v_{j+r}}$ (lines 6-7). Each tuple (v_{j+r}, f) for features of the source node $f \in \mathbb{F}_{v_j}$ is also added to $\mathbb{D}$ (lines 9-10). Running Skip-gram on $\mathbb{D}$ with b negative samples (line 15) generates the d-dimensional node and feature embeddings ($\boldsymbol{G}$, $\boldsymbol{H}$).



(A) Pooled attributed embedding of node v.



(B) Multi-scale attributed embedding of node v.

FIGURE 3.2: In **Figure 3.2a** we learn a pooled node embedding g_v of the source node $v \in \mathbb{V}$ using features from the 1^{st} and 2^{nd} order proximity. We learn g_v by using the union of feature sets as a context which is defined by the multi-set $\{\spadesuit; \clubsuit; \heartsuit; \spadesuit; \diamondsuit\}$ which contains features of the source node and nodes appearing in the random walk in one and two hops from the source. Comparatively, in **Figure 3.2b** we learn a multi-scale embedding of the node by learning individual embeddings of for each proximity noted by g_v^0, g_v^1 and g_v^2. These embeddings are contextualized by the features sets $\{\spadesuit\}$, $\{\clubsuit; \heartsuit\}$ and $\{\spadesuit; \diamondsuit\}$ respectively. Concatenated together g_v^0, g_v^1 and g_v^2 forms the node embedding.

Multi-scale attributed embedding

The *AE* algorithm pools features across neighborhoods of different proximity. Inspired by the performance of (unattributed) multi-scale node embeddings, we adapt *AE* to learn *multi-scale* attributed embeddings. The embedding component of a node $v \in \mathbb{V}$ at proximity $r \in \{1, ..., t\}$ is given by a mapping $g^r : \mathbb{V} \rightarrow \mathbb{R}^{d/t}$ (where t divides d). Similarly, the embedding component of feature $f \in \mathbb{F}$ at proximity r is given by a mapping $h^r : \mathbb{F} \rightarrow \mathbb{R}^{d/t}$. Concatenating the t components gives a d-dimensional embedding for each node and feature. The *Multi-Scale Attributed Embedding* (*MUSAE*) method is described by Algorithm 3 and the main idea is figuratively summarized in Figure 3.2b. Source and target node pairs are generated from sampled node sequences as for *AE* (lines 2-5). Each feature of a target node $f \in \mathbb{F}_{v_{j+r}}$ is again considered, but tuples (v_j, f) are added to a *sub-corpus* $\mathbb{D}_{\underset{r}{\rightarrow}}$ (lines 6-7) and for each source node feature $f \in \mathbb{F}_{v_j}$ tuples (v_{j+r}, f) are

Data: $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ – Graph to be embedded.
$\{\mathbb{F}_v\}_{\mathbb{V}}$ – Set of node feature sets.
s – Number of sequence samples.
l – Length of sequences.
t – Context size.
d – Embedding dimension.
b – Number of negative samples.

Result: Node embedding g and feature embedding h.

```
1  for i in 1 : s do
2  |  Pick v_1 ∈ V according to P(v) ~ deg(v)/vol(G).
3  |  (v_1, v_2, ..., v_l) ← Sample Nodes(G, v_1, l)
4  |  for j in 1 : l − t do
5  |  |  for r in 1 : t do
6  |  |  |  for f in F_{v_{j+r}} do
7  |  |  |  |  Add tuple (v_j, f) to multiset D.
8  |  |  |  end
9  |  |  |  for f in F_{v_j} do
10 |  |  |  |  Add tuple (v_{j+r}, f) to multiset D.
11 |  |  |  end
12 |  |  end
13 |  end
14 end
15 Run SGNS on D with b negative samples and d dimensions.
16 Output g_v, ∀v ∈ V, and h_f, ∀f ∈ F = ∪_V F_v.
```

Algorithm 2: Attributed node embedding sampling and training procedure.

added to another sub-corpus $\mathbb{D}_{\overleftarrow{r}}$ (lines 9-10). Running Skip-gram with b negative samples on each sub-corpus $\mathbb{D}_r = \mathbb{D}_{\overrightarrow{r}} \cup \mathbb{D}_{\overleftarrow{r}}$ (line 17) generates the $\frac{d}{t}$-dimensional components to concatenate.

3.4 Attributed embedding as implicit matrix factorization

[110] showed that the loss function of Skip-gram with negative sampling (SGNS) is minimized if its two output embedding matrices $\boldsymbol{W}, \boldsymbol{C}$ factorize a matrix of pointwise mutual information (PMI) of word co-occurrence statistics. Specifically, for a corpus $\mathbb{D}$ over a dictionary $\mathbb{V}$ with $|\mathbb{V}| = n$, SGNS (with b negative samples) generates embeddings $\boldsymbol{w}_w, \boldsymbol{c}_c \in \mathbb{R}^d$ (columns of $\boldsymbol{W}, \boldsymbol{C} \in \mathbb{R}^{d \times n}$) for each target and context word $w, c \in \mathbb{V}$, satisfying:

$$\boldsymbol{w}_w^\top \boldsymbol{c}_c \approx \log\left(\frac{\#(w,c)|\mathbb{D}|}{\#(w)\#(c)}\right) - \log b \,,$$

where $\#(w)$, $\#(c)$ and $\#(w,c)$ denote counts of w, c and both words appearing within a sliding context window. Considering $\frac{\#(w)}{|\mathbb{D}|}$, $\frac{\#(c)}{|\mathbb{D}|}$, $\frac{\#(w,c)}{|\mathbb{D}|}$ as empirical estimates of $p(w)$, $p(c)$ and $p(w,c)$ respectively gives the approximate low-rank factorization (since typically $d \ll n$):

$$\boldsymbol{W}^\top \boldsymbol{C} \approx \left[\,\mathrm{PMI}(w,c) - \log b\,\right]_{w,c \in \mathbb{V}} \,,$$

[154] extended this result to node embedding models that apply SGNS to a "corpus" generated from random walks over the graph. In the case of *DeepWalk* where random walks are first-order Markov, the joint probability distributions over nodes at different steps of a random walk can

Data: $\mathcal{G} = (\mathbb{V}, \mathbb{E})$ – Graph to be embedded.
$\{\mathbb{F}_v\}_{\mathbb{V}}$ – Set of node feature sets.
s – Number of sequence samples.
l – Length of sequences.
t – Context size.
d – Embedding dimension.
b – Number of negative samples.

Result: Node embedding components g^r and feature embeddings component h^r, for $r \in \{1, ..., t\}$.

for i in $1 : s$ **do**
  Pick $v_1 \in \mathbb{V}$ according to $P(v) \sim deg(v)/\text{vol}(\mathcal{G})$.
  $(v_1, v_2, \ldots, v_l) \leftarrow$ Sample Nodes$(\mathcal{G}, v_1, l)$
  for j in $1 : l - t$ **do**
    for r in $1 : t$ **do**
      for f in $\mathbb{F}_{v_{j+r}}$ **do**
        Add the tuple (v_j, f) to multiset $\mathbb{D}_{\underset{r}{\rightarrow}}$.
      end
      for f in $\mathbb{F}_{v_j}$ **do**
        Add the tuple (v_{j+r}, f) to multiset $\mathbb{D}_{\underset{r}{\leftarrow}}$.
      end
    end
  end
end
for r in $1 : t$ **do**
  Create $\mathbb{D}_r$ by unification of $\mathbb{D}_{\underset{r}{\rightarrow}}$ and $\mathbb{D}_{\underset{r}{\leftarrow}}$.
  Run SGNS on $\mathbb{D}_r$ with b negative samples and $\frac{d}{t}$ dimensions.
  Output $g_v^r,\ \forall v \in \mathbb{V}$, and $h_f^r,\ \forall f \in \mathbb{F} = \cup_{\mathbb{V}} \mathbb{F}_v$.
end

Algorithm 3: Multi-scale attributed node embedding sampling and training procedure.

be expressed in closed form, and a closed form for the factorized PMI matrix follows. Here we derive the matrices implicitly factorized by *AE* and *MUSAE*.

Notation: For a graph $\mathcal{G} = (\mathbb{V}, \mathbb{E})$, $|\mathbb{V}| = n$, let $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ denote the adjacency matrix and $\boldsymbol{D} \in \mathbb{R}^{n \times n}$ the diagonal degree matrix, i.e. $\boldsymbol{D}_{v,v} = \deg(v) = \sum_w \boldsymbol{A}_{v,w}$. Let $c = \sum_{v,w} \boldsymbol{A}_{v,w}$ denote the *volume* of $\mathcal{G}$. We define the binary attribute matrix $\boldsymbol{F} \in \{0,1\}^{|\mathbb{V}| \times |\mathbb{F}|}$ by $\boldsymbol{F}_{v,f} = \mathbb{1}_{f \in \mathbb{F}_v},\ \forall v \in \mathbb{V}, f \in \mathbb{F}$. To ease notation, we set $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{A}$ and $\boldsymbol{E} = diag(\mathbf{1}^\top \boldsymbol{D}\boldsymbol{F})$, where $diag$ indicates a diagonal matrix.

Interpretation: Assuming $\mathcal{G}$ is ergodic, $p(v) = \frac{deg(v)}{c}$ is the stationary distribution over nodes $v \in \mathbb{V}$, i.e. $c^{-1}\boldsymbol{D} = diag(p(v))$; and $c^{-1}\boldsymbol{A}$ is the stationary joint distribution over consecutive nodes of a random walk $p(v_j, v_{j+1})$. $\boldsymbol{F}_{v,f}$ can be considered a Bernoulli parameter of the probability $p(f|v)$ of observing feature f at a node v, hence $c^{-1}\boldsymbol{D}\boldsymbol{F}$ describes the stationary joint distribution $p(f, v)$ over nodes and features. Accordingly, $\boldsymbol{P}$ is the transition matrix of conditional probabilities $p(v_{j+1}|v_j)$; and $\boldsymbol{E}$ is a diagonal matrix proportional to the (binary) probability $p(f)$ of observing feature f at the stationary distribution. We note that $p(f)$ need not sum to 1, whereas $p(v)$ necessarily must.

Multi-scale case (MUSAE)

We know that the SGNS aspect of *MUSAE* learns embeddings g_v^r, h_f^r that satisfy ${g_v^r}^\top h_f^r \approx \log\left(\frac{\#(v,f)_r |\mathbb{D}_r|}{\#(v)_r \#(f)_r}\right) - \log b$, $\forall v \in \mathbb{V}, f \in \mathbb{F}$. Our aim is to express this factorization in terms of known properties of the graph $\mathcal{G}$ and its features.

Lemma 3.1. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps **(i) after; or (ii) before** node $v \in \mathbb{V}$, as given by:*

$$\underset{l\to\infty}{plim} \frac{\#(v,f)_{\overrightarrow{r}}}{|\mathbb{D}_{\overrightarrow{r}}|} = c^{-1}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f}$$
$$\underset{l\to\infty}{plim} \frac{\#(v,f)_{\overleftarrow{r}}}{|\mathbb{D}_{\overleftarrow{r}}|} = c^{-1}(\boldsymbol{F}^\top\boldsymbol{D}\boldsymbol{P}^r)_{f,v}$$

Proof. See Appendix A. □

Lemma 3.2. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps **either side** of node $v \in \mathbb{V}$, given by:*

$$\underset{l\to\infty}{plim} \frac{\#(v,f)_r}{|\mathbb{D}_r|} = c^{-1}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f} \ ,$$

Proof. See Appendix A. □

Marginalizing gives unbiased estimates of stationary probability distributions of nodes and features:

$$\underset{l\to\infty}{plim} \frac{\#(v)}{|\mathbb{D}_r|} = \frac{deg(v)}{c} = c^{-1}\boldsymbol{D}_{v,v}$$
$$\underset{l\to\infty}{plim} \frac{\#(f)}{|\mathbb{D}_r|} = \sum_{v|f\in\mathbb{F}_v} \frac{deg(v)}{c} = c^{-1}\boldsymbol{E}_{f,f}$$

Theorem 3.3. *MUSAE embeddings approximately factorize the node-feature PMI matrix:*

$$log\left(c\,\boldsymbol{P}^r\boldsymbol{F}\boldsymbol{E}^{-1}\right) - \log b, \quad for\ r = 1, \ldots, t.$$

Proof.

$$\begin{aligned}\frac{\#(v,f)_r|\mathbb{D}_r|}{\#(f)_r\#(v)_r} &= \left(\frac{\#(v,f)_r}{|\mathbb{D}_r|}\right)/\left(\frac{\#(f)_r}{|\mathbb{D}_r|}\frac{\#(v)_r}{|\mathbb{D}_r|}\right)\\ &\xrightarrow{p} \left((c\boldsymbol{D}^{-1})(c^{-1}\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})(c\boldsymbol{E}^{-1})\right)_{v,f}\\ &= c(\boldsymbol{P}^r\boldsymbol{F}\boldsymbol{E}^{-1})_{v,f}\end{aligned}$$

□

Pooled case (AE)

Lemma 3.4. *The empirical statistics of node-feature pairs learned by the AE algorithm give unbiased estimates of mean joint probabilities over different path lengths:*

$$\underset{l\to\infty}{plim} \frac{\#(v,f)}{|\mathbb{D}|} = \frac{c}{t}\left(\boldsymbol{D}(\sum_{r=1}^{t}\boldsymbol{P}^r)\boldsymbol{F}\right)_{v,f}$$

Proof. By construction, $|\mathbb{D}| = \sum_r |\mathbb{D}_r|$, $\#(v,f) = \sum_r \#(v,f)_r$, $|\mathbb{D}_r| = |\mathbb{D}_s|\ \forall\ r,s \in \{1,...,t\}$ and so $|\mathbb{D}_s| = t^{-1}|\mathbb{D}|$. Combining with Lemma 3.2, the result follows. □

Theorem 3.5. *AE embeddings approximately factorize the pooled node-feature matrix:*

$$\log\left(\tfrac{c}{t}(\sum_{r=1}^{t}\boldsymbol{P}^r)\boldsymbol{F}\boldsymbol{E}^{-1}\right) - \log b\,.$$

Proof. Analogous to the proof of Theorem 3.3. □

Remark 3.6. *DeepWalk is a special case of AE with* $\boldsymbol{F} = \boldsymbol{I}_{|\mathbb{V}|}$.

That is, *DeepWalk* is equivalent to *AE* if each node has a single unique feature. Thus $\boldsymbol{E} = diag(\mathbf{1}^\top \boldsymbol{D}\boldsymbol{I}) = \boldsymbol{D}$ and, by Theorem 3.5, the embeddings of *DeepWalk* factorize $\log\left(\tfrac{c}{t}(\sum_{r=1}^{t}\boldsymbol{P}^r)\boldsymbol{D}^{-1}\right) - \log b$, as derived by [154].

Remark 3.7. *Walklets is a special case of MUSAE with* $\boldsymbol{F} = \boldsymbol{I}_{|\mathbb{V}|}$.

Thus, for $r = 1, \ldots, t$, the embeddings of *Walklets* factorise $\log\left(c\,\boldsymbol{P}^r\boldsymbol{D}^{-1}\right) - \log b$.

Remark 3.8. *Appending an identity matrix* $\boldsymbol{I}$ *to the feature matrices* $\boldsymbol{F}$ *of AE and MUSAE (denoted* $[\boldsymbol{F}; \boldsymbol{I}]$*) adds a unique feature to each node. The resulting algorithms, named AE-EGO and MUSAE-EGO, respectively, learn embeddings that approximately factorize the node-feature PMI matrices:*

$$\log\left(c\,\boldsymbol{P}^r\,[\boldsymbol{F};\boldsymbol{I}]\,\boldsymbol{E}^{-1}\right) - \log b, \quad \forall r \in \{1, ..., t\};$$

and $$\log\left(\tfrac{c}{t}(\sum_{r=1}^{t}\boldsymbol{P}^r)\,[\boldsymbol{F};\boldsymbol{I}]\,\boldsymbol{E}^{-1}\right) - \log b\,.$$

Complexity analysis

Assuming a constant number of features per source node, the corpus generation has runtime complexity of $\mathcal{O}(s\,l\,t\,\frac{m}{n})$, where $m = \sum_{v\in\mathbb{V}}|\mathbb{F}_v|$ the total number of features across all nodes (with repetition), $q = |\mathbb{F}|$, and $n = |\mathbb{V}|$. With b negative samples, the optimization runtime of a single asynchronous gradient descent epoch on *AE* and the joint optimization runtime of *MUSAE* embeddings is $\mathcal{O}(b\,d\,s\,l\,t\,\frac{m}{n})$. With p truncated walks from each source node, the corpus generation complexity is $\mathcal{O}(p\,n\,l\,t\,m)$ and the model optimization runtime is $\mathcal{O}(b\,d\,p\,n\,l\,t\,m)$. The runtime experiments (Section 3.5) empirically support this analysis.

Corpus generation has a memory complexity of $\mathcal{O}(s\,l\,t\,\frac{m}{n})$ while the same when generating p truncated walks per node has a memory complexity of $\mathcal{O}(p\,n\,l\,t\,m)$. Storing the parameters of an *AE* embedding has a memory complexity of $\mathcal{O}((n+q)\cdot d)$ and *MUSAE* embeddings also use $\mathcal{O}((n+q)\cdot d)$ memory.

3.5 Experimental evaluation

We evaluate the representations learned by *AE*, *MUSAE* and their *EGO* extensions on several common downstream tasks: node classification, regression, link prediction and also transfer learning across networks. We also report how number of nodes and dimensionality affect runtime. We use standard citation graph benchmark datasets (Cora, Citeseer [118] and Pubmed [133])

together with datasets of social networks and web graphs collected from Facebook, Github, Wikipedia and Twitch.

Datasets

Our method was evaluated on a variety of social networks and web page-page graphs that we collected from openly available API services. In Table 3.2 we described the graphs with widely used statistics with respect to size, diameter, and level of clustering. We also included the average number of features per vertex and unique feature count in the last columns. These datasets are available with the source code of *MUSAE* and *AE* at `https://github.com/iclr2020/MUSAE`.

TABLE 3.2: Descriptive statistics of node level datasets used for evaluating MUSAE and AE.

Dataset	Nodes	Edges	Diameter	Clustering Coefficient	Density	Average Feature	Unique Features
Facebook Page-Page	22,470	171,002	15	0.232	0.001	14.000	4,714
GitHub Web-ML	37,700	289,003	7	0.013	0.001	18.312	4,005
Wikipedia Chameleon	2,277	31,421	11	0.314	0.012	21.547	3,132
Wikipedia Crocodile	11,631	170,918	11	0.026	0.003	75.161	13,183
Wikipedia Squirrel	5,201	198,493	10	0.348	0.015	26.474	3,148
Twitch DE	9,498	153,138	7	0.047	0.003	20.397	2,545
Twitch EN	7,126	35,324	10	0.042	0.002	20.799	2,545
Twitch ES	4,648	59,382	9	0.084	0.006	19.391	2,545
Twitch FR	6,549	112,666	7	0.054	0.005	19.758	2,545
Twitch PT	1,912	31,299	7	0.131	0.017	19.944	2,545
Twitch RU	4,385	37,304	9	0.049	0.004	20.635	2,545

Facebook page-page dataset

This webgraph is a page-page graph of verified Facebook sites. Nodes represent official Facebook pages while the links are mutual likes between sites. Node features are extracted from the site descriptions that the page owners created to summarize the purpose of the site. This graph was collected through the Facebook Graph API in November 2017 and restricted to pages from 4 categories which are defined by Facebook. These categories are: *politicians*, *governmental organizations*, *television shows* and *companies*. As one can see in Table 3.2 it is a highly clustered graph with a large diameter. The task related to this dataset is multi-class node classification for the 4 site categories.

Github web and machine learning developers dataset

The largest graph used for evaluation is a social network of GitHub developers which we collected from the public API in June 2019. Nodes are developers who have starred at least 10 repositories and edges are mutual follower relationships between them. The vertex features are extracted based on the location, repositories starred, employer and e-mail address. The task related to the graph is binary node classification – one has to predict whether the GitHub user is a web or a machine learning developer. This target feature was derived from the job title of each user. As

the descriptive statistics show in Table 3.2 this is the largest graph that we use for evaluation with the highest sparsity.

Wikipedia datasets

The datasets that we use to perform node level regression are Wikipedia page-page networks collected on three specific topics: chameleons, crocodiles and squirrels. In these networks nodes are articles from the English Wikipedia collected in December 2018, edges are mutual links that exist between pairs of sites. Node features describe the presence of nouns appearing in the articles. For each node we also have the average monthly traffic between October 2017 and November 2018. In the regression tasks used for embedding evaluation the logarithm of average traffic is the target variable. Table 3.2 shows that these networks are heterogeneous in terms of size, density, and clustering.

Twitch datasets

These datasets used for node classification and transfer learning are Twitch user-user networks of gamers who stream in a certain language. Nodes are the users themselves and the links are mutual friendships between them. Vertex features are extracted based on the games played and liked, location and streaming habits. Datasets share the same set of node features, this makes *transfer learning* across networks possible. These social networks were collected in May 2018. The supervised task related to these networks is binary node classification – one has to predict whether a streamer uses explicit language.

Hyperparameter settings

Across all experiments we use the same hyperparameter settings of our own model, competing unsupervised methods and graph neural networks.

Hyperparameter settings of the AE and MUSAE algorithms

In *MUSAE* and *AE* models we have a set of parameters that we use for model evaluation. Our parameter settings listed in Table 3.3 are quite similar to the widely used general settings of random walk sampled implicit factorization machines [66, 147, 148, 159]. Each of our models is augmented with a *Doc2Vec* [129, 130] embedding of node features – this is done such way that the overall dimension is still 128.

Hyperparameter settings of the downstream algorithms

The downstream tasks uses logistic and elastic net regression from *Scikit-learn* [145] for node level classification, regression and link prediction. For the evaluation of every embedding model we use the standard settings of the library except for the regularization and norm mixing parameters. These are described in Table 3.4.

TABLE 3.3: Standard hyperparameter settings of the AE and MUSAE algorithms.

Parameter	Value	Notation
Dimensions	128	d
Walk length	80	l
Number of walks per node	10	p
Number of epochs	5	k
Window size	3	t
Initial learning rate	0.05	α_{max}
Final learning rate	0.025	α_{min}
Negative samples	5	b

TABLE 3.4: Standard hyperparameter settings of the downstream logistic and elastic net regression models that use the embeddings for classification, link prediction and regression.

Parameter	Value	Notation
Regularization coefficient	0.01	λ
Norm mixing parameter	0.5	γ

Hyperparameter settings of competing unsupervised algorithms

Our purpose was a fair evaluation compared to other node embedding procedures. Because of this each we tried to use hyperparameter settings that give similar expressive power to the competing methods with respect to target matrix approximation [66, 147, 148] and number of dimensions.

- *DeepWalk* [147]: We used the hyperparameter settings described in Table 3.3. While the original *DeepWalk* model uses hierarchical softmax to speed up calculations we used a negative sampling based implementation. This way *DeepWalk* can be seen as a special case of *Node2Vec* [66] when the second-order random walks are equivalent to the firs-order walks.

- *LINE*$_2$ [198]: We created 64 dimensional embeddings based on first and second order proximity and concatenated these together for the downstream tasks. Other hyperparameters are taken from the original work.

- *Node2Vec* [66]: Except for the *in-out* and *return* parameters that control the second-order random walk behavior we used the hyperparameter settings described in Table 3.3. These behavior control parameters were tuned with grid search from the $\{4, 2, 1, 0.5, 0.25\}$ set using a train-validation split of $80\% - 20\%$ *within the training set* itself.

- *Walklets* [148]: We used the hyperparameters described in Table 3.3 except for window size. We set a window size of 4 with individual embedding sizes of 32. This way the overall number of dimensions of the representation remained the same.

- The attributed node embedding methods *AANE*, *ASNE*, *BANE*, *TADW*, *TENE* all use the hyperparameters described in the respective papers except for the dimension. We parametrized these methods such way that each of the final embeddings used in the downstream tasks is 128 dimensional.

Hyperparameter settings of competing supervised algorithms

Each model was optimized with the Adam optimizer [91] with the standard moving average parameters and the model implementations are sparsity aware modifications based on PyTorch Geometric [47]. We needed these modifications in order to accommodate the large number of vertex features – see the last column in Table 3.2. Except for the *GAT* model [210] we used ReLU intermediate activation functions [132] with a softmax unit in the final layer for classification. The hyperparameters used for the training and regularization of the neural models are listed in Table 3.5.

TABLE 3.5: Hyperparameter settings used for training the graph neural network baselines.

Parameter	**Value**
Epochs	200
Learning rate	0.01
Dropout	0.5
l_2 Weight regularization	0.001
Depth	2
Filters per layer	32

Except for the *APPNP* model each baseline uses information up to 2-hop neighbourhoods. The model specific settings when we needed to deviate from the basic settings which are listed in Table 3.5 were as follows:

- *Classical GCN* [92]: We used the standard parameter settings described in this section.
- *GraphSAGE* [71]: We utilized a graph convolutional aggregator on the sampled neighbourhoods, samples of 40 nodes per source, and standard settings.
- *GAT* [210]: The negative slope parameter of the leaky ReLU function was 0.2, we applied a single attention head, and used the standard hyperparameter settings.
- *MixHop* [5]: We took advantage of the 0^{th}, 1^{st} and 2^{nd} powers of the normalized adjacency matrix with 32 dimensional convolutional filters for creating the first hidden representations. This was fed to a feed-forward layer to classify the nodes.
- *ClusterGCN* [28]: Just as [28] did, we used the *METIS* procedure [88]. We clustered the graphs into disjoint clusters, and the number of clusters was the same as the number of node classes (e.g. in case of the Facebook page-page network we created 4 clusters). For training we used the earlier described setup.
- *APPNP* [93]: The top level feed-forward layer had 32 hidden neurons, the teleport probability was set as 0.2 and we used 20 steps for approximate personalized pagerank calculation.
- *SGCONV* [222]: We used the 2^{nd} power of the normalized adjacency matrix for training the classifier.

Node attribute classification

In this experiment we take k randomly selected samples per class, and use the attributed node embeddings to train a logistic regression model with l_2 regularization and predict the labels on the remaining vertices. We repeated the above procedure with seeded splits 100 times to obtain robust comparable results [181]. From these we calculated the average of micro averaged F_1 scores to compare our own methods with other unsupervised node embedding procedures. We varied k in order to show the efficacy of the methods – what are the gains when the training set size is increased. These results are plotted in Figure 3.3 for Facebook, Github and Twitch Portugal networks.

Based on these plots it is evident that *MUSAE* and *AE* embeddings have little gains in terms of micro F_1 score when additional data points are added to the training set when k is larger than 12. This implies that our method is data efficient. Moreover, *MUSAE-EGO* and *AE-EGO* have a slight performance advantage, which means that including the nodes in the attributed random walks helps when a small amount of labeled data is available in the downstream task.

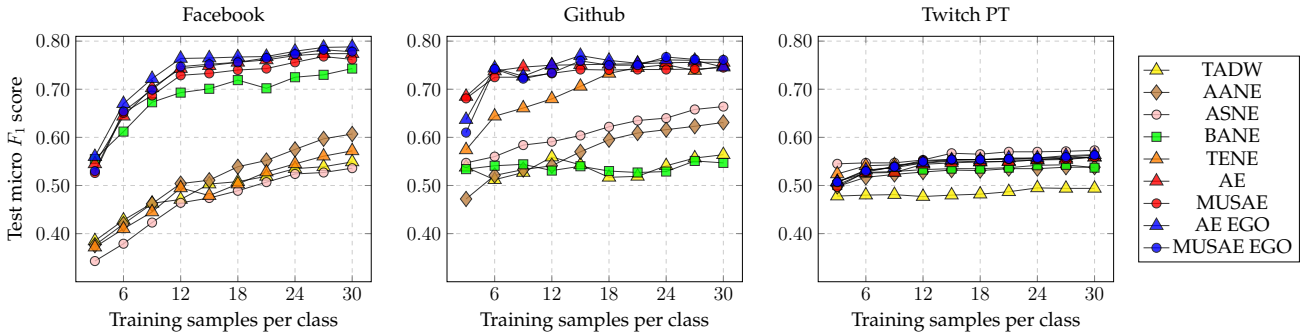


FIGURE 3.3: k-shot node classification performance for varying k, evaluated by micro F_1 scores over 100 seeded train-test splits.

Fixed ratio train-test splits

In this series of experiments we created a 100 seeded train test splits of nodes (80% train - 20% test) and calculated weighted, micro and macro averaged F_1 scores on the test set to compare our methods to various embedding and graph neural network methods. Across procedures the same random seeds were used to obtain the train-test split this way the performances are directly comparable. We attached these results on the Facebook, Github and Twitch Portugal graphs as Table 3.6. In each column red denotes the best performing unsupervised embedding model and blue corresponds to the strongest supervised neural model. We also attached additional supporting results using the same experimental setting with the unsupervised methods on the Cora, Citeseer, and Pubmed graphs as Table 3.7.

In terms of micro F_1 score our strongest method outperforms on the Facebook and GitHub networks the best unsupervised method by 1.01% and 0.47% respectively. On the Twitch Portugal network the relative micro F_1 advantage of *ASNE* over our best method is 1.02%. Supervised node embedding methods outperform our and other unsupervised methods on every dataset for most metrics. In terms of micro F_1 this relative advantage over our best performing model variant is the largest with 4.67% on the Facebook network, and only 0.11% on Twitch Portugal.

TABLE 3.6: Node classification test performance evaluated by weighted, micro and macro F_1 scores calculated from 10 seeded train-test splits. We included standard errors of the scores and used 80% of nodes for training / 20% of nodes for testing. Red numbers denote the best performing node embedding method and blue ones denote the best performing *supervised* graph neural network.

	Datasets								
	Facebook Page-Page			**GitHub WebML**			**Twitch Portugal**		
	Weighted	**Micro**	**Macro**	**Weighted**	**Micro**	**Macro**	**Weighted**	**Micro**	**Macro**
DeepWalk	0.861 ±0.001	0.863 ±0.001	0.848 ±0.001	0.852 ±0.001	0.858 ±0.001	0.801 ±0.002	0.650 ±0.008	0.672 ±0.007	0.594 ±0.009
LINE$_2$	0.874 ±0.001	0.875 ±0.001	0.862 ±0.001	0.852 ±0.001	0.858 ±0.001	0.800 ±0.002	0.636 ±0.006	0.670 ±0.005	0.571 ±0.005
Node2Vec	0.889 ±0.001	0.890 ±0.001	0.880 ±0.001	0.853 ±0.001	0.859 ±0.001	0.802 ±0.001	0.665 ±0.004	0.686 ±0.004	0.612 ±0.004
Walklets	0.886 ±0.001	0.887 ±0.001	0.875 ±0.001	0.854 ±0.001	0.860 ±0.001	0.804 ±0.002	0.652 ±0.006	0.671 ±0.006	0.599 ±0.005
TADW	0.760 ±0.002	0.765 ±0.002	0.740 ±0.003	0.650 ±0.001	0.748 ±0.001	0.528 ±0.007	0.459 ±0.001	0.659 ±0.005	0.406 ±0.003
AANE	0.793 ±0.001	0.796 ±0.001	0.775 ±0.001	0.848 ±0.001	0.856 ±0.001	0.794 ±0.002	0.636 ±0.006	0.661 ±0.006	0.577 ±0.006
ASNE	0.794 ±0.001	0.797 ±0.001	0.776 ±0.001	0.829 ±0.001	0.839 ±0.001	0.766 ±0.002	**0.670 ±0.006**	**0.685 ±0.006**	**0.620 ±0.006**
BANE	0.868 ±0.001	0.868 ±0.001	0.859 ±0.002	0.711 ±0.001	0.762 ±0.001	0.576 ±0.001	0.644 ±0.006	0.664 ±0.006	0.587 ±0.006
TENE	0.724 ±0.002	0.731 ±0.002	0.699 ±0.002	0.842 ±0.001	0.850 ±0.001	0.785 ±0.002	0.613 ±0.005	0.664 ±0.006	0.536 ±0.006
AE	0.887 ±0.001	0.888 ±0.001	0.879 ±0.001	0.858 ±0.001	0.863 ±0.001	0.807 ±0.001	0.653 ±0.005	0.672 ±0.004	0.598 ±0.006
AE-EGO	**0.898 ±0.001**	**0.899 ±0.001**	**0.890 ±0.001**	0.857 ±0.001	0.863 ±0.001	0.807 ±0.002	0.652 ±0.007	0.671 ±0.007	0.599 ±0.009
MUSAE	0.886 ±0.001	0.887 ±0.001	0.877 ±0.001	**0.859 ±0.001**	**0.864 ±0.001**	**0.810 ±0.001**	0.654 ±0.006	0.672 ±0.006	0.600 ±0.007
MUSAE-EGO	0.893 ±0.001	0.894 ±0.001	0.884 ±0.001	**0.859 ±0.001**	**0.864 ±0.001**	**0.810 ±0.001**	0.655 ±0.003	0.671 ±0.002	0.604 ±0.003
GCN	0.931 ±0.001	0.932 ±0.001	0.928 ±0.001	0.859 ±0.001	0.865 ±0.001	0.809 ±0.002	0.650 ±0.013	0.695 ±0.007	0.577 ±0.02
GraphSAGE	0.812 ±0.002	0.814 ±0.002	0.795 ±0.002	0.848 ±0.001	0.854 ±0.001	0.794 ±0.002	0.618 ±0.003	0.631 ±0.004	0.563 ±0.005
GAT	0.918 ±0.001	0.919 ±0.001	0.912 ±0.001	0.856 ±0.001	0.864 ±0.001	0.803 ±0.002	0.648 ±0.008	0.678 ±0.007	0.588 ±0.009
MixHop	**0.940 ±0.001**	**0.941 ±0.002**	**0.937 ±0.001**	0.847 ±0.000	0.85 ±0.001	0.800 ±0.001	0.626 ±0.003	0.630 ±0.004	0.576 ±0.003
ClusterGCN	0.937 ±0.001	0.937 ±0.001	0.934 ±0.001	0.855 ±0.001	0.859 ±0.001	0.807 ±0.001	0.647 ±0.004	0.654 ±0.004	0.602 ±0.005
APPNP	0.938 ±0.001	0.938 ±0.001	0.935 ±0.001	**0.860 ±0.002**	**0.868 ±0.001**	**0.811 ±0.002**	**0.683 ±0.009**	**0.702 ±0.012**	**0.623 ±0.010**
SGCONV	0.832 ±0.002	0.836 ±0.002	0.812 ±0.002	0.816 ±0.001	0.829 ±0.001	0.747 ±0.002	0.652 ±0.003	0.663 ±0.003	0.604 ±0.004

One can make four general observations based on our results (i) multi-scale representations can help with the classification tasks compared to pooled ones; (ii) the addition of the nodes in the ego augmented models to the feature sets does not help the performance when a large amount of labeled training data is available; (iii) based on the standard errors supervised neural models do not necessarily have a significant advantage over unsupervised methods (see the results on the Github and Twitch datasets); (iv) attributed node embedding methods that only consider first-order neighbourhoods have a poor performance.

Node attribute regression

We compare the ability of our attributed embeddings against those of other unsupervised methods at predicting real valued node attributes. For each model, we train embeddings on an unsupervised basis and learn regression parameters of an elastic net to predict the log average web traffic (attribute) of each page (node) of the Wikipedia datasets (created for this task). Table 3.8 reports average test R^2 (explained variance) and standard error over 100 seeded (80% train - 20% test) splits. This shows that: (i) our attributed embeddings tend to outperform all other methods at the regression task; (ii) multi-scale methods (e.g. *MUSAE*) tend to outperform pooled

methods (e.g. *AE*); (iii) the additional network information of the *EGO* models appears beneficial, but only in the multi-scale case. Overall, *MUSAE-EGO* outperforms the best baseline for each dataset by 2-10%.

TABLE 3.7: Node classification test performance evaluated by weighted, micro and macro F_1 scores calculated from 10 seeded train-test splits. We included standard errors of the scores and used 80% of nodes for training / 20% of nodes for testing. Red numbers denote the best performing node embedding method.

	Datasets								
	Cora			**Citeseer**			**Pubmed**		
	Weighted	**Micro**	**Macro**	**Weighted**	**Micro**	**Macro**	**Weighted**	**Micro**	**Macro**
DeepWalk	0.832 ±0.003	0.833 ±0.004	0.823 ±0.004	0.597 ±0.007	0.603 ±0.007	0.560 ±0.006	0.801 ±0.001	0.802 ±0.001	0.789 ±0.002
LINE$_2$	0.775 ±0.004	0.777 ±0.004	0.768 ±0.005	0.529 ±0.006	0.542 ±0.006	0.486 ±0.005	0.798 ±0.001	0.799 ±0.001	0.785 ±0.001
Node2Vec	0.840 ±0.003	0.840 ±0.003	0.826 ±0.003	0.616 ±0.005	0.622 ±0.005	0.581 ±0.005	0.809 ±0.002	0.810 ±0.002	0.797 ±0.002
Walklets	0.843 ±0.003	0.843 ±0.003	0.827 ±0.003	0.624 ±0.005	0.630 ±0.006	0.590 ±0.005	0.815 ±0.001	0.815 ±0.001	0.804 ±0.002
TADW	0.819 ±0.004	0.819 ±0.004	0.804 ±0.005	0.725 ±0.004	0.734 ±0.004	0.685 ±0.004	0.862 ±0.002	0.862 ±0.002	0.863 ±0.002
AANE	0.793 ±0.006	0.793 ±0.006	0.777 ±0.006	0.728 ±0.005	0.733 ±0.004	0.693 ±0.005	**0.867 ±0.001**	**0.867 ±0.001**	**0.867 ±0.002**
ASNE	0.831 ±0.003	0.830 ±0.003	0.812 ±0.004	0.713 ±0.004	0.718 ±0.004	0.677 ±0.004	0.846 ±0.002	0.846 ±0.002	0.843 ±0.002
BANE	0.807 ±0.005	0.807 ±0.005	0.787 ±0.005	0.707 ±0.003	0.713 ±0.003	0.670 ±0.004	0.823 ±0.002	0.823 ±0.002	0.822 ±0.002
TENE	0.829 ±0.005	0.829 ±0.005	0.815 ±0.004	0.664 ±0.004	0.681 ±0.003	0.611 ±0.002	0.842 ±0.001	0.842 ±0.001	0.843 ±0.002
AE	0.835 ±0.005	0.835 ±0.005	0.815 ±0.006	0.730 ±0.005	0.739 ±0.005	0.688 ±0.006	0.839 ±0.002	0.839 ±0.002	0.840 ±0.002
AE-EGO	0.835 ±0.005	0.835 ±0.006	0.816 ±0.005	0.729 ±0.004	0.739 ±0.005	0.690 ±0.007	0.840 ±0.002	0.840 ±0.003	0.839 ±0.002
MUSAE	0.848 ±0.004	0.848 ±0.004	0.832 ±0.005	**0.737 ± 0.004**	**0.742 ±0.004**	**0.706 ±0.004**	0.853 ±0.001	0.853 ±0.001	0.854 ±0.002
MUSAE-EGO	**0.849 ±0.004**	**0.849 ±0.004**	**0.833 ±0.004**	0.736 ±0.004	0.741 ±0.004	0.706 ±0.004	0.850 ±0.002	0.851 ±0.002	0.850 ±0.002

TABLE 3.8: Node attribute regression: average R^2 statistics and standard error for predicting website traffic (best results in bold).

	Wiki Chameleons	**Wiki Crocodiles**	**Wiki Squirrels**
DeepWalk	.375 ± .004	.553 ± .002	.170 ± .001
LINE$_2$	.381 ± .003	.586 ± .001	.232 ± .002
Node2Vec	.414 ± .003	.574 ± .001	.174 ± .002
Walklets	.426 ± .003	.625 ± .001	.249 ± .002
TADW	.527 ± .003	.636 ± .001	.271 ± .002
AANE	.598 ± .007	.732 ± .002	.287 ± .002
ASNE	.440 ± .009	.572 ± .003	.229 ± .005
BANE	.464 ± .003	.617 ± .001	.168 ± .002
TENE	.494 ± .020	.701 ± .003	.321 ± .007
AE	.642 ± .006	.743 ± .003	.291 ± .006
AE-EGO	.644 ± .009	.732 ± .002	.283 ± .006
MUSAE	**.658** ± .004	.736 ± .003	.338 ± .007
MUSAE-EGO	.653 ± .011	**.747** ± .003	**.354** ± .009

Link prediction

A common downstream task for node embeddings is to predict edges in a graph, or *link prediction*. Intuitively, whether attribute-level information helps to predict edges seems dataset/attribute dependent and we do not necessarily expect attributed embeddings to outperform those learned by algorithms focused specifically on network structure (e.g. *DeepWalk*, *Node2Vec*). We evaluate link

prediction performance similarly to [66], by: (i) randomly removing 50% of edges (maintaining graph connectivity); (ii) learning node embeddings on the attenuated graph; (iii) applying each of four binary operators[1] to generate d-dimensional "edge representations" from pairs of node embeddings for both removed edges (positive samples) and randomly selected nodes (negative samples); and (iv) with an 80% train - 20% test split, training a logistic regression classifier to distinguish positive and negative samples.node

TABLE 3.9: Link prediction results evaluated by average AUC scores on the test set using attributed node embedding techniques and regularized logistic regression. We created a 100 seeded splits (80% training - 20% test). Standard errors of the AUC score are included below. Red denotes the best performing embedding model on a given dataset considering both the neighbourhood based and neighbourhood based embedding techniques. We used 4 different element-wise operators to create edge features.

		Datasets					
Operator	**Method**	**Facebook Page-Page**	**GitHub Web-ML**	**Twitch Spain**	**Twitch Germany**	**Wikipedia Chameleons**	**Wikipedia Crocodiles**
Average	**DeepWalk**	0.526 ±0.006	0.550 ±0.004	0.568 ±0.009	0.575 ±0.005	0.635 ±0.002	0.661 ±0.007
	LINE$_2$	0.517 ±0.007	0.551 ±0.003	0.540 ±0.011	0.544 ±0.004	0.627 ±0.001	0.708 ±0.004
	Node2Vec	0.534 ±0.005	0.573 ±0.003	0.575 ±0.012	0.584 ±0.006	0.641 ±0.009	0.669 ±0.007
	Walklets	0.518 ±0.008	0.552 ±0.003	0.541 ±0.006	0.545 ±0.004	0.635 ±0.015	0.716 ±0.003
Hadamard	**DeepWalk**	0.981 ±0.001	0.799 ±0.001	0.781 ±0.002	0.750 ±0.003	0.974 ±0.002	0.966 ±0.001
	LINE$_2$	0.979 ±0.001	0.899 ±0.001	0.843 ±0.003	0.755 ±0.001	0.939 ±0.003	0.938 ±0.001
	Node2Vec	0.982 ±0.001	0.822 ±0.001	0.810 ±0.005	0.780 ±0.003	**0.979 ±0.001**	0.973 ±0.001
	Walklets	**0.984 ±0.001**	0.925 ±0.001	0.873 ±0.003	0.819 ±0.001	0.966 ±0.004	**0.980 ±0.001**
l_1 Norm	**DeepWalk**	0.921 ±0.001	0.658 ±0.001	0.723 ±0.004	0.711 ±0.002	0.950 ±0.002	0.896 ±0.001
	LINE$_2$	0.924 ±0.001	0.913 ±0.002	0.882 ±0.002	0.855 ±0.001	0.922 ±0.003	0.930 ±0.001
	Node2Vec	0.928 ±0.001	0.725 ±0.001	0.761 ±0.004	0.745 ±0.001	0.953 ±0.003	0.913 ±0.001
	Walklets	0.980 ±0.001	0.932 ±0.001	**0.898 ±0.002**	0.870 ±0.001	0.961 ±0.004	0.976 ±0.001
l_2 Norm	**DeepWalk**	0.922 ±0.001	0.663 ±0.001	0.731 ±0.004	0.717 ±0.002	0.951 ±0.002	0.899 ±0.002
	LINE$_2$	0.924 ±0.001	0.910 ±0.001	0.880 ±0.002	0.855 ±0.001	0.925 ±0.003	0.936 ±0.001
	Node2Vec	0.929 ±0.002	0.731 ±0.001	0.768 ±0.007	0.750 ±0.001	0.954 ±0.002	0.920 ±0.001
	Walklets	0.981 ±0.001	0.930 ±0.001	0.897 ±0.002	0.870 ±0.002	0.960 ±0.005	0.978 ±0.001

Tables 3.10 and 3.9 show the average AUC for each model over 100 seeded splits. The results show: (i) our attributed embeddings perform comparably to other attributed models, with *MUSAE-EGO* similar to the best performing neighbourhood-preserving algorithm (*Walklets*) and attributed embeddings (*TADW*); (ii) our multi-scale embeddings outperform pooled embeddings; and (iii) the encoded network features of *EGO* embeddings are beneficial.

Overall, no algorithm dominates the link prediction task. Importantly, we see that the additional node level information our attributed embeddings learn does not detract from their link prediction performance, which matches that of algorithms that exclusively learn network structure.

Transfer learning

Neighbourhood based methods such as *DeepWalk* [147] learn node embeddings, typically with no mechanism to relate the embeddings of distinct graphs. Thus a regression model learned to

[1] As in [66] element-wise: average, product (Hadamard), absolute difference (l_1), squared difference (l_2).

TABLE 3.10: Link prediction results evaluated by average AUC scores on the test set using attributed node embedding techniques and regularized logistic regression. We created a 100 seeded splits (80% training - 20% test). Standard errors of the AUC score are included below. Red denotes the best performing embedding model on a given dataset considering both the neighbourhood based and attributed techniques. We used 4 different element-wise operators to create edge features.

Operator	Method	Datasets					
		Facebook Page-Page	GitHub Web-ML	Twitch Spain	Twitch Germany	Wikipedia Chameleons	Wikipedia Crocodiles
Average	TADW	0.517 ±0.004	0.553 ±0.004	0.541 ±0.007	0.556 ±0.008	0.573 ±0.012	0.625 ±0.006
	AANE	0.523 ±0.003	0.539 ±0.003	0.536 ±0.001	0.554 ±0.004	0.552 ±0.013	0.577 ±0.005
	ASNE	0.547 ±0.005	0.596 ±0.002	0.562 ±0.007	0.579 ±0.005	0.650 ±0.005	0.723 ±0.004
	BANE	0.625 ±0.003	0.630 ±0.002	0.616 ±0.001	0.634 ±0.003	0.617 ±0.011	0.671 ±0.003
	TENE	0.547 ±0.004	0.515 ±0.004	0.532 ±0.011	0.555 ±0.006	0.555 ±0.011	0.644 ±0.003
	AE	0.572 ±0.006	0.719 ±0.002	0.679 ±0.006	0.723 ±0.003	0.669 ±0.007	0.834 ±0.002
	MUSAE	0.642 ±0.003	0.780 ±0.001	0.733 ±0.004	0.771 ±0.002	0.810 ±0.006	0.899 ±0.001
	AE-EGO	0.514 ±0.007	0.546 ±0.005	0.523 ±0.011	0.545 ±0.007	0.563 ±0.011	0.660 ±0.006
	MUSAE-EGO	0.511 ±0.005	0.542 ±0.003	0.533 ±0.005	0.546 ±0.008	0.622 ±0.008	0.699 ±0.003
Hadamard	TADW	0.973 ±0.001	0.915 ±0.001	0.886 ±0.003	**0.884 ±0.001**	0.964 ±0.002	0.967 ±0.001
	AANE	0.911 ±0.002	0.772 ±0.002	0.833 ±0.003	0.811 ±0.002	0.917 ±0.005	0.892 ±0.005
	ASNE	0.973 ±0.001	0.912 ±0.001	0.883 ±0.003	0.866 ±0.002	0.945 ±0.005	0.940 ±0.001
	BANE	0.653 ±0.002	0.664 ±0.003	0.659 ±0.009	0.816 ±0.002	0.578 ±0.014	0.738 ±0.002
	TENE	0.735 ±0.012	0.878 ±0.009	0.722 ±0.007	0.748 ±0.003	0.883 ±0.003	0.872 ±0.015
	AE	0.926 ±0.001	0.814 ±0.002	0.743 ±0.003	0.702 ±0.003	0.939 ±0.002	0.949 ±0.001
	MUSAE	0.945 ±0.001	0.917 ±0.002	0.871 ±0.005	0.863 ±0.002	0.950 ±0.005	0.968 ±0.001
	AE-EGO	0.928 ±0.001	0.786 ±0.002	0.727 ±0.006	0.687 ±0.003	0.935 ±0.002	0.939 ±0.001
	MUSAE-EGO	0.938 ±0.001	0.911 ±0.002	0.881 ±0.003	0.859 ±0.001	0.952 ±0.007	0.969 ±0.001
l_1 Norm	TADW	0.971 ±0.001	0.909 ±0.002	0.882 ±0.003	0.881 ±0.001	0.959 ±0.002	0.962 ±0.001
	AANE	0.866 ±0.002	0.720 ±0.001	0.771 ±0.004	0.768 ±0.001	0.944 ±0.002	0.913 ±0.001
	ASNE	0.815 ±0.002	0.866 ±0.001	0.836 ±0.002	0.849 ±0.001	0.869 ±0.001	0.874 ±0.001
	BANE	0.653 ±0.002	0.664 ±0.003	0.658 ±0.009	0.816 ±0.002	0.578 ±0.014	0.74 ±0.002
	TENE	0.940 ±0.001	**0.942 ±0.001**	0.857 ±0.004	0.837 ±0.001	0.945 ±0.003	0.927 ±0.001
	AE	0.968 ±0.001	0.889 ±0.001	0.871 ±0.001	0.870 ±0.002	0.955 ±0.002	0.952 ±0.002
	MUSAE	0.973 ±0.001	0.908 ±0.001	0.885 ±0.002	0.879 ±0.002	0.956 ±0.003	0.967 ±0.001
	AE-EGO	0.973 ±0.001	0.891 ±0.001	0.872 ±0.002	0.872 ±0.002	0.953 ±0.002	0.955 ±0.001
	MUSAE-EGO	0.977 ±0.001	0.911 ±0.001	0.891 ±0.002	**0.884 ±0.002**	0.955 ±0.003	0.963 ±0.001
l_2 Norm	TADW	0.972 ±0.001	0.913 ±0.001	0.883 ±0.003	0.879 ±0.001	0.961 ±0.002	0.964 ±0.001
	AANE	0.877 ±0.001	0.732 ±0.001	0.779 ±0.003	0.774 ±0.002	0.941 ±0.005	0.901 ±0.002
	ASNE	0.806 ±0.004	0.872 ±0.001	0.839 ±0.003	0.852 ±0.002	0.875 ±0.006	0.880 ±0.001
	BANE	0.653 ±0.002	0.664 ±0.003	0.659 ±0.009	0.816 ±0.0002	0.578 ±0.014	0.738 ±0.002
	TENE	0.893 ±0.001	0.884 ±0.016	0.826 ±0.009	0.797 ±0.005	0.930 ±0.003	0.863 ±0.013
	AE	0.968 ±0.001	0.881 ±0.001	0.872 ±0.001	0.867 ±0.002	0.954 ±0.003	0.953 ±0.001
	MUSAE	0.973 ±0.001	0.905 ±0.001	0.884 ±0.002	0.877 ±0.002	0.952 ±0.005	0.965 ±0.001
	AE-EGO	0.973 ±0.001	0.884 ±0.001	0.873 ±0.001	0.871 ±0.002	0.952 ±0.003	0.956 ±0.001
	MUSAE-EGO	0.977 ±0.001	0.907 ±0.001	0.891 ±0.003	0.881 ±0.002	0.951 ±0.006	0.961 ±0.001

predict attributes from node embeddings for one graph (as in any of the previous tasks) will not generally perform well given the node embeddings of another graph. The attributed models *MUSAE* and *AE*, however, learn representations of both nodes and features, in principle enabling information to be shared between graphs with common features. That is, if node and feature embeddings and a regression model are learned for a source graph $\mathcal{S}$ with feature set $\mathbb{F}$, then node embeddings can be learned for a target graph $\mathcal{T}$ (with the same features $\mathbb{F}$) that *fit* with the feature embeddings learned for $\mathcal{S}$ and so also its regression model. This amounts to *transfer learning*, or zero-shot attribute prediction on the target graph. The information shared between graphs allows nodes of $\mathcal{T}$ to be embedded into the same latent space as those of $\mathcal{S}$. Since other unsupervised embedding methods do not learn explicit feature embeddings, they do not enable such transfer learning.

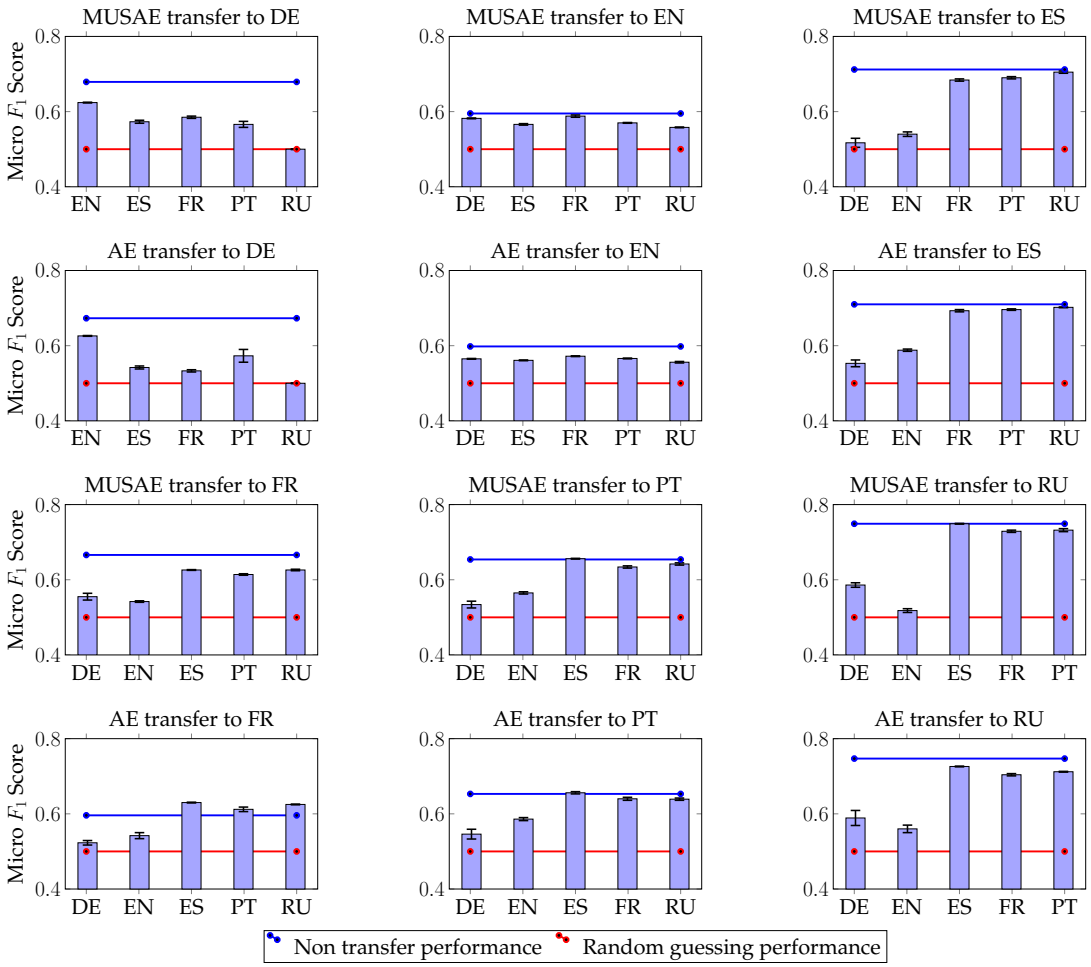


FIGURE 3.4: Mean micro F_1 scores and standard errors calculated from 10 transfer learning runs with *MUSAE* and *AE* on the Twitch graphs. The blue reference line denotes the test performance on the target dataset in a non transfer learning scenario (standard hyperparameter settings and split ratio). The red reference line denotes the performance of random guesses.

The Twitch datasets contain disjoint sets of nodes but a common set of features $\mathbb{F}$. To perform transfer learning, we: (i) learn node and attribute embeddings for a source graph; (ii) train a regression model to predict a test attribute $f \notin \mathbb{F}$; (iii) re-run the embedding algorithm on a target graph but with *feature embedding parameters fixed* (as learned in step (i)); and (iv) use the regression model to predict attribute f for target graph nodes. Figure 3.4 shows micro F_1 scores and standard

error (over 10 runs) for the Twitch datasets as target and each other dataset as source graphs. The results confirm that *MUSAE* and *AE* learn feature embeddings that transfer between graphs with common features. Specifically, we see that performance always beats random guessing (red line) and that in some cases compares closely to re-training the feature embeddings and regression on the target graph (blue line).

Scalability

We demonstrate the efficiency of our algorithms with synthetic data, varying the number of nodes and features per node. Figure 8.7 shows the mean runtime for sets of 100 experiments on Erdos-Renyi graphs with the number of nodes as indicated, 8 edges per node and the indicated number of unique features per node uniformly selected from a set of 2^{11}. The hyperparameters are in Table 3.3 except we perform a single epoch with asynchronous gradient descent.

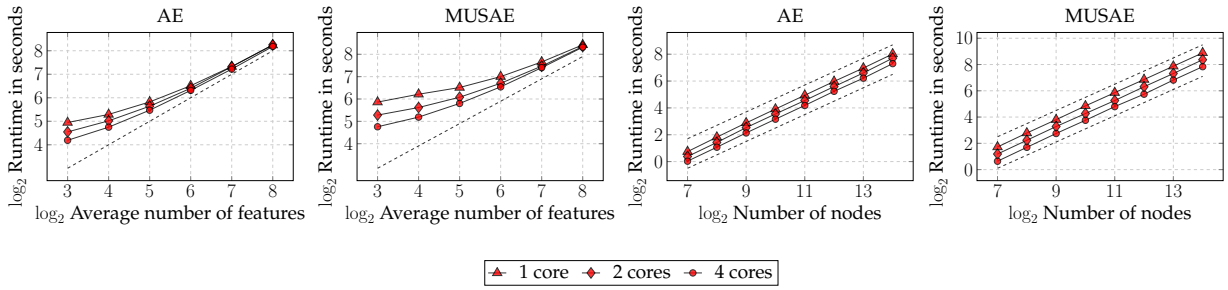


FIGURE 3.5: Training time as a function of average feature count and number of nodes. Dashed lines are linear runtime references.

For both *AE* and *MUSAE*, we see that (i) runtime is linear in the number of nodes and in the average number of features per node; and (ii) increasing the number of cores does not decrease runtime as the number of unique features per vertex approaches the size of the feature set. Observation (i) agrees with our complexity analysis in Section 3.4. In addition, if one interpolates linearly from these results it comes that a network with 1 million nodes, 8 edges per node, 8 unique features per node can be embedded with *MUSAE* on commodity hardware in less than 5 hours. This interpolation assumes that the standard parameter settings and 4 cores were used for optimization.

3.6 Discussion and conclusions

We present attributed node embedding algorithms that learn local feature information on a pooled (*AE*) and multi-scale (*MUSAE*) basis. We augment these models to also explicitly learn local network information (*AE-EGO, MUAE-EGO*), blending the benefits of attributed and neighbourhood-preserving algorithms. Results on a range of datasets show that distinguishing neighbourhood information at different scales (*MUSAE* models) is typically beneficial for the downstream tasks of node attribute and link prediction; and that the supplementary network information encoded by *EGO* models typically improves performance further, particularly for link prediction. Combining both, *MUSAE-EGO* outperforms other unsupervised attributed algorithms at predicting attributes and matches performance of neighbourhood-preserving embeddings in link prediction.

Furthermore, by learning distinct node and feature embeddings, the *AE* and *MUSAE* algorithms enable transfer learning between graphs with common features, as we demonstrate on real datasets.

We derive explicit pointwise mutual information matrices that each of our algorithms implicitly factorise to enable future interpretability and potential analysis of the information encoded (e.g. as possible for *Word2Vec* [12]) and its use in downstream tasks. We see also that two widely used neighbourhood-preserving algorithms [147, 148] are special cases of our models. All of our algorithms are efficient and scale linearly with the numbers of nodes and features per node.

Chapter 4

Characteristic Functions on Graphs

In this chapter, we propose a flexible notion of characteristic functions defined on graph vertices to describe the distribution of vertex features at multiple scales. We introduce *FEATHER*, a computationally efficient algorithm to calculate a specific variant of these characteristic functions where the probability weights of the characteristic function are defined as the transition probabilities of random walks. We argue that features extracted by this procedure are useful for node level machine learning tasks. We discuss the pooling of these node representations, resulting in compact descriptors of graphs that can serve as features for graph classification algorithms. We analytically prove that *FEATHER* describes isomorphic graphs with the same representation and exhibits robustness to data corruption. Using the node feature characteristic functions we define parametric models where evaluation points of the functions are learned parameters of supervised classifiers. Experiments on real world large datasets show that our proposed algorithm creates high quality representations, performs transfer learning efficiently, exhibits robustness to hyperparameter changes and scales linearly with the input size.

4.1 Introduction

Recent works in network mining have focused on characterizing node neighbourhoods. Features of a neighbourhood serve as valuable inputs to downstream machine learning tasks such as node classification, link prediction and community detection [22, 66, 75, 147, 227]. In social networks, the importance of neighbourhood features arises from the property of homophily (correlation of network connections with similarity of attributes), and social neighbours have been shown to influence habits and attributes of individuals [149]. Attributes of a neighbourhood is found to be important in other types of networks as well. Several network mining methods have used aggregate features from several degrees of neighbourhoods for network analysis and embedding [75, 227, 231, 244].

Neighbourhood features can be complex to interpret. Network datasets can incorporate multiple attributes, with varied distributions that influence the characteristics of a node and the network. Attributes such as income, wealth or number of page accesses can have an unbounded domain, with unknown distributions. Simple linear aggregates [75, 227, 231, 244] such as the mean values do not represent this diverse information.

We use characteristic functions [33] as a rigorous but versatile way of utilising diverse neighborhood information. A unique characteristic function always exists irrespective of the nature

of the distribution, and characteristic functions can be meaningfully composed across multiple nodes and even multiple attributes. These features let us represent and compare different neighborhoods in a unified framework.

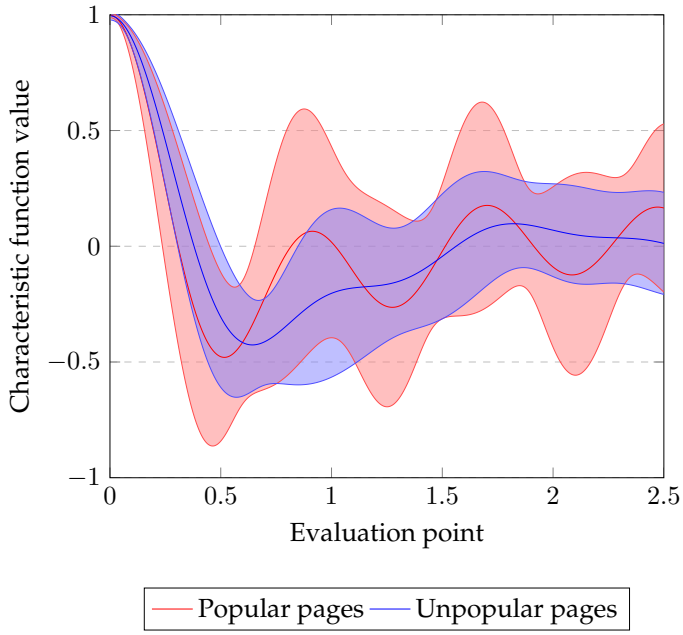


FIGURE 4.1: The real part of class dependent mean characteristic functions with standard deviations around the mean for the log transformed degree on the Wikipedia Crocodiles dataset.

Figure 4.1 shows the distribution of node level characteristic function values on the Wikipedia Crocodiles web graph [165]. In this dataset nodes are webpages which have two types of labels – popular and unpopular. With log transformed degree centrality as the vertex attribute, we conditioned the distributions on the class memberships. We plotted the mean of the distribution at each evaluation point with the standard deviation around the mean. One can easily observe that the value of the characteristic function is discriminative with respect to the class membership of nodes. Our experimental results about node classification in Subsection 4.4 validates this observation about characteristic functions for the Wikipedia dataset and various other social networks.

Present work. We propose complex valued characteristic functions [33] for representation of neighbourhood feature distributions. Characteristic functions are analogues of Fourier Transforms defined for probability distributions. We show that these continuous functions can be evaluated suitably at discrete points to obtain effective characterisation of neighborhoods and describe an approach to learn the appropriate evaluation points for a given task.

The correlation of attributes are known to decrease with the decrease in tie strength, and with increasing distance between nodes [29]. We use a random-walk based tie strength definition, where tie strength at the scale r between a source and target node pair is the probability of an r length random walk from the source node ending at the target. We define the *r-scale random walk weighted characteristic function* as the characteristic function weighted by these tie strengths. We propose *FEATHER* an algorithm to efficiently evaluate this function for multiple features on a graph.

We theoretically prove that graphs which are isomorphic have the same pooled characteristic function when the mean is used for pooling node characteristic functions. We argue that the *FEATHER* algorithm can be interpreted as the forward pass of a parametric statistical model (e.g. logistic regression or a feed-forward neural network). Exploiting this we define the r-scale random walk weighted characteristic function based softmax regression and graph neural network models (respectively named *FEATHER-L* and *FEATHER-N*).

We evaluate *FEATHER* model variants on two machine learning tasks – node and graph classification. Using data from various real world social networks (Facebook, Deezer, Twitch) and web graphs (Wikipedia, GitHub), we compare the performance of *FEATHER* with graph neural networks, neighbourhood preserving and attributed node embedding techniques. Our experiments illustrate that *FEATHER* outperforms comparable unsupervised methods by as much as 4.6% on node labeling and 12.0% on graph classification tasks in terms of test AUC score. The proposed procedures show competitive transfer learning capabilities on social networks and the supervised *FEATHER* variants show a considerable advantage over the unsupervised model, especially when the number of evaluation points is limited. Runtime experiments establish that *FEATHER* scales linearly with the input size.

Main contributions. To summarize, this chapter makes the following contributions:

1. We introduce a generalization of characteristic functions to node neighbourhoods, where the probability weights of the characteristic function are defined by tie strength.
2. We discuss a specific instance of these functions – the r-scale random walk weighted characteristic function. We propose *FEATHER*, an algorithm that calculates these characteristic functions efficiently to create Euclidean node embeddings.
3. We demonstrate that this function can be applied simultaneously to multiple features.
4. We show that the r-scale random walk weighted characteristic function calculated by *FEATHER* can serve as the building block for an end-to-end differentiable parametric classifier.
5. We experimentally assess the behaviour of *FEATHER* on real world node and graph classification tasks.

The remainder of this work has the following structure. In Section 4.2 we overview the relevant literature on node embedding techniques, graph kernels and neural networks. We introduce characteristic functions defined on graph vertices in Section 4.3 and discuss using them as building blocks in parametric statistical models. We empirically evaluate *FEATHER* on various node and graph classification tasks, transfer learning problems, and test its sensitivity to hyperparameter changes in Section 4.4. The chapter concludes with Section 4.5 where we discuss our main findings and point out directions for future work. The newly introduced node classification datasets and a Python reference implementation of *FEATHER* is available at `https://github.com/benedekrozemberczki/FEATHER`.

4.2 Related work

Characteristic functions have previously been used in relation to heat diffusion wavelets [41], which defined the functions for uniform ties strengths and restricted features types.

A general overview

Node embedding techniques map nodes of a graph into Euclidean spaces where the similarity of vertices is approximately preserved – each node has a vector representation. Various forms of embeddings have been studied recently, *Neighbourhood preserving* node embeddings are learned by explicitly [22, 139, 154] or implicitly decomposing [147, 148, 198] a proximity matrix of the graph. *Attributed node embedding* techniques [116, 227, 231, 234, 244] augment the neighbourhood information with generic node attributes (e.g. the user's age in a social network) and nodes sharing metadata are closer in the learned embedding space. *Structural embeddings* [11, 66, 75] create vector representations of nodes which retain the similarity of structural roles and equivalences of nodes. The non-supervised *FEATHER* algorithm which we propose can be seen as a node embedding technique. We create a mapping of nodes to the Euclidean space, simply by evaluating the characteristic function for metadata based generic, neighbourhood and structural node attributes. With the appropriate tie strength definition we are able to hybridize all three types of information with our embedding.

Whole graph embedding and *statistical graph fingerprinting* techniques map graphs into Euclidean spaces where graphs with similar structures and subpatterns are located in close proximity – each graph obtains a vector representation. Whole graph embedding procedures [27, 134] achieve this by decomposing graph – structural feature matrices to learn an embedding. Statistical graph fingerprinting techniques [34, 51, 202, 213] extract information from the graph Laplacian eigenvalues, or using the graph scattering transform without learning. Node level *FEATHER* representations can be pooled by permutation invariant functions to output condensed graph fingerprints which is in principle similar to statistical graph fingerprinting. These statistical fingerprints are related to *graph kernels* as the pooled characteristic functions can serve as inputs for appropriate kernel functions. This way the similarity of graphs is not compared based on the presence of sparsely appearing common random walks [53], cyclic patterns [78] or subtree patterns [182], but via the kernel defined on pairs of dense pooled graph characteristic function representations.

There is also a parallel between *FEATHER* and the forward pass of *graph neural network layers* [71, 92]. During the *FEATHER* function evaluation using the tie strength weights and vertex features we create multi-scale descriptors of the feature distributions which are parameterized by the evaluation points. This can be seen as the forward pass of a multi-scale graph neural network [5, 93] which is able to describe vertex features at multiple scales. Using this we essentially define end-to-end differentiable parametric statistical models where the modulation of evaluation points (the relaxation of fixed evaluation points) can help with the downstream learning task at hand. Compared to traditional graph neural networks [5, 28, 71, 92, 93, 222], which only calculate the first moments of the node feature distributions, *FEATHER* models give summaries of node feature distributions with trainable characteristic function evaluation points.

A desiderata based comparison

A node representation learning technique must possess certain essential properties with respect to the representation itself, the learning process and the memory requirements and runtime. We created a summative description of node embedding and graph neural network techniques in Table 4.1 with the respective space and time complexities.

TABLE 4.1: A summary of existing node embedding techniques (proximity preserving and attributed) and graph neural networks with respect to having (✔) and missing (✘) desired properties. The time an space complexities are reported as a function of vertex and edge counts ($|V|$ and $|E|$), unique feature count $|F|$, average node feature count $|f|$ and embedding dimensions d.

	Generic Features	Multi Scale	Error Bound	Moment Encoding	Supervision Agnostic	Inductive	Space Complexity	Time Complexity
DeepWalk [147]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert\ d)$	$\mathcal{O}(\lvert V\rvert\ d)$
LINE$_2$ [198]	✘	✔	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert\ d)$	$\mathcal{O}(\lvert V\rvert\ d)$
Node2Vec [66]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert^3)$	$\mathcal{O}(\lvert V\rvert\ d)$
Walklets [148]	✘	✔	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert\ d)$	$\mathcal{O}(\lvert V\rvert\ d)$
GraRep [22]	✘	✔	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert^2)$	$\mathcal{O}(\lvert V\rvert^3\ d)$
NetMF [154]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert^2)$	$\mathcal{O}(\lvert V\rvert^3\ d)$
BoostNE [85]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert^2)$	$\mathcal{O}(\lvert V\rvert^3\ d)$
Diff2Vec [170]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert\ d)$	$\mathcal{O}(\lvert V\rvert\ d)$
HOPE [139]	✘	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert\ d)$	$\mathcal{O}(\lvert V\rvert\ d)$
TADW [227]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert + \lvert F\rvert)\ d)$	$\mathcal{O}(\lvert V\rvert^2\ \lvert F\rvert\ d)$
ASNE [116]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert + \lvert F\rvert)\ d)$	$\mathcal{O}(\lvert E\rvert\ \lvert f\rvert\ d)$
AANE [81]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert^2\ \lvert f\rvert\ d)$	$\mathcal{O}(\lvert V\rvert^2\ \lvert f\rvert\ d)$
BANE [231]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert^2\ \lvert f\rvert\ d)$	$\mathcal{O}(\lvert V\rvert^3\ \lvert f\rvert\ d)$
TENE [234]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert + \lvert F\rvert)\ d)$	$\mathcal{O}(\lvert E\rvert\ \lvert f\rvert\ d)$
SINE [244]	✔	✘	✘	✘	✘	✘	$\mathcal{O}(\lvert V\rvert + \lvert F\rvert)\ d)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
MUSAE [165]	✔	✔	✘	✘	✘	✔	$\mathcal{O}((\lvert V\rvert + \lvert F\rvert)\ d)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
GCN [92]	✔	✘	✘	✘	✘	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
GAT [210]	✔	✘	✘	✘	✘	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
GraphSage [71]	✔	✘	✘	✘	✔	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
ClusterGCN [28]	✔	✘	✘	✘	✘	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
APPNP [93]	✔	✘	✘	✘	✘	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
MixHop [5]	✔	✔	✘	✘	✘	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
SGConv [222]	✔	✘	✘	✘	✘	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$
FEATHER (ours)	✔	✔	✔	✔	✔	✔	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert)$	$\mathcal{O}(\lvert V\rvert\ \lvert f\rvert\ d)$

- **Generic features:** A node embedding algorithm should be able to incorporate information from generic node attributes. For example in a social network information about the users' age could be encoded in the embedding.
- **Multi scale:** Information from distinct proximities is encoded by separate blocks of features in the representation. The *FEATHER* variants and other multi-scale methods such as *LINE* [198], *Walklets* [148], *GraRep* [22], *MUSAE* [165], and *MixHop* [5] all use the same truncated random walk based multi-scale proximity definition.
- **Error bound:** When a node feature is corrupted the effect of corruption on the learned representation should be bounded. Practically an unbounded change of the node representation

could have detrimental effects on downstream learning. The competing unsupervised node embedding and supervised graph neural networks all fail to achieve this desired property.

- **Moment encoding:** Given a feature distribution in a neighbourhood the learning algorithm should be able to describe the higher order moments of the distribution not just the mean. The fact that *FEATHER* is able to do this is one of the main contributions in our work.
- **Supervision agnostic:** A node representation learning algorithm is supervision agnostic if it is able to learn features in an unsupervised way (embeddings) without supervision (node level target variable). The basic *FEATHER* algorithm which we propose is completely unsupervised, while *FEATHER-N* and *FEATHER-L* require supervision.
- **Inductive:** The node embedding function is able to map unseen nodes to the embedding space which are not necessarily connected to the nodes seen at the training time. This is an extremely important property in most practical use cases. Besides the supervised and unsupervised *FEATHER* variants all of the supervised representation learning methods [5, 28, 71, 92, 93, 210, 222] and *MUSAE* [165] are able to do this as the node embeddings are learned based on the feature context in the neighbourhood.

4.3 Characteristic functions on graphs

In this section we introduce the idea of characteristic functions defined on attributed graphs. Specifically, we discuss the idea of describing node feature distributions in a neighbourhood with characteristic functions. We propose a specific instance of these functions, the *r-scale random walk weighted characteristic function* and we describe an algorithm to calculate this function for all nodes in linear time. We prove the robustness of these functions and how they represent isomorphic graphs when node level functions are pooled. Finally, we discuss how characteristic functions can serve as building blocks for parametric statistical models.

Node feature distribution characterization

We assume that we have an attributed and undirected graph $G = (V, E)$. Nodes of G have a feature described by the random variable X, specifically defined as the feature vector $\mathbf{x} \in \mathbb{R}^{|V|}$, where $\mathbf{x}_v$ is the feature value for node $v \in V$. We are interested in describing the distribution of this feature in the neighbourhood of $u \in V$. The characteristic function of X for source node u at characteristic function evaluation point $\theta \in \mathbb{R}$ is the function defined by Equation 4.1 where i denotes the imaginary unit.

$$\mathrm{E}\left[e^{i\theta X}|G, u\right] = \sum_{w \in V} P(w|u) \cdot e^{i\theta \mathbf{x}_w} \tag{4.1}$$

In Equation 4.1 the *affiliation probability* $P(w|u)$ describes the strength of the relationship between the source node u and the target node $w \in V$. We would like to emphasize that the source node u and the target nodes do not have to be connected directly and that $\sum_{w \in V} P(w|u) = 1$ holds $\forall u \in V$. We use Euler's identity to obtain the real and imaginary part of the function described

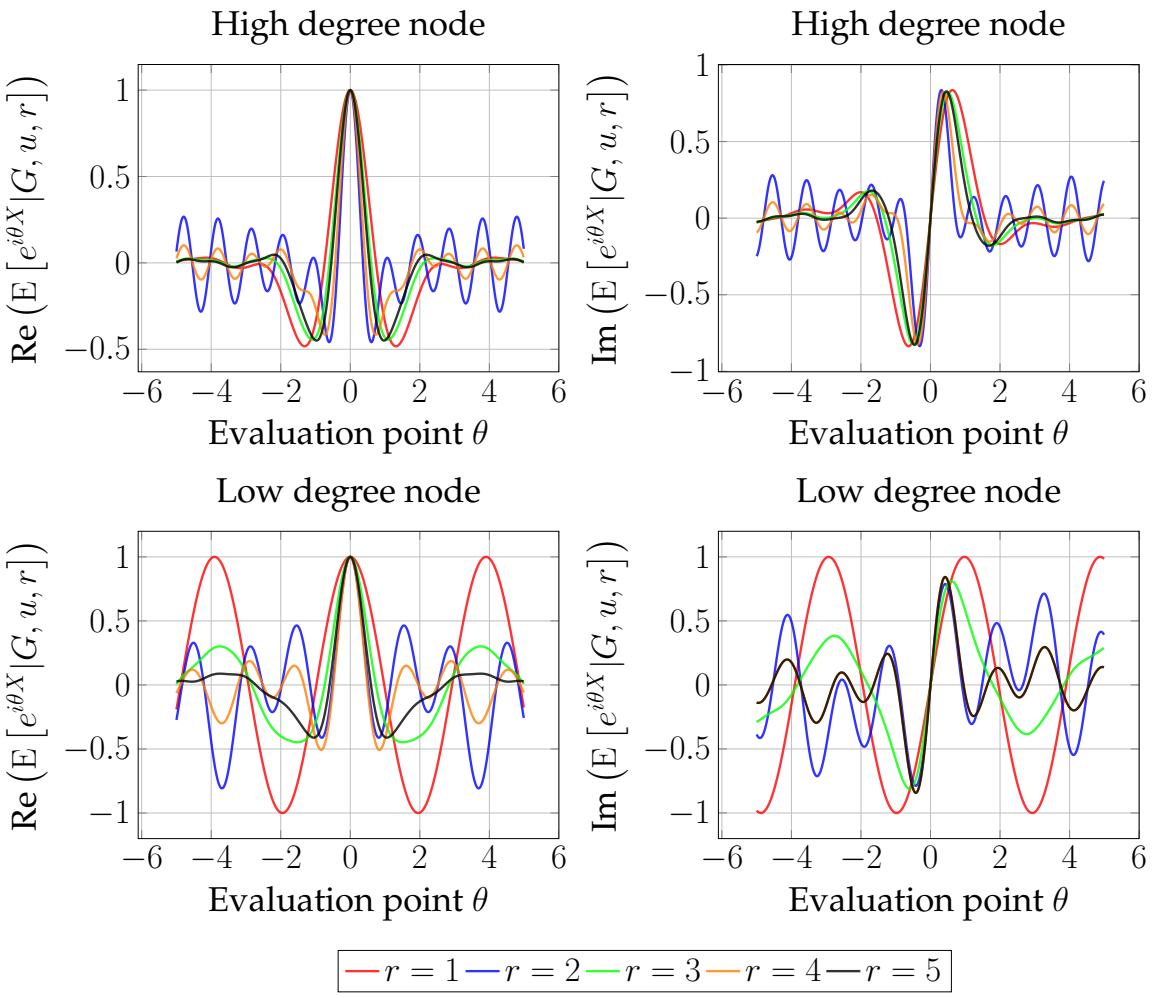


FIGURE 4.2: The real and imaginary parts of the r-scale random walk weighted characteristic function of the log transformed degree for a low degree and high degree node from the Twitch England graph.

by Equation 4.1 which are respectively defined by Equations 4.2 and 4.3.

$$\text{Re}\left(\text{E}\left[e^{i\theta X}|G,u\right]\right) = \sum_{w\in V} P(w|u)\cos(\theta \mathbf{x}_w) \tag{4.2}$$

$$\text{Im}\left(\text{E}\left[e^{i\theta X}|G,u\right]\right) = \sum_{w\in V} P(w|u)\sin(\theta \mathbf{x}_w) \tag{4.3}$$

The real and imaginary parts of the characteristic function are respectively weighted sums of cosine and sine waves where the weight of an individual wave is $P(w|u)$, the evaluation point θ is equivalent to time, and $\mathbf{x}_w$ describes the angular frequency.

The r-scale random walk weighted characteristic function

So far we have not specified how the affiliation probability $P(w|u)$ between the source u and target w is parametrized. Now we will introduce a parametrization which uses random walk transition probabilities. The sequence of nodes in a random walk on G is denoted by $\{v_j, v_{j+1}, \ldots, v_{j+r}\}$.

Let us assume that the neighbourhood of u at scale r consists of nodes that can be reached by a random walk in r steps from source node u. We are interested in describing the distribution

of the feature in the neighbourhood of $u \in V$ at scale r with the real and imaginary parts of the characteristic function – these are respectively defined by Equations 4.4 and 4.5.

$$\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} P(v_{j+r}=w|v_j=u)\cos(\theta \mathbf{x}_w) \tag{4.4}$$

$$\mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} P(v_{j+r}=w|v_j=u)\sin(\theta \mathbf{x}_w) \tag{4.5}$$

In Equations 4.4 and 4.5, $P(v_{j+r}=w|v_j=u) = P(w|u)$ is the probability of a random walk starting from source node u, hitting the target node w in the r^{th} step. The adjacency matrix of G is denoted by $\mathbf{A}$ and the weighted diagonal degree matrix is $\mathbf{D}$. The normalized adjacency matrix is defined as $\widehat{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$. We can exploit the fact that, for a source-target node pair (u,w) and a scale r, we can express $P(v_{j+r}=w|v_j=u)$ with the r^{th} power of the normalized adjacency matrix. Using $\widehat{\mathbf{A}}^r_{u,w} = P(v_{j+r}=w|v_j=u)$, we get Equations 4.6 and 4.7.

$$\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} \widehat{\mathbf{A}}^r_{u,w}\cos(\theta \mathbf{x}_w) \tag{4.6}$$

$$\mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) = \sum_{w\in V} \widehat{\mathbf{A}}^r_{u,w}\sin(\theta \mathbf{x}_w) \tag{4.7}$$

Figure 4.2 shows the real and imaginary part of the *r-scale random walk weighted characteristic function* of the log transformed degree for a low and high degree node in the Twitch England network [165]. A few important properties of the function are visible; (i) the real part is an even function while the imaginary part is odd, (ii) the range of both parts is in the [-1,1] interval, (iii) nodes with different structural roles have different characteristic functions.

Calculating the *r*-scale random walk weighted characteristic function

Up to this point we have only discussed the characteristic function at scale r for a single $u \in V$. However, we might want to characterize every node with respect to a feature in the graph in an efficient way. Moreover, we do not want to evaluate each node characteristic function on the whole domain. Because of this we will sample d points from the domain and evaluate the function at these points which are described by the *evaluation point vector* $\Theta \in \mathbb{R}^d$. We define the *r*-scale random walk weighted characteristic function of the whole graph as the complex matrix valued function denoted as $\mathcal{CF}(G,X,\Theta,r) \rightarrow \mathbb{C}^{|V|\times d}$. The real and imaginary parts of this complex matrix valued function are described by the matrix valued functions in Equations 4.8 and 4.9 respectively.

$$\mathrm{Re}(\mathcal{CF}(G,X,\Theta,r)) = \widehat{\mathbf{A}}^r \cdot \cos(\mathbf{x}\otimes\Theta) \tag{4.8}$$

$$\mathrm{Im}(\mathcal{CF}(G,X,\Theta,r)) = \widehat{\mathbf{A}}^r \cdot \sin(\mathbf{x}\otimes\Theta) \tag{4.9}$$

These matrices describe the feature distributions around nodes if two rows are similar it, implies that the corresponding nodes have similar distributions of the feature around them at scale *r*. This representation can be seen as a *node embedding*, which characterizes the nodes in terms of the

local feature distribution. Calculating the r-scale random walk weighted characteristic function for the whole graph has a time complexity of $\mathcal{O}(|E| \cdot d \cdot r)$ and memory complexity of $\mathcal{O}(|V| \cdot d)$.

Characterizing multiple features for all nodes

Up to this point, we have assumed that the nodes have a single feature, described by the feature vector $\mathbf{x} \in \mathbb{R}^{|V|}$. Now we will consider the more general case when we have a set of k node feature vectors. In a social network these vectors might describe the age, income, and other generic real valued properties of the users. This set of vertex features is defined by $\mathcal{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^k\}$.

We now consider the design of an efficient sparsity aware algorithm which can calculate the r scale random walk weighted characteristic function for each node and feature. We named this procedure *FEATHER*, and it is summarized by Algorithm 4. It evaluates the characteristic functions for a graph for each feature $\mathbf{x} \in \mathcal{X}$ at all scales up to r. The connectivity of the graph is described by the normalized adjacency matrix $\widehat{\mathbf{A}}$. For each feature vector $\mathbf{x}^i,\ i \in 1, \ldots, k$ at scale r we have a corresponding characteristic function evaluation vector $\Theta^{i,r} \in \mathbb{R}^d$. For simplicity we assume that we evaluate the characteristic functions at the same number of points.

Let us look at the mechanics of Algorithm 4. First, we initialize the real and imaginary parts of the embeddings denoted by $\mathbf{Z}_{Re}$ and $\mathbf{Z}_{Im}$ respectively (lines 1 and 2). We iterate over the k different node features (line 3) and the scales up to r (line 4). When we consider the first scale (line 6) we calculate the outer product of the feature being considered and the corresponding evaluation point vector – this results in $\mathbf{H}$ (line 7). We elementwise take the sine and cosine of this matrix (lines 8 and 9). For each scale we calculate the real and imaginary parts of the graph characteristic function evaluations ($\mathbf{H}_{Re}$ and $\mathbf{H}_{Im}$) – we use the normalized adjacency matrix to define the probability weights (lines 10 and 11). We append these matrices to the real and imaginary part of the embeddings (lines 13 and 14). When the characteristic function of each feature is evaluated at every scale we concatenate the real and imaginary part of the embeddings (line 17) and we return this embedding (line 18).

Calculating the outer product (line 7) $\mathbf{H}$ takes $O(|V| \cdot d)$ memory and time. The probability weighting (lines 10 and 11) is an operation which requires $O(|V| \cdot d)$ memory and $O(|E| \cdot d)$ time. We do this for each feature at each scale with a separate evaluation point vector. This means that altogether calculating the r scale graph characteristic function for each feature has a time complexity of $\mathcal{O}((|E| + |V|) \cdot d \cdot r^2 \cdot k)$ and the memory complexity of storing the embedding is $\mathcal{O}(|V| \cdot d \cdot r \cdot k)$.

Theoretical properties

We focus on two theoretical aspects of the r-scale random weighted characteristic function which have practical implications: robustness and how it represents isomorphic graphs.

Remark 4.1. *Let us consider a graph G, the feature X and its corrupted variant X' represented by the vectors $\mathbf{x}$ and $\mathbf{x}'$. The corrupted feature vector only differs from $\mathbf{x}$ at a single node $w \in V$ where $\mathbf{x}'_w = \mathbf{x}_w \pm \varepsilon$ for any $\varepsilon \in \mathbb{R}$. The absolute changes in the real and imaginary parts of the r-scale random walk weighted characteristic function for any $u \in V$ and $\theta \in \mathbb{R}$ satisfy that:*

Data: $\widehat{\mathbf{A}}$ – Normalized adjacency matrix.
$\mathcal{X} = \{\mathbf{x}^1, \ldots, \mathbf{x}^k\}$ – Set of node feature vectors.
$\widetilde{\Theta} = \{\Theta^{1,1}, \ldots, \Theta^{1,r}, \Theta^{2,1}, \ldots, \Theta^{k,r}\}$ – Set of evaluation point vectors.
r – Scale of empirical graph characteristic function.

Result: Node embedding matrix $\mathbf{Z}$.

$\mathbf{Z}_{Re} \leftarrow$ Initialize Real Features()
$\mathbf{Z}_{Im} \leftarrow$ Initialize Imaginary Features()
for i in $1 : k$ **do**
  for j in $1 : r$ **do**
    for l in $1 : j$ **do**
      if $l = 1$ **then**
        $\mathbf{H} \leftarrow \mathbf{x}^i \otimes \Theta^{i,j}$
        $\mathbf{H}_{Re} \leftarrow \cos(\mathbf{H})$
        $\mathbf{H}_{Im} \leftarrow \sin(\mathbf{H})$
      $\mathbf{H}_{Re} \leftarrow \widehat{\mathbf{A}}\mathbf{H}_{Re}$
      $\mathbf{H}_{Im} \leftarrow \widehat{\mathbf{A}}\mathbf{H}_{Im}$
    end
    $\mathbf{Z}_{Re} \leftarrow [\mathbf{Z}_{Re} \mid \mathbf{H}_{Re}]$
    $\mathbf{Z}_{Im} \leftarrow [\mathbf{Z}_{Im} \mid \mathbf{H}_{Im}]$
  end
end
$\mathbf{Z} \leftarrow [\mathbf{Z}_{Im} \mid \mathbf{Z}_{Re}]$
Output $\mathbf{Z}$.

Algorithm 4: Efficient r-scale random walk weighted characteristic function calculation for multiple node features.

$$\underbrace{\left|\mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) - \mathrm{Re}\left(\mathrm{E}\left[e^{i\theta X'}|G,u,r\right]\right)\right|}_{\Delta\mathrm{Re}} \leq 2 \cdot \widehat{\mathbf{A}}^r_{u,w}$$
$$\underbrace{\left|\mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X}|G,u,r\right]\right) - \mathrm{Im}\left(\mathrm{E}\left[e^{i\theta X'}|G,u,r\right]\right)\right|}_{\Delta\mathrm{Im}} \leq 2 \cdot \widehat{\mathbf{A}}^r_{u,w}.$$

Proof. We know that the absolute difference in the real and imaginary part of the characteristic function is bounded by the maxima of such differences:

$$|\Delta\mathrm{Re}| \leq \max|\Delta\mathrm{Re}| \quad \text{and} \quad |\Delta\mathrm{Im}| \leq \max|\Delta\mathrm{Im}|.$$

We will prove the bound for the real part, the proof for the imaginary one follows similarly. Let us substitute the difference of the characteristic functions in the right hand side of the bound:

$$|\Delta\mathrm{Re}| \leq \max\left|\sum_{v\in V}\widehat{\mathbf{A}}^r_{u,v}\cos(\theta\mathbf{x}_v) - \sum_{v\in V}\widehat{\mathbf{A}}^r_{u,v}\cos(\theta\mathbf{x}'_v)\right|.$$

We exploit that $x_v = x'_v,\ \forall v \in V \setminus \{w\}$ so we can rewrite the difference of sums because $\cos(\theta\mathbf{x}_v) - \cos(\theta\mathbf{x}'_v) = 0,\ \forall v \in V \setminus \{w\}$.

$$|\Delta \mathrm{Re}| \leq \max \left| \sum_{v \in V \setminus \{w\}} \left[\widehat{\mathbf{A}}^r_{u,v} \underbrace{\left(\cos(\theta \mathbf{x}_v) - \cos(\theta \mathbf{x}'_v)\right)}_{0} \right] + \widehat{\mathbf{A}}^r_{u,w} \left(\cos(\theta \mathbf{x}_w) - \cos(\theta \mathbf{x}'_w)\right) \right|$$

The maximal absolute difference between two cosine functions is 2 so our proof is complete which means that the effect of corrupted features on the r-scale random walk weighted characteristic function values is bounded by the tie strength regardless the extent of data corruption.

$$|\Delta \mathrm{Re}| \leq \widehat{\mathbf{A}}^r_{u,w} \cdot \underbrace{\max \left|\cos(\theta \mathbf{x}_w) - \cos(\theta \mathbf{x}'_w)\right|}_{2}$$

□

Definition 4.2. *The real and imaginary part of the **mean pooled** r-scale random walk weighted characteristic function are defined as* $\sum_{u \in V} \sum_{w \in V} \widehat{\boldsymbol{A}}^r_{u,w} \cos(\theta \boldsymbol{x}_w)/|V|$ *and* $\sum_{u \in V} \sum_{w \in V} \widehat{\boldsymbol{A}}^r_{u,w} \sin(\theta \boldsymbol{x}_w)/|V|$.

The functions described by Definition 4.2 allow for the characterization and comparison of whole graphs based on structural properties. Moreover, these descriptors can serve as features for graph level machine learning algorithms.

Remark 4.3. *Given two isomorphic graphs G, G' and the respective degree vectors $\boldsymbol{x}, \boldsymbol{x}'$ the mean pooled r-scale random walk weighted degree characteristic functions are the same.*

Proof. Let us denote the normalized adjacency matrices of G and G' as $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{A}}'$. Because G and G' are isomorphic there is a $\mathbf{P}$ permutation matrix for which it holds that $\widehat{\mathbf{A}} = \mathbf{P}\widehat{\mathbf{A}}'\mathbf{P}^{-1}$. Using the same permutation matrix we get that $\mathbf{x} = \mathbf{P}\mathbf{x}'$. Using Definition 4.2 and the previous two equations it follows that the real and imaginary parts of pooled characteristic functions satisfy that

$$\sum_{u \in V} \sum_{w \in V} \widehat{\mathbf{A}}^r_{u,w} \cos(\theta \mathbf{x}_w)/|V| = \sum_{u \in V} \sum_{w \in V} (\mathbf{P}\widehat{\mathbf{A}}\mathbf{P}^{-1})^r_{u,w} \cos(\theta \cdot (\mathbf{P}\mathbf{x})_w)/|V|$$
$$\sum_{u \in V} \sum_{w \in V} \widehat{\mathbf{A}}^r_{u,w} \sin(\theta \mathbf{x}_w)/|V| = \sum_{u \in V} \sum_{w \in V} (\mathbf{P}\widehat{\mathbf{A}}\mathbf{P}^{-1})^r_{u,w} \sin(\theta \cdot (\mathbf{P}\mathbf{x})_w)/|V|.$$

□

Parametric characteristic functions

Our discussion postulated that the evaluation points of the r-scale random walk characteristic function are predetermined. However, we can define parametric models where these evaluation points are learned in a semi-supervised fashion to make the evaluation points selected the most discriminative with regards to a downstream classification task. The process which we describe in Algorithm 4 can be interpreted as the forward pass of a graph neural network which uses the normalized adjacency matrix and node features as input. This way the evaluation points and the weights of the classification model could be learned jointly in an end-to-end fashion.

Softmax parametric model

Now we define the classifiers with learned evaluation points, let $\mathbf{Y}$ be the $|V| \times C$ one-hot encoded matrix of node labels, where C is the number of node classes. Let us assume that $\mathbf{Z}$ was calculated by using Algorithm 4 in a forward pass with a trainable $\widetilde{\Theta}$. The classification weights of the softmax characteristic function classifier are defined by the trainable weight matrix $\beta \in \mathbb{R}^{(2 \cdot k \cdot d \cdot r) \times C}$. The class label distribution matrix of nodes outputted by the softmax characteristic function model is defined by Equation 4.10 where the softmax function is applied row-wise. We reference this supervised softmax model as *FEATHER-L*.

$$\widehat{\mathbf{Y}} = \text{softmax}(\mathbf{Z} \cdot \beta) \tag{4.10}$$

Neural parametric model

We introduce the forward pass of a neural characteristic function model with a single hidden layer of feed forward neurons. The trainable input weight matrix is $\beta_0 \in \mathbb{R}^{(2 \cdot k \cdot d \cdot r) \times h}$ and the output weight matrix is $\beta_1 \in \mathbb{R}^{h \times C}$, where h is the number of neurons in the hidden layer. The class label distribution matrix of nodes output by the neural model is defined by Equation 4.11, where $\sigma(\cdot)$ is an activation function applied element-wise (in our experiments it is a ReLU). We refer to neural models with this architecture as *FEATHER-N*.

$$\widehat{\mathbf{Y}} = \text{softmax}(\sigma(\mathbf{Z} \cdot \beta_0) \cdot \beta_1) \tag{4.11}$$

Optimizing the parametric models

The log-loss of the *FEATHER-N* and *FEATHER-L* models being minimized is defined by Equation 4.12 where $U \subseteq V$ is the set of labeled training nodes.

$$\mathcal{L} = -\sum_{u \in U} \sum_{c=1}^{C} \mathbf{Y}_{u,c} \cdot \log(\widehat{\mathbf{Y}}_{u,c}) \tag{4.12}$$

This loss is minimized with a variant of gradient descent to find the optimal values of β (respectively β_0 and β_1) and $\widetilde{\Theta}$. The softmax model has $\mathcal{O}(k \cdot r \cdot d \cdot C)$ while the neural model has $\mathcal{O}(k \cdot r \cdot d \cdot h + C \cdot h)$ free trainable parameters, As a comparison, generating the representations upstream with *FEATHER* and learning a logistic regression has $\mathcal{O}(k \cdot r \cdot d \cdot C)$ trainable parameters.

4.4 Experimental evaluation

In this section, we overview the datasets used to quantify representation quality. We demonstrate how node and graph features distilled with *FEATHER* can be used to solve node and graph classification tasks. Furthermore, we highlight the transfer learning capabilities, scalability and robustness of our method.

Datasets

We briefly discuss the real world datasets and their descriptive statistics which we use to evaluate the node and graph features extracted with our proposed methods.

TABLE 4.2: Statistics of social networks used for the evaluation of node classification algorithms, sensitivity analysis, and transfer learning.

Dataset	Nodes	Density	Clustering Coefficient	Diameter	Unique Features	Classes
Wiki Croco	11,631	0.003	0.026	11	13,183	2
FB Page-Page	22,470	0.001	0.232	15	4,714	4
LastFM ASIA	7,624	0.001	0.179	15	7,842	18
Deezer EUR	28,281	0.002	0.096	21	31,240	2
Twitch DE	9,498	0.003	0.047	7	3,169	2
Twitch EN	7,126	0.002	0.042	10	3,169	2
Twitch ES	4,648	0.006	0.084	9	3,169	2
Twitch PT	1,912	0.017	0.131	7	3,169	2
Twitch RU	4,385	0.004	0.049	9	3,169	2
Twitch TW	2,772	0.017	0.120	7	3,169	2

Node level datasets.

We used various publicly available, and self-collected social network and webgraph datasets to evaluate the quality of node features extracted with *FEATHER*. The descriptive statistics of these datasets are summarized in Table 7.1. These graphs are heterogeneous with respect to size, density, and number of features, and they allow for binary and multi-class node classification.

- **Wikipedia Crocodiles [168]:** A webgraph of Wikipedia articles about crocodiles where each node is a page and edges are mutual links between edges. Attributes represent nouns appearing in the articles and the binary classification task on the dataset is deciding whether a page is popular or not.
- **Twitch Social Networks [165]:** Social networks of gamers from the streaming service Twitch. Features describe the history of games played and the task is to predict whether a gamer streams adult content. The country specific graphs share the same node features which means that we can perform transfer learning with these datasets.
- **Facebook Page-Page [165]:** A webgraph of verified Facebook pages which liked each other. Features were extracted from page descriptions and the classification target is the page category.
- **LastFM Asia:** The LastFM Asia graph is a social network of users from Asian (e.g. Philippines, Malaysia, Singapore) countries which we collected. Nodes represent users of the music streaming service LastFM and links among them are friendships. We collected these datasets in March 2020 via the use of the API. The classification task related to these two datasets is to predict the home country of a user given the social network and artists liked by the user.

- **Deezer Europe:** A social network of European Deezer users which we collected from the public API in March 2020. Nodes represent users and links are mutual follower relationships among users. The related classification task is the prediction of gender using the friendship graph and the artists liked.

Graph level datasets.

We utilized a range of publicly available non-attributed, social graph datasets to assess the quality of graph level features distilled via our procedure. Summary statistics, enlisted in Table 7.2, demonstrate that these datasets have a large number of small graphs with varying size, density and diameter.

- **Reddit Threads [168]**: A collection of Reddit thread and non-thread based discussion graphs. The related task is to correctly classify graphs according the thread – non-thread categorization.
- **Twitch Egos [168]** The ego-networks of Twitch users who participated in the partnership program. The classification task entails the identification of gamers who only play with a single game.
- **GitHub Repos [168]:** Social networks of developers who starred machine learning and web design repositories. The target is the type of the repository itself.
- **Deezer Egos [168]:** A small collection of ego-networks for European Deezer users. The related task involves the prediction of the ego node's gender.

TABLE 4.3: Statistics of graph datasets used for the evaluation of graph classification algorithms.

		Nodes		**Density**		**Diameter**	
Dataset	**Graphs**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Reddit Threads	203,088	11	97	0.021	0.382	2	27
Twitch Egos	127,094	14	52	0.038	0.967	1	2
GitHub Repos	12,725	10	957	0.003	0.561	2	18
Deezer Egos	9,629	11	363	0.015	0.909	2	2

Node classification

The node classification performance of *FEATHER* variants is compared to neighbourhood based, structural and attributed node embedding techniques. We also studied the performance in contrast to various competitive graph neural network architectures.

Experimental settings

We report mean micro averaged test AUC scores with standard errors calculated from 10 seeded splits with a 20%/80% train-test split ratio in Table 7.5.

The unsupervised neighbourhood based [22, 147, 148, 154, 170, 198], structural [11] and attributed node [116, 165, 227, 231, 234, 244] embeddings were created by the *Karate Club* [168]

software package and used the default hyperparameter settings of the 1.0 release. This ensure that the number of free parameters used to represent the nodes by the upstream unsupervised models is the same. We used the publicly available official Python implementation of *Node2Vec* [66] with the default settings and the *In-Out* and *Return* hyperparameters were fine tuned with 5-fold cross validation within the training set. The downstream classifier was a logistic regression which used the default hyperparameter settings of *Scikit-Learn* [145] with the *SAGA* optimizer [37].

TABLE 4.4: Mean micro-averaged AUC values on the test set with standard errors on the node level datasets calculated from 10 seed train-test splits. Black bold numbers denote the best performing unsupervised model, while blue ones denote the best supervised one.

	Wikipedia Crocodiles	**Facebook Page-Page**	**LastFM Asia**	**Deezer Europe**
DeepWalk [147]	.820 ± .001	.880 ± .001	.918 ± .001	.520 ± .001
LINE [198]	.856 ± .001	.956 ± .001	.949 ± .001	.543 ± .001
Walklets [148]	.872 ± .001	.975 ± .001	**.950 ± .001**	.547 ± .001
HOPE [139]	.855 ± .001	.903 ± .002	.922 ± .001	.539 ± .001
NetMF [154]	.859 ± .001	.946 ± .001	.943 ± .001	.538 ± .001
Node2Vec [66]	.850 ± .001	.974 ± .001	.944 ± .001	.556 ± .001
Diff2Vec [170]	.812 ± .001	.867 ± .001	.907 ± .001	.521 ± .001
GraRep [22]	.871 ± .001	.951 ± .001	.926 ± .001	.547 ± .001
Role2Vec [11]	.801 ± .001	.911 ± .001	.924 ± .001	.534 ± .001
GEMSEC [166]	.858 ± .001	.933 ± .001	**.951 ± .001**	.544 ± .001
ASNE [116]	.853 ± .001	.933 ± .001	.910 ± .001	.528 ± .001
BANE [231]	.534 ± .001	.866 ± .001	.610 ± .001	.521 ± .001
TENE [234]	.893 ± .001	.874 ± .001	.855 ± .002	.639 ± .002
TADW [227]	.901 ± .001	.849 ± .001	.851 ± .001	.644 ± .001
SINE [244]	.895 ± .001	.975 ± .001	.944 ± .001	.618 ± .001
GCN [92]	.924 ± .001	**.984 ± .001**	.962 ± .001	.632 ± .001
GAT [210]	.917 ± .002	**.984 ± .001**	.956 ± .001	.611 ± .002
GraphSAGE [71]	.916 ± .001	**.984 ± .001**	.955 ± .001	.618 ± .001
ClusterGCN [28]	.922 ± .001	.977 ± .001	.944 ± .002	.594 ± .002
APPNP [93]	.900 ± .001	**.986 ± .001**	**.968 ± .001**	.667 ± .001
MixHop [5]	.928 ± .001	.976 ± .001	.956 ± .001	**.682 ± .001**
SGConv [222]	.889 ± .001	.966 ± .001	.957 ± .001	.647 ± .001
FEATHER	**.943 ± .001**	**.981 ± .001**	**.954 ± .001**	**.651 ± .001**
FEATHER-L	**.944 ± .002**	**.984 ± .001**	.960 ± .001	.656 ± .001
FEATHER-N	**.947 ± .001**	**.987 ± .001**	**.970 ± .001**	.673 ± .001

Supervised baselines were implemented with the *PyTorch Geometric* framework [47] and as a pre-processing step, the dimensionality of vertex features was reduced to be 128 by the *Scikit-Learn* implementation of Truncated SVD [70]. Each supervised model considers neighbours from 2 hop neighbourhoods except *APPNP* [93] which used a teleport probability of 0.2 and 10 personalized PageRank approximation iterations. Models were trained with the Adam optimizer [91] with a learning rate of 0.01 for 200 epochs. The input layers of the models had 32 filters and between the

final and input layer we used a 0.5 dropout rate [185] during training time. The *ClusterGCN* [28] models decomposed the graph with the *METIS* [88] algorithm before training – the number of clusters equaled the number of classes.

The *FEATHER* model variants used a combination of neighbourhood based, structural and generic vertex attributes as input besides the graph itself. Specifically we used:

- **Neighbourhood features:** We used Truncated SVD to extract 32 dimensional node features from the normalized adjacency matrix.
- **Structural features:** The log transformed degree and the clustering coefficient are used as structural vertex features.
- **Generic node features:** We reduced the dimensionality of generic vertex features with Truncated SVD to be 32 dimensional for each network.

The unupservised *FEATHER* model used 16 evaluation points per feature, which were initialized uniformly in the $[0, 5]$ domain, and a scale of $r = 2$. We used a logistic regression downstream classifier. The characteristic function evaluation points of the supervised *FEATHER-L* and *FEATHER-N* models were initialized similarly. Models were trained by the Adam optimizer with a learning rate of 0.001, for 50 epochs and the neural model had 32 neurons in the hidden layer.

Node classification performance

Our results in Table 7.5 demonstrate that the unsupervised *FEATHER* algorithm outperforms the proximity preserving, structural and attributed node embedding techniques. This predictive performance advantage varies between 0.4% and 4.6% in terms of micro averaged test AUC score. On the Wikipedia, Facebook and LastFM datasets, the performance difference is significant at an $\alpha = 1\%$ significance level. On these three datasets the best supervised *FEATHER* variant marginally outperforms *graph neural networks* between 0.1% and 2.1% in terms of test AUC. However, this improvement of classification is only significant on the Wikipedia dataset at $\alpha = 1\%$.

Graph classification

The graph classification performance of unsupervised and supervised *FEATHER* models is compared to that of implicit matrix factorization, spectral fingerprinting and graph neural network models.

Experimental settings

We report average test AUC values with standard errors on binary classification tasks calculated from 10 seeded splits with a 20%/80% train-test split ratio in Table 7.4.

The unsupervised implicit matrix factorization [27, 134] and spectral fingerprinting [34, 51, 202, 213] representations were produced by the *Karate Club* framework with the standard hyperparameter settings of the 1.0 release. The downstream graph classifier was a logistic

regression model implemented in *Scikit-Learn*, with the standard hyperparameters and the *SAGA* [37] optimizer.

Supervised models used the one-hot encoded degree, clustering coefficient and eccentricity as node features. Each method was trained by minimizing log-loss with the Adam optimizer [91] using a learning rate of 0.01 for 10 epochs with a batch size of 32. We used two consecutive graph convolutional [92] layers with 32 and 16 filters and ReLu activations. In the case of mean and maximum pooling the output of the second convolutional layer was pooled and fed to a fully connected layer. We do the same with Sort Pooling [247] by keeping 5 nodes and flattening the output of the pooling layer to create graph representations. In the case of Top-K pooling [52] and SAG Pooling [102] we pool the nodes after each convolutional layer with a pooling ratio of 0.5 and output graph representations with a final max pooling layer. The output of advanced pooling layers [52, 102, 247] was fed to a fully connected layer. The output of the final layers was transformed by the softmax function.

We used the unsupervised *FEATHER* model to create graph descriptors. We pooled the node features extracted for each characteristic function evaluation point with a permutation invariant aggregation function such as the mean, maximum and minimum. Node level representations only used the log transformed degree as a feature. We set $r = 5$, $d = 25$, and initialized the characteristic function evaluation points in the $[0, 5]$ interval uniformly. Using these descriptors we utilized logistic regression as a classifier with the settings used with other unsupervised methods.

Graph classification performance

Our results demonstrate that our proposed pooled characteristic function based classification method outperforms both supervised and unsupervised graph classification methods on the Reddit Threads, Twitch Egos and Github Repos datasets. The performance advantage of *FEATHER* varies between 1.1% and 12.0% in terms of AUC, which is a significant peformance gain on all three of these datasets at an $\alpha = 1\%$ significance level. On the Deezer Egos dataset the disadvantage of our method is not significant, but specific supervised and unsupervised procedures have a somewhat better predictive performance in terms of test AUC. We also have evidence that the mean pooling of the node level characteristic functions provides superior peformance on most datasets considered.

Sensitivity analysis

We investigated how the representation quality changes when the most important hyperparameters of the r-scale random walk weighted characteristic function are manipulated. Precisely, we looked at the scale of the r-scale random walk weighted characteristic function and the number of evaluation points.

We use the Facebook Page-Page dataset, with the standard (20% /80%) split and input the log transformed degree as a vertex feature. Figure 4.3 plots the average test AUC against the manipulated hyperparameter calculated from 10 seeded splits. The models were trained with the hyperparameter settings discussed in Subsection 4.4. We chose a scale of 5 when the number of

TABLE 4.5: Mean AUC values with standard errors on the graph datasets calculated from 10 seed train-test splits. Bold numbers denote the model with the best performance.

	Reddit Threads	**Twitch Egos**	**GitHub Repos**	**Deezer Egos**
GL2Vec [27]	.754 ± .001	.670 ± .001	.532 ± .002	.500 ± .001
Graph2Vec [134]	.808 ± .001	.698 ± .001	.563 ± .002	.510 ± .001
SF [34]	.819 ± .001	.642 ± .001	.535 ± .001	.503 ± .001
NetLSD [202]	.817 ± .001	.630 ± .001	.614 ± .002	.525 ± .001
FGSD [213]	.822 ± .001	.699 ± .001	.650 ± .002	**.528 ± .001**
Geo-Scatter [51]	.800 ± .001	.695 ± .001	.532 ± .001	.524 ± .001
Mean Pool	.801 ± .002	.708 ± .001	.599 ± .003	.503 ± .001
Max Pool	.805 ± .001	.713 ± .001	.612 ± .013	.515 ± .001
Sort Pool [247]	.807 ± .001	.712 ± .001	.614 ± .010	**.528 ± .001**
Top K Pool [52]	.807 ± .001	.706 ± .002	.634 ± .001	.520 ± .003
SAG Pool [102]	.804 ± .001	.705 ± .002	.620 ± .001	.518 ± .003
FEATHER MIN	**.834 ± .001**	**.719 ± .001**	.694 ± .001	.518 ± .001
FEATHER MAX	**.831 ± .001**	**.718 ± .001**	.689 ± .001	.521 ± .002
FEATHER AVE	.823 ± .001	**.719 ± .001**	**.728 ± .002**	**.526 ± .001**

evaluation points was modulated and used 25 evaluation points when the scale was manipulated. The evaluation points were initialized uniformly in the $[0, 5]$ domain.

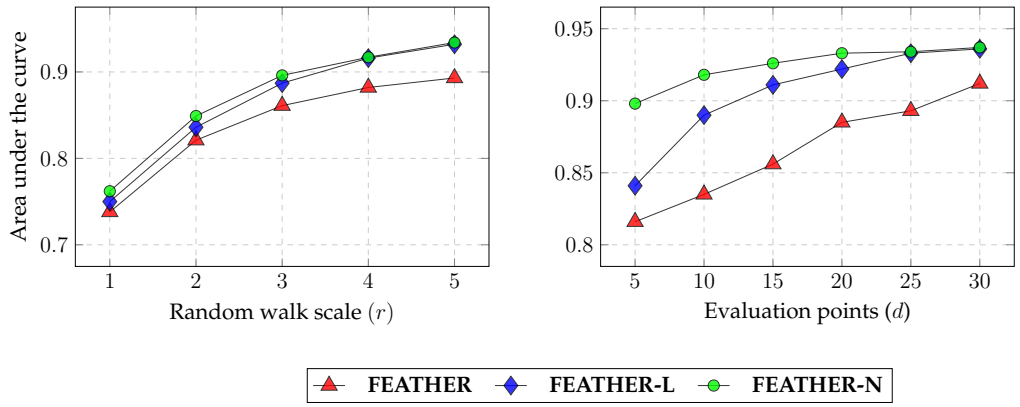


FIGURE 4.3: Mean AUC values on the Facebook page-page test set (10 seeded splits) achieved by FEATHER model variants as a function of random walk scale and characteristic function evaluation point count.

Scale of the characteristic function

First, we observe that including information from higher order neighbourhoods is valuable for the classification task. Second, the marginal performance increase is decreasing with the increase of the scale. Finally, when we only consider the first hop of neighbours we observe little performance difference between the unsupervised and supervised model variants. When higher order neighbourhoods are considered the supervised models have a considerable advantage.

Number of evaluation points

Increasing the number of characteristic function evaluation points increases the performance on the downstream predictive task. Supervised models are more efficient when the number of characteristic function evaluation points was low. The neural model is efficient in terms of the evaluation points needed for a good predictive performance. It is evident that the marginal predictive performance gains of the supervised models are decreasing with the number of evaluation points.

Transfer learning

Using the Twitch datasets, we demonstrate that the r-scale random walk weighted characteristic function features are robust and can be easily utilized in a transfer learning setup. We support evidence that the supervised parametric models also work in such scenarios. Figure 4.4 shows the transfer learning results for German, Spanish and Portuguese users, where the predictive performance is measured by average AUC values based on 10 experiments.

Each model was trained with nodes of a fully labeled source graph and evaluated on the nodes of the target graph. This transfer learning experiment requires that graphs share the target variable (abusing the Twitch platform), and that the characteristic function is calculated for the same set of node features. All models utilized the log transformed degree centrality of nodes as a shared and cheap-to-calculate structural feature. We used a scale of $r = 5$ and $d = 25$ characteristic function evaluation points for each *FEATHER* model variant. Models were fitted with the hyperparameter settings described in Subsection 4.4.

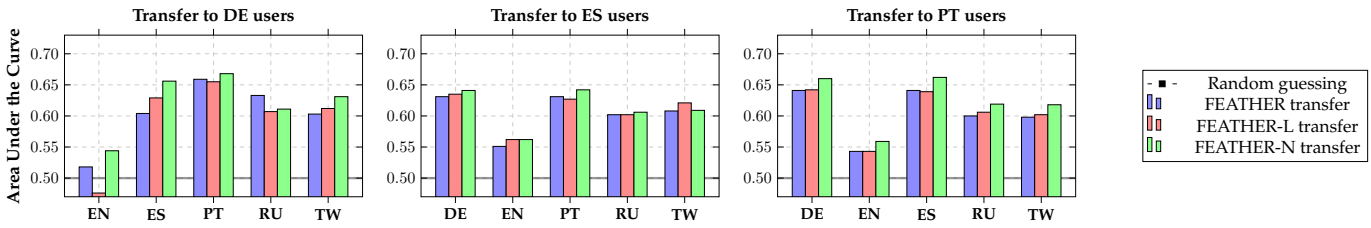


FIGURE 4.4: Transfer learning performance of FEATHER variants on the Twitch Germany, Spain and Portugal datasets as target graphs. The transfer performance was evaluated by mean AUC values calculated from 10 seeded experimental repetitions.

Firstly, our results presented on Figure 4.4 support that even the characteristic function of a single structural feature is sufficient for transfer learning as we are able to outperform the random guessing of labels in most transfer scenarios. Secondly, we see that the neural model has a predictive performance advantage over the unsupervised *FEATHER* and the shallow *FEATHER-L* model. Specifically, for the Portuguese users this advantage varies between 2.1 and 3.3% in terms of average AUC value. Finally, transfer from the English users seems to be poor to the other datasets. Which implies that the abuse of the platform is associated with different structural features in that case.

Runtime performances

We evaluated the runtime needed for calculating the proposed r-scale random walk weighted characteristic function. Using a synthetic Erdős-Rényi graph with 2^{12}nodes and 2^4 edges per node, we measured the runtime of Algorithm 4 for various values of the size of the graph, number of features and characteristic function evaluation points. Figure 4.5 shows the mean logarithm of the runtimes against the manipulated input parameter based on 10 experimental repetitions.

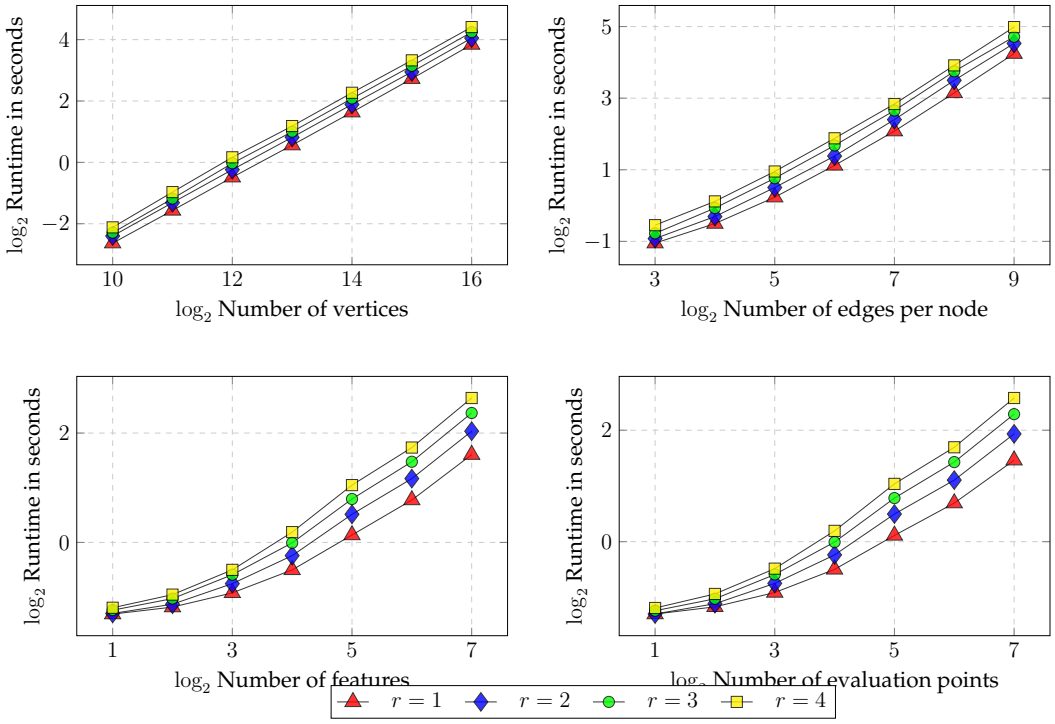


FIGURE 4.5: Average runtime of *FEATHER* as a function of node and edge count, number of features and characteristic function evaluation points. The mean runtimes were calculated from 10 repetitions using synthetic Erdős Rényi graphs.

Our results support the theoretical runtime complexities discussed in Subsection 4.3. Practically it means that doubling the number of nodes, edges, features or characteristic function evaluation points doubles the expected runtime of the algorithm. Increasing the scale of random walks (considering more hops) increases the runtime. However, for small values of r the quadratic increase in runtime is not evident.

4.5 Conclusions and future directions

We presented a general notion of characteristic functions defined on attributed graphs. We discussed a specific instance of these – the r-scale random walk weighted characteristic function. We proposed *FEATHER* an efficient algorithm to calculate this characteristic function efficiently on large attributed graphs in linear time to create Euclidean vector space representations of nodes. We proved that *FEATHER* is robust to data corruption and that isomorphic graphs have the same vector space representations. We have shown that *FEATHER* can be interpreted as the

forward pass of a neural network and can be used as a differentiable building block for parametric classifiers.

We demonstrated on various real world node and graph classification datasets that *FEATHER* variants are competitive with comparable embedding and graph neural network models. Our transfer learning results support that *FEATHER* models are able to efficiently and robustly transfer knowledge from one graph to another one. The sensitivity analysis of characteristic function based models and node classification results highlight that supervised *FEATHER* models have an edge compared to unsupervised representation creation with characteristic functions. Furthermore, runtime experiments presented show that our proposed algorithm scales linearly with the input size such as number of nodes, edges and features in practice.

As a future direction we would like to point out that the forward pass of the *FEATHER* algorithm could be incorporated in temporal, multiplex and heterogeneous graph neural network models to serve as a multi-scale vertex feature extraction block. Moreover, one could define a characteristic function based node feature pooling where the node feature aggregation is a learned permutation invariant function. Finally, our evaluation of the proposed algorithms was limited to social networks and web graphs – testing it on biological networks and other types of datasets could be an important further direction.

Chapter 5

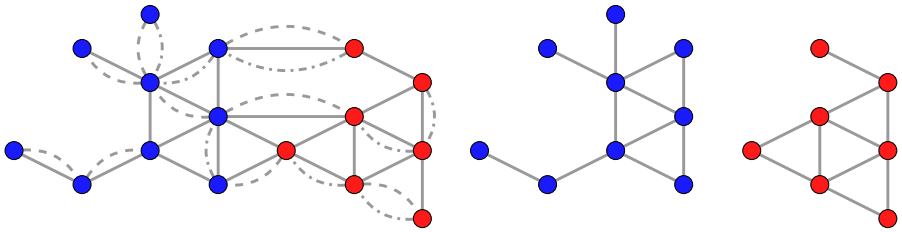
Pathfinder Discovery Networks for Neural Message Passing

In this work we propose *Pathfinder Discovery Networks* (PDNs), a method for jointly learning a message passing graph over a multiplex network with a downstream semi-supervised model. PDNs inductively learn an aggregated weight for each edge, optimized to produce the best outcome for the downstream learning task. PDNs are a generalization of attention mechanisms on graphs which allow flexible construction of similarity functions between nodes. They also support edge convolutions and cheap multiscale mixing layers. We show that PDNs overcome weaknesses of existing methods for graph attention (e.g. Graph Attention Networks), such as the diminishing weight problem.

Our experimental results demonstrate competitive predictive performance on academic node classification tasks. Additional results from a challenging suite of node classification experiments show how PDNs can learn a wider class of functions than existing baselines. We analyze the relative computational complexity of PDNs, and show that PDN runtime is not considerably higher than static-graph models. Finally, we discuss how PDNs can be used to construct an easily interpretable attention mechanism that allows users to understand information propagation in the graph.

5.1 Introduction

Recently, there has been a surge of interest in applying neural networks to graph data. The last few years have seen the development of a wide variety of approaches, ranging from graph embedding [66, 147, 152, 170], to graph convolutional networks [71, 92], to message passing neural networks [58]. Though powerful, many of these approaches have a serious limitation: they assume that the underlying graph is static, provided as an immutable input parameter where edges between node-pairs have only a single weight. However, in many real world applications, there is rarely one 'correct' graph – instead, the best task performance comes from combining many different types of relationships [69]. For example, in a video classification task, the best graph to use might consider several different types of similarity between videos (e.g. both image similarity and audio similarity). In practice, we believe that it is a mistake to separate graph construction from the learning task at hand; rather, the optimal graph must come from deep consideration of the problem being solved.



(A) Multiplex graph (B) Learned pathfinder graph

FIGURE 5.1: Pathfinder discovery networks take multiple sets of weighted edges and learn a graph specifically suited to a downstream predictive task. In Subfigure 5.1a we have three types of edges and a two types of nodes. The pathfinder discovery network would perform node classification and output a learned pathfinder graph where inter-class edges are forgotten – see Subfigure 5.1b.

Some methods have attempted to relax these limitations. For example, *Graph Attention Networks* (GATs) [210], attempt to re-weight each edge in the graph. However, GAT models are prone to overfitting, and due to their over-parameterization have difficulties being trained on large real world datasets. Further, they suffer from a diminishing weight problem, which drives learned edge weights towards zero as the degree of a node increases. Finally (and perhaps most importantly) GAT-style attention constrains edge reweighting to be a single aggregation of the node features, which prevents GATs from learning complex difference operators between edges in varying neighborhood structures.

A separate body of work has proposed specific models for heterogeneous data [237, 248]. However these models are often highly specific to specific kinds of data inputs (e.g. a model might support video and relational data, but not geospatial data), and are therefore difficult to integrate with new advances in modeling. More to the point, though heterogeneous approaches do incorporate a wide variety of data types, they still treat graph construction as a fundamentally isolated problem from graph learning. As a result, heterogeneous approaches will struggle for the same reasons mentioned above.

In this work, we answer the question: *"How can we learn to construct the optimal graph for solving any given learning problem?"*. For inspiration, we look back to *pathfinder networks* [178], a graph construction measure from the psychology literature. In a pathfinder network, multiple kinds of proximity judgements (e.g. relatedness scores from humans) are considered simultaneously in order to determine edges between node-pairs. A discrete algorithm finds the graph which best preserves some property (e.g. shortest paths) of the input proximities. While traditional pathfinder networks are useful for tasks such as aggregating subjective information, they are unfortunately unsuitable for use in most graph learning tasks. More recently, Halcrow et. al [69] describe a similar problem of graph construction from multiple proximities which occurs in a wide variety of industrial applications. Their solution, Grale, uses a model to precompute a fused similarity network for graph learning (one edge at a time). While this system has many advantages (scalability, allows use of different kinds of relationships, etc) and has been used in a wide variety of applications at Google, it is not able to learn a graph jointly with a downstream task. Here, we go one step further, and present *Pathfinder Discovery Networks*, a framework for learning a network over a set of entities with diverse similarity scores jointly with a graph neural network task.

Our main contribution is the design and validation of the *pathfinder layer* - a differentiable neural network layer which is able to combine multiple sources of proximity information defined over a set of nodes to form a single weighted graph. The pathfinder layer uses a feed forward neural network to learn the edge weights while the sparsity of the underlying weighted multiplex graph is unchanged (see Figure 5.1). This layer feeds directly into a downstream GNN model that is set up to learn arbitrary tasks – in this chapter, semi-supervised classification. Gradients from the supervised classification task propagate down to the edge weights, allowing PDNs to create a graph that is optimized for the classification task at hand. Our model learns this graph in an inductive manner, allowing transfer of learned graph aggregation from one graph to another.

We demonstrate the flexibility of our framework by showing that a number models can be seen as special cases of our general framework. First, we show how edge convolutional models can be formulated with our modeling framework. Second, we establish that one can define models that perform cheap multi-scale mixing with the pathfinder layer.

Our empirical analysis focuses on node classification tasks, feature importance measurements, and runtime comparisons. We use synthetic experiments to demonstrate a class of learnable tasks where PDNs significantly outperform current state-of-the-art methods thanks to the unique ability to learn arbitrary functions over multiple proximity inputs. We then switch to real world node classification problems, and demonstrate that PDN has a 0.8%-3.5% predictive performance advantage over the most competitive existing graph neural network models in terms of accuracy. We analyze the runtime of PDNs and demonstrate that the pathfinder layer increases the training runtime by a constant multiplier. Finally, we describe how the weights of the pathfinder layer can be seen as attention over the input graphs and edge features, and add interpretability to the underlying information propagation. The key contributions of our work are as follows:

1. We propose a flexible framework to learn a single graph for message passing from multiple graphs jointly with any graph convolution layer.

2. We showcase how this framework can be used to define edge convolution neural network models where the message passing graph is learned from node features.

3. We define models with cheap multi-scale mixing where the adjacency matrix of the message passing graph is a linear combination of adjacency matrix powers.

4. We empirically demonstrate that our models have competitive results on a range of node classification tasks, have decent runtimes on small-scale graphs, and have explainable weights in case of the simple models.

The source code of Pathfinder Discovery Networks is available at `https://github.com/benedekrozemberczki/PDN`.

5.2 Preliminaries

We begin by summarizing the notation used in our work and reviewing the related concepts of multiplexity, simplified spectral graph convolutions, and graph attention. We frame our model as a general building block that can be applied to a wide variety of graph neural network designs.

Notation

We assume that we have a set of vertices V, and D graphs defined on these vertices described by $\mathcal{G}_1, \ldots, \mathcal{G}_D$ with respective edge sets $E_1, \ldots, E_D$. These graphs can be represented as $|V| \times |V|$ adjacency matrices which are respectively denoted by $\widetilde{\mathbf{A}}_1, \ldots, \widetilde{\mathbf{A}}_D$. We assume that nodes have generic vertex features. For the whole set V, these features are described by a feature matrix $\mathbf{X} \in \mathbb{R}^{|V| \times F}$, where F is the number of features. In addition, for each node we have a target that we want to predict. For the whole set V, the targets are defined as a $|V| \times C$ binary matrix, where C is the number of node classes. Our goal is to predict the target class matrix using the graphs and the node features.

Multiplex graphs and learning This problem setup can be framed as node classification with a *weighted multiplex graph* [128] which has no inter-layer edges (Figure 5.1a). Unlike heterogeneous graphs, multiplex graphs operate on a single node type. Current approaches to learning from multiplex graphs only generalize neighbourhood based embeddings to accommodate a multiplex setting [127, 245]. In these approaches, a separate node embedding is learned for each graph (or each layer in the multiplex graph); these embeddings are concatenated to form the node representations. These approaches have two limitations. First, the node embeddings are transductive so the models do not generalize from one graph to another. Second, these approaches are expensive, as node level embeddings must be calculated for each graph separately.

Graph convolutions

The traditional setting of spectral graph convolutional networks [92] has a single graph $\mathcal{G}$ and a corresponding adjacency matrix $\mathbf{A}$. In the forward pass of the spectral model the degree normalized adjacency matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is used to propagate the hidden node representations. These hidden representations are obtained by multiplying the feature matrix $\mathbf{X}$ by a trainable $F \times d$ weight matrix $\mathbf{W}$. Finally, the aggregated representations are transformed by $\sigma(\cdot)$ an elementwise non-linearity just as in Equation equation 5.1 which describes the whole forward pass.

$$\mathbf{Z} = \sigma(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{X}\mathbf{W}) \tag{5.1}$$

As defined, the spectral graph convolutional model cannot accommodate the presence of multiple graphs – one either has to come up with a pre-defined edge weight aggregation function or use only one of the graphs from $\mathcal{G}_1, \ldots, \mathcal{G}_D$ as the message passing graph. PDNs are directly motivated by this fundamental weakness. We note that a range of graph neural network architectures use the spectral graph convolution as a building block [26]. We believe that overcoming the single graph limitation could therefore lead to significant improvements in all other related models.

Graph attention networks

Graph attention networks [210] learn the edge weights used for message passing using the features of nodes at the edge endpoints. Node features are transformed by a learnable parameter matrix and concatenated together for each edge. The node-pair representations are multiplied by an attention vector, and the weight of each edge is decided by a softmax unit defined over the

neighbors of the source node. Although the GAT model can learn multiple edge weights with multiple attention heads, this is not a straightforward comparison to learning multiplex graphs. In [210], each attention head is trained on the *same node features*, which precludes incorporating unique features from multiple sources. Furthermore, GAT cannot fully leverage the power of multiple attention heads, because the final edge weight is a simple average over the individual heads (and not a learned function). The GAT model is therefore unable to learn an expressive range of functions over multiple edge sets, which severely limits its applicability to multiplex graph problems.

Pathfinder discovery networks as a building block

The pathfinder learning module introduced in our work is sufficiently general to serve the message passing matrix for a wide range of general graph neural network models defined on multiplex graphs. A pathfinder learning module can output a graph defined over a set of nodes with a single edge weight for each edge; as a result, the pathfinder learning module is easily applied to any neural models that use the graph directly (including *Spectral Graph Convolutions* [92], *Graph Sampling and Aggregation* [71, 239], *Multi-Scale Graph Convolutions* [3, 5], *Clustered Graph Convolutions* [28], *Personalized Propagation of Neural Predictions* [18, 93] and *Simplified Graph Convolutions* [222]). We further note that pathfinder learning modules can be used as a building block for learning tasks beyond node prediction. For example, using the pathfinder module, an appropriate graph convolutional layer, and graph level pooling such as *Sort Pooling* or *Diff Pooling* [240, 247], one can easily define models which characterize or classify whole graphs.

5.3 Message passing on learned graphs

Our model jointly learns a single graph from a set of similarity graphs, and a graph neural network which uses the adjacency matrix of this learned graph as a propagation matrix. The adjacency matrices describing the input graphs themselves can be learned or pre-computed. An exemplar graph could be a set of k-nearest neighbor graphs of pairwise similarities calculated from multimodal datapoints, with separate graphs for images, sound, and text. Another potential example could be the use of normalized adjacency matrix powers as measures of pairwise similarity between nodes. In general, we can include as much information as possible with the expectation that the PDN will find the optimal graph structure based on consideration of all feature correlations.

Pathfinder learning layers

Here we detail the design of *Pathfinder Learning Layers* – neural network architectures for combining different kinds of proximity data together. We begin with the consideration of a simple model for combining proximity information, and extend it to support modeling complex relationships over multiplex data.

Definition 5.1. *Pathfinder Neuron.* A pathfinder neuron (Fig 5.2a) takes weighted adjacency matrices $\widetilde{\boldsymbol{A}}_1, \ldots, \widetilde{\boldsymbol{A}}_D$ as input and combines them into a single $|V| \times |V|$ learned graph $\widetilde{\mathbf{G}}$ as

its output. It uses trainable weights to learn the relative importance of each kind of similarity information, as follows:

$$\widetilde{\boldsymbol{G}} = \sigma\left(\sum_{i=1}^{D} \beta_i \cdot \widetilde{\boldsymbol{A}}_i\right). \tag{5.2}$$

The elementwise function $\sigma(\cdot)$ is a non-linearity and β_i is a trainable weight specific to the i^{th} input adjacency matrix (see Figure 5.2a). We assume that there is no bias term present, which implies: (i) calculating $\widetilde{\mathbf{G}}$ can be done entirely with sparse linear algebra operations; (ii) the output graph from one pathfinder neuron can be used as an input to other pathfinder neurons. Further, we note that multiple pathfinder neurons can take in the same inputs and learn different weights, akin to the GAT multi-attention-head.

Definition 5.2. *Pathfinder Layer.* A pathfinder layer uses multiple pathfinder neurons as building blocks for a more complex neural model. The l^{th} *pathfinder layer* with q neurons using p input graphs can be written by:

$$\widetilde{\mathbf{G}}_{l+1,1}; \ldots; \widetilde{\mathbf{G}}_{l+1,q} = f^l\left(\widetilde{\mathbf{G}}_{l,1}; \ldots; \widetilde{\mathbf{G}}_{l,p}\right). \tag{5.3}$$

Each $\widetilde{\mathbf{G}}$ is output by a single pathfinder neuron. The number of parameters in the neuron depends on the number of pathfinder graphs in the previous layer.

Definition 5.3. *Pathfinder Graph.* The final output of a pathfinder neuron, or a series of pathfinder layers in a pathfinder discovery network is the *pathfinder graph*, denoted by $\hat{\mathbf{G}}$.

The pathfinder graph can be used for message passing in an arbitrary downstream graph convolutional model (see Figure 5.2b). If the edge sets of input graphs sufficiently overlap and the original graphs are sparse, we expect $\hat{\mathbf{G}}$ to be sparse.

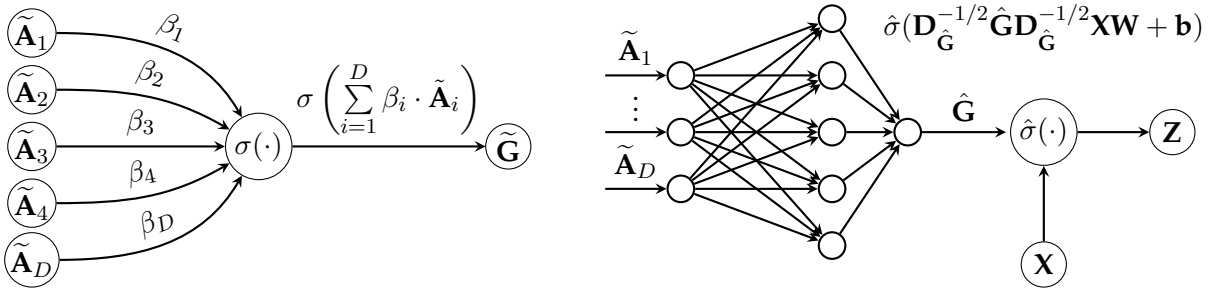
Pathfinder discovery networks

The pathfinder graph described above can be used as an input for an arbitrary downstream graph neural network. As a motivating example, consider the spectral graph convolutional network defined by Equation 5.1. We can augment this equation by replacing $\mathbf{A}$ with the final pathfinder graph:

$$\mathbf{Z} = \hat{\sigma}(\mathbf{D}_{\hat{\mathbf{G}}}^{-1/2}\hat{\mathbf{G}}\mathbf{D}_{\hat{\mathbf{G}}}^{-1/2}\mathbf{X}\mathbf{W}) \tag{5.4}$$

Definition 5.4. *Pathfinder Discovery Networks.* This general combined design of a message passing model and pathfinder layers (or neurons) is a *Pathfinder Discovery Network*. An instance where the the pathfinder network has a single hidden layer is depicted in Figure 5.2b.

While we focus on specific applications in this work, we note that PDNs can be used with most graph neural network models and objectives (both supervised and unsupervised).



(A) Architecture of a single pathfinder neuron

(B) A pathfinder layer and spectral graph convolutional neural network combined architecture

FIGURE 5.2: A single pathfinder neuron (5.2a) and a pathfinder discovery network (5.2b) with multiple pathfinder neurons in a single hidden layer. The pathfinder graph $\tilde{\mathbf{G}}$ output by the pathfinder module is used by some graph convolutional layer $\hat{\sigma}(\cdot)$. We illustrate this here using the GCN model [92], showing how a learned graph can be normalized.

5.4 Advantages of PDNs

As noted by [69] and others [36, 223], the performance of graph learning systems can vary greatly based on the quality of the network used. In this section we compare methods based on the popular graph attention network (GAT) model with PDNs. While GAT is not explicitly motivated by the problem of graph construction, we note that in the current literature the GAT approach can be viewed as attempting to learn a graph jointly with a deep learning task. However, GAT and related attention models have the following critical weaknesses that make them ill-suited for graph building:

1. GAT models learn a single aggregation over a single source of features, which significantly constrains the expressivity of the graph it can learn.

2. The GAT framework is heavily dependent on multiple attention heads for regularization. Unfortunately, the increase in parameters results in overfitting, which raises issues when training on real world datasets [181, 216].

PDNs can mitigate these weaknesses.

Expressivity

PDNs are designed from the ground up to handle an arbitrary number of modalities, each defined as a similarity measure over the vertices. The pathfinder network is able to combine these similarity measures in arbitrary ways.

Exclusive OR

As a motivating example, consider an XOR operation. In a multiplex graph setting, an XOR can describe a case where the presence of two edges together has a different semantic meaning than the presence of either edge separately.
Proposition. PDNs can learn XOR operations over different layers of a multiplex graph.

TABLE 5.1: The PDN model is able to learn complex relationships over multiplex edges. Here we show how an XOR relationship can be learned over two networks ($\mathbf{A}'_{u,v}$ and $\mathbf{A}''_{u,v}$).

Edge State		PDN activations		
$\mathbf{A}'_{u,v}$	$\mathbf{A}''_{u,v}$	h_1	h_2	$\alpha_{u,v}$
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	2	1	0

Proof. Let us consider a node classification problem with two component binary valued edge weight vectors $(\mathbf{A}'_{u,v}, \mathbf{A}''_{u,v})$ on each edge. Further, consider a PDN composed of a hidden layer with 2 neurons, described by these equations:

$$\begin{aligned} h_1 &= \text{ReLU}(\mathbf{A}'_{u,v} + \mathbf{A}''_{u,v}) \\ h_2 &= \text{ReLU}(\mathbf{A}'_{u,v} + \mathbf{A}''_{u,v} - 1) \\ \alpha_{u,v} &= h_1 - 2 \cdot h_2 \end{aligned}$$

As shown in Table 5.1, this network reproduces the *exclusive-or* function. We also include the hidden states (h_1, h_2) and predicted edge weights $(\alpha_{u,v})$ obtained with this pathfinder layer. ■

In practice without feature engineering GCN and GAT models can only utilize one of the edge features or the edge existence as an edge weight. Such weights on their own cannot separate different edge types.

Different edge weight semantics

Similar analysis can be done when considering the case where a node has two edge types where one edge denotes similarity and one denotes distance. A PDN can correctly learn to invert the distance edge, producing a single similarity measure that incorporates the full range of provided information. Because GAT uses a softmax aggregation, it cannot learn inversions; the best it can do is ignore the distance edge. The standard GCN treats all edges as either similarity or distance, and so will end up misinterpreting the information being provided by one of the two edge types.

Resilience to skewed degree distributions

Many GNN implementations suffer when faced with nodes that have very large degrees. As shown by [120] the limiting behaviour of $\alpha_{u,v}$ in the graph attention (GAT) model [210] forces edges weights to 0 as the neighbourhood size of u increases:

$$\lim_{|N(u)|\to\infty} \alpha_{u,v} = \lim_{|N(u)|\to\infty} \frac{\exp(f_\theta(\mathbf{H}_{u,:}; \mathbf{H}_{v,:}))}{\sum_{w\in N(u)} \exp(f_\theta(\mathbf{H}_{u,:}; \mathbf{H}_{w,:}))} = 0. \tag{5.5}$$

This over-smoothing limits the effectiveness of GAT and similar models on real world graphs, where node degrees often follow a power law distribution [120]. By comparison, PDNs can score edge weights independently, and therefore do not incorrectly penalize high degree nodes.

Proposition. PDNs can be constructed such that high degree nodes do not drive edge weights to 0.

Proof. Let us consider an edge (u, v) of the undirected graph $\mathcal{G}$ used for message passing. Let us denote the message passing weight of this edge as $\alpha_{u,v}$. Moreover, let us assume that the edge has a two dimensional feature vector $(\mathbf{A}'_{u,v}, \mathbf{A}''_{u,v})$. Let us further define the behaviour of $\alpha_{u,v}$ in a PDN which has no hidden layer and has a linear activation function:

$$\lim_{|N(u)|\to\infty} \alpha_{u,v} = \beta_1 \cdot \mathbf{A}'_{u,v} + \beta_2 \cdot \mathbf{A}''_{u,v}. \tag{5.6}$$

In Equation 5.6 the β values are trainable parameters of the PDN. We see that the limiting behaviour of the PDN edge weight does not depend on the neighbourhood size. ■

High degree nodes in the GAT and GCN model will have a large number of edge weights close to zero. This results in poor quality neighbourhood representations which are not discriminative on the downstream task.

Edge-weight calculation time complexity

GAT implementations are heavily regularized through averaging of multiple attention heads; experimental results in [210] use 8 unique attention heads over all of the node features. This can result in significant overfitting for cases where there are few features, or extremely large matrix calculations for large feature spaces.

This has an impact on the runtime necessary to calculate each edge, both during training and during inference. Naive implementations of the GCN and GAT models which do not use a cache can calculate the weight $\alpha_{u,v}$ in $\mathcal{O}(|\mathcal{N}(v)|)$ time. By contrast the same edge weight can be calculated in $\mathcal{O}(1)$ time with a shallow PDN which has a linear activation function.

5.5 Variations on the basic model

In order to understand the core motivations behind PDNs, we have thus far limited our discussion to high level, general characteristics. In this section, we drill down to specific variations on the basic model to demonstrate the flexibility and expressive power of the proposed framework.

A model with learned similarities

We have mentioned in passing that one can design a PDN where the weights in the adjacency matrices describing the similarity graphs are themselves parametrized by neural networks. We have thus far assumed that the weights described by the adjacency matrices $\widetilde{\mathbf{A}}_1, \ldots, \widetilde{\mathbf{A}}_D$ are coming from pre-calculated similarities. Instead, let us assume that for each *binary* $\widetilde{\mathbf{A}}_1, \ldots, \widetilde{\mathbf{A}}_D$ we have a feature matrix $\mathbf{X}_i, \ldots, \mathbf{X}_D$. We can then define a graph convolutional model where the edge weights of an input graph are learned by node features. Let $\mathbf{H}_i$ be the node hidden representation matrix,

$$\mathbf{H}_i = \sigma(\mathbf{X}_i \cdot \mathbf{W}'_i + \mathbf{b}_i). \tag{5.7}$$

Here $\mathbf{X}_i$ is the i^{th} generic node feature matrix, the function $\sigma(\cdot)$ is an elementwise non-linearity, and $\mathbf{W}'_i$ and $\mathbf{b}_i$ are the feature matrix specific trainable weight matrix and bias vector. Using the

endpoint representations we define $\hat{\mathbf{G}}_i$ as a learned input adjacency matrix for a graph learning neuron:

$$\hat{\mathbf{G}}_i = \mathbf{A}_i \odot \hat{\sigma}(\mathbf{H}_i \cdot \mathbf{H}_i^{\top}). \tag{5.8}$$

We use the elementwise non linearity $\hat{\sigma}(\cdot)$ to transform the raw edge weights which are conditioned on the original adjacency matrix by a Hadamard product. Exploiting the similarity of the individual adjacency matrices the calculation of 5.8 happens in $\mathcal{O}(|E|)$.

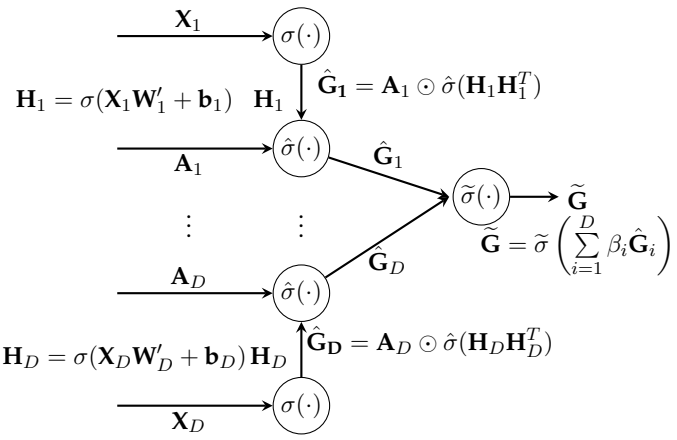


FIGURE 5.3: The pathfinder neuron design with learned similarity scores. From each node feature matrix conditioned by the corresponding adjacency matrix we learn a similarity graph. In the pathfinder neuron we learn to combine these together as a single learned graph denoted by $\widetilde{\mathbf{G}}$. This output graph can serves as the input for an arbitrary downstream graph convolutional layer.

A pathfinder neuron receives multiple learned graphs as input, combines those and outputs a final graph. This idea is summarized by Figure 5.3 where we have D different feature matrices and from each of them we learn a separate graph that we use as input for the pathfinder neuron, which in turn outputs $\widetilde{\mathbf{G}}$. This final aggregation is defined by Equation 5.9 in which β_i is a learned parameter that acts as a weight for the learned graphs and $\widetilde{\sigma}(\cdot)$ is a non-linearity.

$$\widetilde{\mathbf{G}} = \widetilde{\sigma}\left(\sum_{i=1}^{D} \beta_i \hat{\mathbf{G}}_i\right) \tag{5.9}$$

A model with cheap multi-scale mixing

Multi-scale graph neural network models obtain information about the neighbourhoods of nodes at multiple hops [5, 148, 165, 171] and learn features for each hop. Most graph neural networks [28, 71, 92, 93, 224] which are not multi-scale (with the exception being *AttentionWalk* [4] and *DCRNN* [115]) pool features from neighbourhoods at different scales without considering what is the optimal mixing of information. In the following we will define a corner case of our model which allows for supervised and explainable pooling of multi-scale information with trainable weights.

Data: $\widetilde{\mathbf{A}}$ - Normalized adjacency matrix
$\mathbf{X}$ - Feature matrix
D - Order of adjacency matrix powers
d - Number of filters
Result: $\mathbf{Z}$ – Hidden state matrix
$\mathbf{Z} \leftarrow$ Initialize representations(d).
$\mathbf{Z}_0 \leftarrow \mathbf{XW}$
for $i \in \{1, \ldots, D\}$ **do**
  $\mathbf{Z}_i \leftarrow \widetilde{\mathbf{A}}\mathbf{Z}_{i-1}$
  $\mathbf{Z} \leftarrow \mathbf{Z} + P_i \cdot \mathbf{Z}_i$
end

Algorithm 5: Efficient sparsity aware forward pass multi-scale mixing with a softmax learned graph and a linear graph convolutional activation function.

Let $\widetilde{\mathbf{A}}$ be the normalized adjacency matrix of the weighted undirected graph $\mathbf{G}$. We assume that the similarity graphs of interest are described by powers of this normalized adjacency matrix for a given D number of hops – $\widetilde{\mathbf{A}}_i = \widetilde{\mathbf{A}}^i, \quad \forall i = 1, \ldots, D$. The learned graph used for the forward pass is defined as:

$$\widehat{\mathbf{G}} = \sum_{i=1}^{D} P_i \cdot \widetilde{\mathbf{A}_i} \tag{5.10}$$

where P_i is the weight of a given adjacency matrix power, and is parametrized with a softmax as $\exp(\alpha_i) / \left(\sum_{i=1}^{D} \exp(\alpha_i)\right)$. As the direct calculation of the adjacency matrix powers is prohibitive, we instead use an efficient forward pass algorithm to calculate the hidden state matrices described by Algorithm 5. The core idea is to exploit the sparsity of the adjacency matrix in each iteration by using the normalized adjacency to average node representations, and weighting the representations with learned P_i scores.

5.6 Experimental evaluation

Above, we theoretically motivated the development of PDNs by discussing the importance of jointly learning graphs and GNNs for specific tasks and evaluating the expressivity of the pathfinder learning module. In the following, we empirically validate our analysis by demonstrating that PDNs have a significant advantage on a class of graph learning tasks, while maintaining competitive predictive performance on other baselines. We also describe how weights in a pathfinder neuron can be interpreted as attention, and we analyze model runtime to discuss the scalability of our models.

Synthetic node classification experiment

PDNs are a natural fit for dealing with noisy node and edge features, because they can learn complex correlations across many different combined modalities of data while removing unimportant

information. To highlight this key advantage, we investigate node classification performance on synthetically generated datasets that are specifically designed with imperfect feature information.

Experimental settings

The default setting for synthetic graph generation are as follows: we generate graphs with $C = 3$ label classes and $n = 500$ nodes per class; we set edge probabilities to $P = 0.01$ and $Q = 0.005$; we set feature dimensions to $F = 32$ and $D = 32$; and we set feature correlations to $\sigma_F = 5.0$ and $\sigma_D = 2.0$ standard deviations. We modulate these hyperparameters in the experimental scenarios described in Table 5.2, where each scenario modifies a single parameter from the defaults described above. For each scenario, we used 80%/20% train-test splits, and report the average of 10 synthetic graph generation%/model training cycles.

For our synthetic experiments, we use GCN, GAT, and DeepWalk as baselines, with the hyperparameter settings described in [5]. To this set, we add the following baselines and corresponding hyperparameters:

- *AAPNP* [18, 93]: The feedforward component of the model has 32 filters and we did 10 personalized pagerank approximation iterations with a teleport probability of 0.2.
- *SGCONV* [222]: We used information from the 2^{nd} order proximity of the normalized adjacency matrix with 32 dimensional filters.
- *ClusterGCN* [28]: We used the settings of the *Spectral GCN* model on the graph pre-clustered by the METIS community detection algorithm [88].

By comparison, we construct a PDN with a single hidden layer containing 16 pathfinder neurons and a ReLU activation function [132] in the hidden layer, followed by a softmax activation function in the output layer of the pathfinder module. On top of the pathfinder module, we add a standard 2-hop spectral GCN [92] with a hidden layer dimension size of 32. All models, including PDNs, were trained using Adam [91] with a learning rate of 10^{-2}, over 200 training epochs. Where relevant, we used a dropout value of 0.5 and an l_2 weight regularization coefficient of 10^{-3}. All models were implemented using the PyTorch Geometric framework [47].

TABLE 5.2: Synthetic node classification scenarios with the range of the manipulated hyperparameters and specific implications of the modulation in the scenario.

Scenario	Parameter	Implication of increase
1	$C \in [2, 6]$	Less clear classes
2	$n \in [2^4, 2^{10}]$	More instances for generalization
3	$P \in [2^{-10}, 2^{-4}]$	Stronger class cohesion
4	$Q \in [2^{-10}, 2^{-4}]$	More inter-class edges
5	$F \in [2^2, 2^6]$	More node features
6	$D \in [2^2, 2^6]$	More edge features
7	$\sigma_F \in [2^{-1}, 2^5]$	Lower node feature quality
8	$\sigma_D \in [2^{-1}, 2^5]$	Easier separation of edge type

Synthetic data generation procedure

Each synthetic graph has C node label classes and n nodes in the graph belonging to a given class. These two hyperparameters decide the overall number of nodes in the synthetically generated graph, $C \times n$. We generate features as follows:

1. *Generation of correlated node features.* For each node we generate d_N continuous node features which are standard normally distributed with a pre-defined correlation structure. The eigenvalues of the node feature correlation matrix are distributed proportional to a standard half-normal distribution. This ensures that the eigenvalues of the generated correlation matrix are positive.

2. *Generation of node labels.* The node feature matrix $\mathbf{X}$ is multiplied by a normally distributed F dimensional weight vector $\mathbf{w}$ which results in a continuous node target feature $\mathbf{y}$. We add zero mean normally distributed noise to this target vector with standard deviation σ_F which results in the noisy target vector $\widetilde{\mathbf{y}}$. We quantile bin the continuous target vector to get a label vector for the node classification task with C distinct classes.

3. *Edge addition.* We define two edge types in our graph: intra-class edges (those edges between nodes that share a class); and inter-class edges (the opposite). An edge exists between two intra-class nodes with probability P, while an edge exists between two inter-class nodes with probability Q.

4. *Generation of edge features.* For each edge we generate D continuous edge features which are normally distributed and uncorrelated. Inter-class edge features have a standard deviation of σ_D while intra-class edge features are distributed according to the standard normal distribution. This allows us to tune how much information can be propagated from the edges themselves.

Findings and discussion

The mean accuracy scores for each scenario are shown in Figure 5.4. PDN materially outperforms the baselines for a wide range of synthetic graphs. The results of Scenario 1 demonstrate that PDN is able to distinguish between less clearly defined classes, while competing graph neural networks struggle to maintain competitive performance. We highlight data efficiency in Scenario 2 – given a fixed number of instances, PDN generalizes better to unseen data where $n \geq 2^6$. In Scenario 3, we observe that stronger class-cohesion results in better classification performance for all models, and that PDN displays superior marginal predictive performance gain. As one can see ClusterGCN uses a pre-processing step which is purely topological and this filters out inter-class edges. Increasing the number of inter-class edges in Scenario 4 initially decreases the predictive performance of the baselines; by comparison, PDN is able to learn to ignore the noise propagating inter-class edges. Scenario 5 shows that all supervised models gain when more vertex features are available, and again PDN displays superior marginal performance gain. On the contrary, we see that the PDN overfits when a large number of edge features is available based on Scenario 6. Though all models are sensitive to node features, Scenario 7 shows that PDNs are significantly

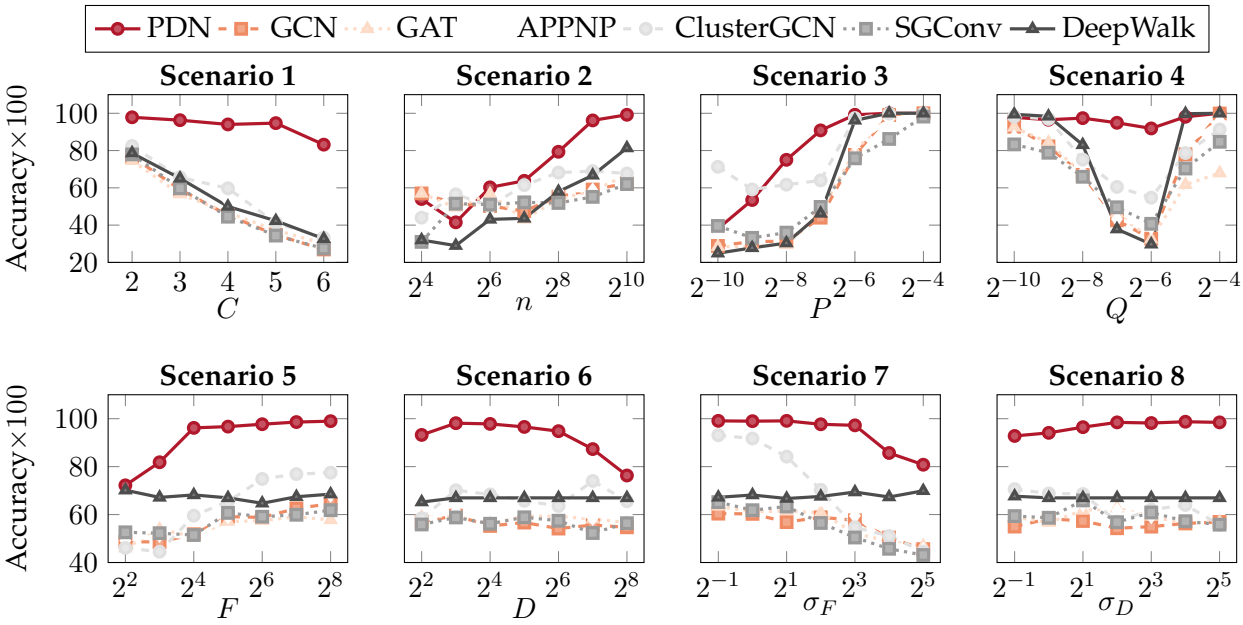


FIGURE 5.4: Node classification performance measured by average test set accuracy (10 experimental repetitions) on the synthetically generated attributed graphs for the scenarios described in Table 5.2. The proposed Pathfinder Discovery Network architecture has robust predictive performance under a wide range of synthetic data generation hyperparameters.

more resilient to node feature corruption. Finally, in Scenario 8, higher quality edge features only help PDN.

We briefly want to focus on the results in Scenario 4, as we believe this demonstrates the XOR functionality of the PDN. The GCN baseline models learn from the expected value of neighboring hidden states. In high homophily graphs (i.e. the low Q region), neighboring states will correlate with node features resulting in high performance. The same holds for low homophily graphs (i.e. the high Q region), except the weights are inverted – in other words, the baseline GCN models will learn to simply invert neighboring node states. In the middle, the baseline models cannot learn a single aggregation that correctly handles the differing edge information. By contrast, PDNs are expressive enough to learn to differentiate the edge weights, allowing it to maintain high performance throughout.

Implicit learning of inter- and intra-class edges

We perform a visual embedding analysis in Figure 5.5, where we examine the 2 dimensional t-SNE embeddings [207, 208] of hidden layer edge representations for the PDN and GAT models. The representations extracted from the PDN show distinct separation for the inter and intra-class edges, which implies that the model has learned to meaningfully distinguish these two modalities of information. By contrast, the GAT representations are not separated by the type of the edge. This further demonstrates the high expressivity of PDNs.

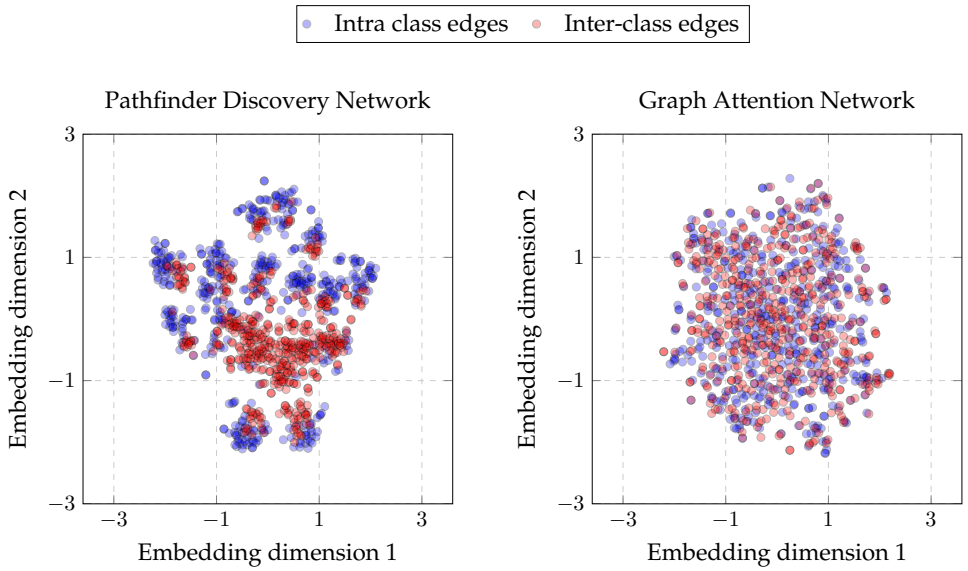


FIGURE 5.5: The t-SNE embedding of PDN and GAT hidden layer edge representations on the synthetic dataset with the standard data generation settings. The PDN model is able to separate inter-class and inter-class edges better.

Node classification performance

Excitingly, PDNs are quite capable in the multiplex graph settings. That said, we want to ensure PDNs maintain high performance in traditional single graph settings. Further, we believe that joint training of the pathfinder and classifier will lead to lift even on well established problems. We therefore evaluate the node classification performance of our proposed model variants on widely used citation graphs [118, 133] and social networks [165, 171]. The descriptive statistics of these datasets are in Table 5.3.

TABLE 5.3: Descriptive statistics of the attributed citations graphs and social networks used for node classification performance evaluation and comparison in our work.

Dataset	Nodes	Clustering Coefficient	Density	Unique Features	Classes
Cora	2,708	0.094	0.002	1,432	7
Citeseer	3,327	0.130	0.001	3.703	6
Pubmed	19,717	0.054	0.001	500	3
Facebook Page-Page	22,470	0.232	0.001	4,714	4
Deezer Europe	28,281	0.096	0.001	31,240	2

Experimental settings

Because PDNs are general, we included a wide variety of unsupervised and supervised baselines to best understand relative performance. We evaluated proximity preserving node embedding techniques [66, 139, 147, 154, 166, 170, 198], including multi-scale methods [22, 148]. We also included a range of attributed node embedding methods [227, 234] and attributed methods that incorporate node attribute information from multiple hops [231]. Each of the upstream node embeddings was trained with the default hyperparameter settings of the Karate Club package

[168] – 128 dimensional node embeddings which have a comparable number of free parameters. The downstream model was an l_1 regularized multinomial logistic regression (softmax) classifier pulled from scikit-learn [145].

TABLE 5.4: Tie strength scoring functions for edge $(u, v) \in E$ used as edge features of the Pathfinder Discovery Networks.

Name	Definition
Adamic-Adar	$\sum_{w \in N(u) \cap N(v)} \frac{1}{\log \lvert N(w) \rvert}$
Association Strength	$\frac{\lvert N(u) \cap N(v) \rvert}{\lvert N(u) \rvert \cdot \lvert N(v) \rvert}$
Common Neighbors	$\lvert N(u) \cap N(v) \rvert$
Cosine	$\frac{\lvert N(u) \cap N(v) \rvert}{\sqrt{\lvert N(u) \rvert \cdot \lvert N(v) \rvert}}$
Degree Product	$\lvert N(u) \rvert \cdot \lvert N(v) \rvert$
Jaccard	$\frac{\lvert N(u) \cap N(v) \rvert}{\lvert N(u) \cup N(v) \rvert}$
Max Overlap	$\frac{\max(\lvert N(u) \rvert, \lvert N(v) \rvert)}{\lvert N(u) \cap N(v) \rvert}$
Min Overlap	$\frac{\min(\lvert N(u) \rvert, \lvert N(v) \rvert)}{\lvert N(u) \cap N(v) \rvert}$
N-Measure	$\frac{\sqrt{2} \lvert N(u) \cap N(v) \rvert}{\sqrt{\lvert N(u) \rvert^2 + \lvert N(v) \rvert^2}}$
Pearson Correlation	$\frac{\lvert V \rvert \cdot \lvert N(u) \cap N(v) \rvert - \lvert N(u) \rvert \cdot \lvert N(v) \rvert}{\sqrt{\lvert V \rvert \cdot \lvert N(u) \rvert - \lvert N(u) \rvert^2} \cdot \sqrt{\lvert V \rvert \cdot \lvert N(v) \rvert - \lvert N(v) \rvert^2}}$
Resource Allocation	$\sum_{w \in N(u) \cap N(v)} \frac{1}{\lvert N(w) \rvert}$

For supervised baselines, we used GNN hyperparameter settings, training setup, and citation graph results from [5], specifically the performance of the two layer feedforward neural network, *Chebyshev GCN* [38], *Spectral GCN* [92] and *GAT* [210]. For comparison, we examine three PDNs: the basic Pathfinder Discovery Network, the PDN EdgeConv method described in 5.5, and the PDN Multi-Scale method described in 5.5. The default PDN has a single hidden layer with 16 pathfinder neurons and uses the neighbourhood similarity metrics [6, 43, 137, 249] described in Table 5.4 as input features. In addition, our edge convolutional model uses information from the 1^{st} and 2^{nd} hop, while the cheap multi-scale model uses information up to the 2^{nd} order proximity. We use the hyperparameters and optimizer settings from the synthetic experiments. All models were trained on a 100-shot learning experiment where we calculated the average node classification accuracy on the test set based on 10 seeded train-test splits.

Findings and discussion

We report the standard deviation of the mean accuracy estimates in Table 5.5. Our results demonstrate that PDNs outperform unsupervised methods by between 2.5 and 16.5 % in terms of accuracy. Against all approaches, including supervised approaches, PDN variants are the most competitive models on the Cora, Pubmed, Facebook, and Deezer benchmarks, with a relative accuracy advantage between 0.8 and 3.5%. PDNs fall behind only the standard GCN model on the Citeseer benchmark.

TABLE 5.5: Average node classification test accuracy results of 100-shot learning runs calculated from 10 experimental runs (standard deviations around the mean below the accuracy) on citation graph datasets and social networks. Bold red numbers denote the best performing model.

Model	Citeseer	Cora	Pubmed	Facebook Pages	Deezer Europe
LINE$_2$ [198]	0.470 ±0.013	0.686 ±0.013	0.675 ±0.017	0.762 ±0.010	0.503 ±0.005
DeepWalk [147]	0.523 ±0.010	0.762 ±0.011	0.704 ±0.014	0.531 ±0.012	0.510 ±0.007
Walklets [148]	0.513 ±0.010	0.735 ±0.010	0.675 ±0.017	0.819 ±0.011	0.511 ±0.008
GraRep [22]	0.421 ±0.027	0.634 ±0.016	0.653 ±0.018	0.705 ±0.008	0.507 ±0.008
HOPE [139]	0.397 ±0.041	0.717 ±0.026	0.561 ±0.035	0.593 ±0.027	0.508 ±0.029
NetMF [154]	0.446 ±0.030	0.707 ±0.009	0.710 ±0.012	0.756 ±0.015	0.512 ±0.010
AANE [81]	0.691 ±0.009	0.760 ±0.009	0.801 ±0.010	0.652 ±0.008	0.621 ±0.007
ASNE [116]	0.589 ±0.015	0.758 ±0.011	0.738 ±0.017	0.636 ±0.010	0.608 ±0.011
MUSAE [165]	0.636 ±0.012	0.758 ±0.011	0.784 ±0.004	0.822 ±0.010	0.563 ±0.010
TADW [227]	0.657 ±0.008	0.644 ±0.009	0.765 ±0.004	0.536 ±0.012	0.558 ±0.010
BANE [231]	0.566 ±0.015	0.743 ±0.015	0.729 ±0.019	0.648 ±0.011	0.517 ±0.009
TENE [234]	0.658 ±0.010	0.662 ±0.011	0.775 ±0.009	0.598 ±0.016	0.593 ±0.022
FEATHER [171]	0.649 ±0.012	0.805 ±0.010	0.769 ±0.015	0.854 ±0.039	0.539 ±0.007
2-Layer MLP	0.706 ±0.010	0.690 ±0.011	0.783 ±0.005	0.761 ±0.010	0.565 ±0.016
Chebyshev [38]	0.742 ±0.005	0.855 ±0.004	0.818 ±0.005	0.838 ±0.009	0.564 ±0.008
GCN [92]	**0.767 ±0.004**	0.861 ±0.003	0.822 ±0.003	0.854 ±0.007	0.545 ±0.008
GAT [210]	0.748 ±0.004	0.830 ±0.010	0.818 ±0.002	0.839 ±0.008	0.532 ±0.009
SGConv [222]	0.699 ±0.013	0.850 ±0.005	0.796 ±0.010	0.762 ±0.005	0.536 ±0.006
ClusterGCN [28]	0.708 ±0.006	0.836 ±0.007	0.819 ±0.005	0.817 ±0.010	0.558 ±0.005
GraphSAGE [71]	0.706 ±0.008	0.840 ±0.009	0.803 ±0.007	0.846 ±0.009	0.554 ±0.006
PDN	0.764 ±0.010	**0.868 ±0.008**	0.835 ±0.004	**0.875 ±0.010**	**0.584 ±0.010**
PDN EdgeConv	0.711 ±0.008	0.864 ±0.008	0.833 ±0.011	0.863 ±0.009	0.548 ±0.007
PDN Multi-Scale	0.740 ±0.007	0.866 ±0.009	**0.836 ±0.013**	0.793 ±0.009	0.568 ±0.009

Relative runtime

The time complexity of training a traditional spectral graph convolutional networks is $\mathcal{O}(|E|F)$ while a Pathfinder Discovery Network has a time complexity of $\mathcal{O}(|E|(F+D))$. Using synthetic data, we compare the relative runtime of PDNs in a number of scenarios to provide a better empirical understanding of what the additional time complexity means in practice.

We generate Watts-Strogatz graphs [220] with 10^{12} nodes, 2^4 edges per node and a rewiring probability of 0.5. In addition, we sample $F = 2^7$ node and $D = 2^7$ edge features using Gaussians and draw labels for the nodes from $C = 4$ classes uniformly. We calculate the average epoch runtime for a spectral GCN [92], a generalized linear PDN, a shallow PDN with $\{32\}$ neurons, and a deep PDN with $\{32, 16\}$ neurons in the hidden layers.

Findings and discussion

The relative runtime is shown in Figure 5.6. The results are in line with the runtime complexities discussed above: increasing the number of edges does not increase the relative runtime of the PDNs, but increasing the edge feature count does increase the relative runtime. We also see that more complex (deeper) edge aggregation models are slower.

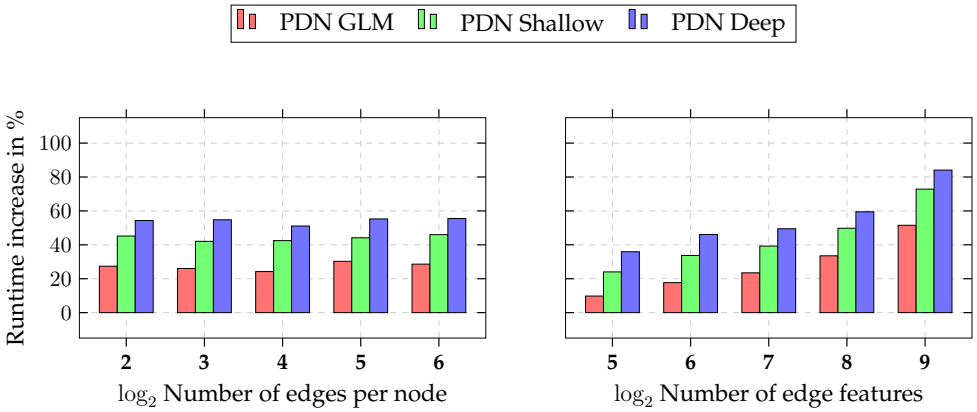


FIGURE 5.6: The relative runtime increase (compared to spectral graph convolutions) needed for training PDN models on synthetic datasets. Increasing the number of edges does not change the relative runtime, but a higher edge feature count results in longer relative training times.

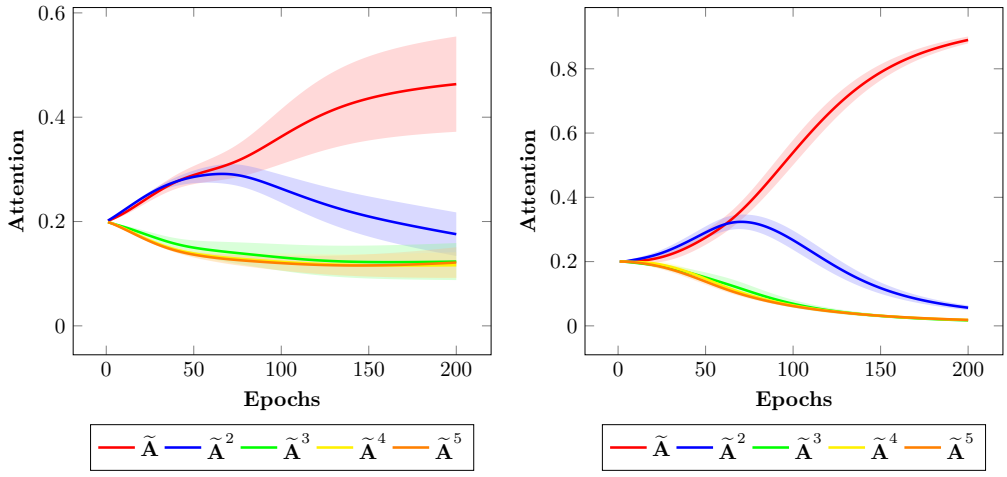
Edge feature importance

Model interpretability is an important part of developing deep neural networks. Architectures with interpretable weights can provide novel insights on the structure of data, while also making validation, inspection, and debugging significantly easier. We believe that PDNs can add significant interpretability to graph learning tasks when we frame the learned weights as an attention mechanism over the input graphs. In this set of experiments we discuss two scenarios when learned PDN weights have direct interpretations.

Attention on proximity

In this experiment, we use the multi-scale model described in Section 5.5. We utilize the first 5 normalized adjacency matrix powers as input similarity graphs and apply the hyperparameters described in Section 5.6. We train this model on 100-shot learning tasks, and report the mean weight for each adjacency power from 100 repetitions (see Figure 5.7).

Based on these learned weights, we observe that the model has learned to prioritize messages that come from the first order neighbourhoods of vertices – in other words, the PDN attends to closer neighbors more. We note that the Cora and Pubmed graphs exhibit high homophily between nodes, suggesting that the model's weighting scheme is well motivated. Interestingly, we also observe that the importance of information coming from the second hop starts to decline between 50-100 epochs; not coincidentally, this is around when peak test accuracy is reached, after which we observe a decline (test accuracy not shown). This implies that graph neural network models overfit to information coming from the first order proximity of individual data points.



(A) Cora (B) Pubmed

FIGURE 5.7: The change of the attention weights as training progresses in a single neuron PDN trained on the Cora and Pubmed datasets.

More importantly, the added interpretability from PDN allows us to observe exactly where the overfitting is occurring.

Attention on neighbourhood similarity

Using the similarity scores listed in Table 5.1 we train a Linear PDN model with the hyperparameter settings described in Subsection 5.6. As a reminder, this implementation of the pathfinder module uses softmax activations and does not have a hidden layer – in this setting, the weights of the pathfinder module can be interpreted as attention. For the citation graph and social network datasets we plot the average attention score (calculated from 10 training runs) for a selected subset of edge scores in Figure 5.8.

The results show that unnormalized edge similarity scores such as the *degree product* and *common neighbours* tend to receive low attention when the edge weight aggregation happens in the pathfinder module. On most datasets similarity metrics which are normalized and do not consider the degree of shared neighbors (e.g. *association strength* and *minimal overlap*) receive relatively high attention.

Multiplex node classification performance

We evaluated the predictive performance of PDNs on real world node classification problems using publicly available multiplex webgraph datasets [142]. The descriptive statistics of these graph datasets are presented Table 5.6. We would like to point out that the layer wise characteristics of the networks are remarkably different for these two datasets.

TABLE 5.6: Descriptive statistics of the multiplex webgraphs (individual layers) used for node classification performance evaluation and comparison in our work.

Dataset	Layers	Nodes	Density	Clustering Coefficient	Unique Features	Classes
IMDB	2	3550	0.005 0.001	0.509 1.000	2000	2
AMC	2	3025	0.242 0.004	1.000 0.687	1870	2

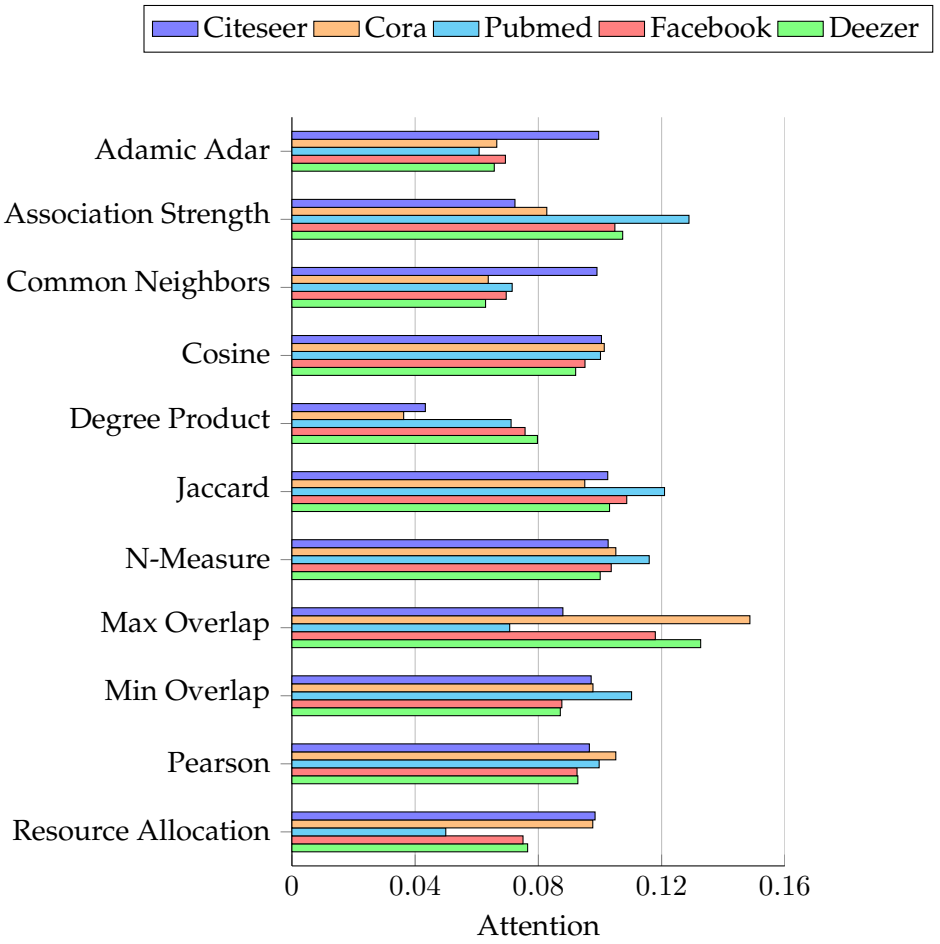


FIGURE 5.8: Comparison of average PDN attention scores (single neuron pathfinder module) on edge similarity scores for the real world datasets.

Multiplex node classification experimental settings

We created 10 seeded 100-shot learning splits for evaluation, because of this the mean performance metrics are comparable across models as there is no variation coming from the splits. The PDN had a single hidden layer with 2 neurons, the other hyperparameters were the same as the ones described earlier. The other supervised reference models Multi-GCN [90] and DMGI [142] used the default hyperparameters from the experimental section of the respective research papers. The evaluation of the unsupervised techniques MVN2Vec [183], MELL [127] and MNE [246] used a two stage upstream and downstream learning setup. First, we trained embeddings with hyperparameters from the original papers. Second, we trained a scikit-learn [145] logistic regression on the embedding features using the default settings.

Findings and discussion

The average test accuracy scores are plotted on Figure 5.9 with standard deviations around the mean. Our results demonstrate that PDNs significantly outperform the competing supervised and unsupervised multiplex graph representation learning techniques on these datasets in terms of test accuracy. It is also evident that supervised learning methods have a considerable performance advantage over the unsupervised ones.

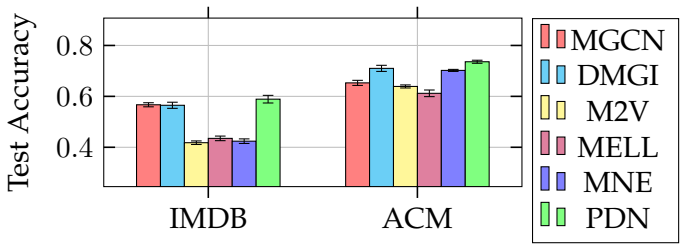


FIGURE 5.9: Average multiplex 100-shot node classification test accuracy results calculated from 10 experimental runs (the error bars standard deviations around the mean) on mutiplex graph benchmark datasets.

5.7 Conclusion and future directions

In this chapter we proposed pathfinder discovery networks (PDN), a graph neural network architecture for learning a message passing graph from a multiplex graph defined on a fixed set of nodes. Our modular architecture allows for joint training of a graph neural network and the pathfinder discovery layer, which in turn allows practitioners to find the optimal message passing graph for a specific supervised task. We examine the comparative characteristics of PDNs, concluding that PDNs are significantly more expressive and resilient than existing approaches. We then describe general extensions of our model, which allow for the definition of multi-scale graph convolutional layers and edge convolutions without edge features.

In our empirical analysis, we establish that PDNs have competitive predictive performance on various node classification tasks. We showed that the relative runtime increase of PDNs is independent of the dataset size in terms of edge set cardinality. And finally, we examined the weights of the graph aggregation model from the lens of learned attention.

We believe there are many exciting areas of future work. We are particularly excited about the possibility of extracting the PDN-learned graph for use in other tasks. We intuit that it would be possible to learn several PDN graphs for many different kinds of supervised tasks, and then combine *those* graphs in another PDN. We believe that this yet-unexplored use case will be very important in improving abstract notions of 'graph accuracy' for a wide range of datasets, while simultaneously opening new areas of transfer learning on graphs.

Chapter 6

The Shapley Value of Classifiers in Ensemble Games

What is the value of an individual model in an ensemble of binary classifiers? We answer this question by introducing a class of transferable utility cooperative games called *ensemble games*. In machine learning ensembles, pre-trained models cooperate to make classification decisions. To quantify the importance of models in these ensemble games, we define *Troupe* – an efficient algorithm which allocates payoffs based on approximate Shapley values of the classifiers. We argue that the Shapley value of models in these games is an effective decision metric for choosing a high performing subset of models from the ensemble. Furthermore, we propose the *Shapley entropy* as a way to quantify the heterogeneity of ensembles with respect to model quality. Our analytical findings prove that our Shapley value estimation scheme is precise and scalable; its performance increases with size of dataset and ensemble. Empirical results on real world graph classification tasks demonstrate that our algorithm produces high quality estimates of the Shapley value. We find that Shapley values can be utilized for ensemble pruning, and that adversarial models receive a low valuation. Complex classifiers are frequently found to be responsible for both correct and incorrect classification decisions.

6.1 Introduction

The advent of black box machine learning models raised fundamental questions about how input features and individual training data points contribute to the decisions of expert systems [55, 119, 172]. There has also been interest in how the heterogeneity of models in an ensemble results in heterogeneous contributions of those to the classification decisions of the ensemble [54, 206]. For example one would assume that computer vision, credit scoring and fraud detection systems which were trained on varying quality proprietary datasets output labels for data points with varying accuracy. Another source of varying model performance can be the complexity of models e.g. the number of weights in a neural network or the depth of a classification tree.

Quantifying the contributions of models to an ensemble is paramount for practical reasons. Given the model valuations, the gains of the task can be attributed to specific models, large ensembles can be reduced to smaller ones without losing accuracy [100, 125] and performance heterogeneity of ensembles can be gauged [54]. This raises the natural question: How can we

measure the contributions of models to the decisions of the ensemble in an efficient, model type agnostic, axiomatic and data driven manner?

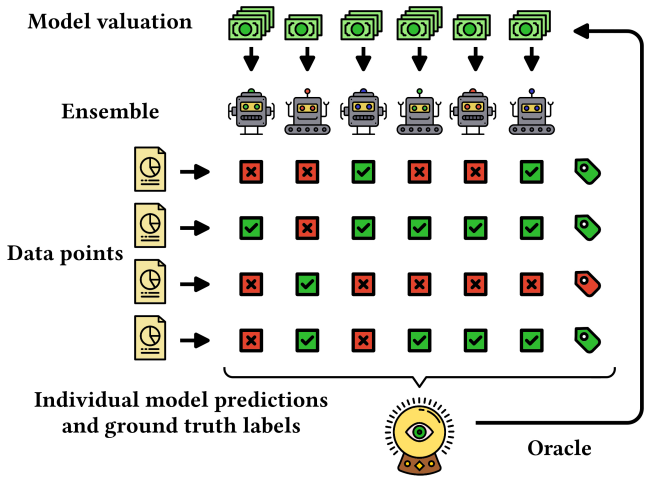


FIGURE 6.1: An overview of the model valuation problem. Models in the ensemble receive a set of data points and score those. Using the predictions and ground truth labels the oracle quantifies the worth of models.

We frame this question as one of valuation of models in an ensemble. The solution to the problem requires an analytical framework to assess the worth of individual classifiers in the ensemble. This idea is described in Figure 6.1. Each classifier in the ensemble receives the data points, and they output for each data point a probability distribution over the potential classes. Using these propensities an oracle – which has access to the ground truth – quantifies the worth of models in the ensemble. These importance metrics can be used to make decisions – e.g. pruning the ensemble and allocation of payoffs.

Present work. We introduce *ensemble games*, a class of transferable utility cooperative games [135]. In these games binary classifiers which form an ensemble play a voting game to assign a binary label to a data point by utilizing the features of the data point. Building on the ensemble games we derive *dual ensemble games* in which the classifiers cooperate in order to misclassify a data point. We do this to characterize the role of models in incorrect decisions.

We argue that the Shapley value [180], a solution concept from cooperative game theory, is a model importance metric. The Shapley value of a classifier in the ensemble game defined for a data point can be interpreted as the probability of the model becoming the pivotal voter in a uniformly sampled random permutation of classifiers. Computing the exact Shapley values in an ensemble game would take factorial time in the number of classifiers. In order to alleviate this we exploit an accurate approximation algorithm of the individual Shapley values which was tailored to voting games [46]. We propose *Troupe*, an algorithm which approximates the average of Shapley values in ensemble games and dual games using data.

We utilize the average Shapley values as measures of model importance in the ensemble. The Shapley values are interpretable as an importance distribution over classifiers, hence the information entropy of this distribution is a straightforward measure of ensemble heterogeneity

when it comes to model quality. Using the newly introduced *Shapley entropy* we are able to quantify heterogeneity with respect to correct and incorrect decisions.

We evaluate *Troupe* by performing various classification tasks. Using data from real world webgraphs (Reddit, GitHub, Twitch) we demonstrate that *Troupe* outputs high quality estimates of the Shapley value and the ensemble heterogeneity metrics. We validate that the Shapley value estimates of *Troupe* can be used as a decision metric to build space efficient and accurate ensembles. Our results establish that more complex models in an ensemble have a prime role in both correct and incorrect decisions.

Main contributions. Specifically the contributions of our work can be summarized as:

1. We propose ensemble games and their dual games to model the contribution of individual classifiers to the decisions of voting based ensembles.
2. We design *Troupe* an approximate Shapley value based algorithm to quantify the role of classifiers in decisions.
3. We define ensemble heterogeneity metrics using the entropy of approximate Shapley values outputted by *Troupe*.
4. We provide a probabilistic bound for the approximation error of average Shapley values estimated from labeled data.
5. We empirically evaluate *Troupe* for model valuation and forward ensemble building on graph classification tasks.

The rest of this work has the following structure. In Section 6.2 we discuss related work on the Shapley value, its approximations and applications in machine learning. We introduce the concept of ensemble games in Section 6.3 and discuss Shapley value based model valuation in Section 6.4 with theoretical results. We evaluate the proposed algorithm experimentally in Section 6.5. We summarize our findings in Section 6.6 and discuss future work. The reference implementation of *Troupe* is available at `https://github.com/benedekrozemberczki/shapley`.

6.2 Related work

The *Shapley* value [180] is a solution to the problem of distributing the gains among players in a transferable utility cooperative game [135]. It is widely known for its desirable axiomatic properties [25] such as *efficiency* and linearity. However, exact computation of Shapley value takes factorial time, making it intractable in games with a large number of players. General [122, 140] and game specific [46] approximation techniques have been proposed. In Table 6.1, we compare various approximation schemes with respect to certain desired properties.

Shapley values can be approximated using a Monte Marlo *MC* sampling of the permutations of players, and a truncated sampling *TMC* [122, 123, 251]. A more tractable approximation is proposed in [140], using a multilinear extension (*MLE*) of the Shapley value. A variant of this technique [103, 104] calculates the value of large players explicitly and applies the *MLE* technique to small ones. The only approximation technique tailored to weighted voting games is

the expected marginal contributions method (*EMC*) which estimates the Shapley values based on contributions to varying size coalitions. Our proposed algorithm *Troupe* builds on *EMC*.

TABLE 6.1: Comparison of Shapley value computation and approximation techniques in terms of having (✔) and missing (✘) desiderata; complexities with respect to the number of players m and permutations p.

Method	Voting	Bound	Non-Random	Space	Time
Explicit	✔	✔	✔	$\mathcal{O}(m)$	$\mathcal{O}(m!)$
MC [123]	✘	✔	✘	$\mathcal{O}(m)$	$\mathcal{O}(mp)$
TMC [55]	✘	✔	✘	$\mathcal{O}(m)$	$\mathcal{O}(mp)$
MLE [140]	✘	✘	✔	$\mathcal{O}(m)$	$\mathcal{O}(m)$
MMLE [103, 104]	✘	✘	✔	$\mathcal{O}(m)$	$\mathcal{O}(m!)$
EMC [46]	✔	✔	✔	$\mathcal{O}(m)$	$\mathcal{O}(m^2)$

Shapley value has previously been used in machine learning for measuring feature importance in linear models [117, 131, 151]. In the feature selection setting the features are seen as players that cooperate to achieve high goodness of fit. Various discussed approximation schemes [122, 188] have been exploited to make feature importance quantification in high dimensional spaces feasible [119, 195, 196] when explicit computation is not tractable. Another machine learning domain for applying the Shapley value was the pruning of neural networks [13, 56, 187]. In this context approximate Shapley values of hidden layer neurons are used to downsize overparametrized classifiers. It is argued in [187] that pruning neurons is analogous to feature selection on hidden layer features. Finally, there has been increasing interest in the equitable valuation of data points with game theoretic tools [84]. In such settings the estimated Shapley values are used to gauge the influence of individual points on a supervised model. These approximate scores are obtained with group testing of features [84] and permutation sampling [23, 55].

6.3 Ensemble games

We now introduce a novel class of co-operative games and examine axiomatic properties of solution concepts which can be applied to these games. We will discuss shapley value as an exact solution for these games, [180], and discuss approximations of the Shapley value based solution [46, 123, 140].

Ensemble game

We define an ensemble game to be one where binary classifier models (players) co-operate to label a single data point. The aggregated decision of the ensemble is assumed to be made by a vote of the players.

We assume that a set of labelled data points are known:

Definition 6.1. ***Labeled data point.*** *Let $(\boldsymbol{x}, y)$ be a labeled data point where $\boldsymbol{x} \in \mathbb{R}^d$ is the feature vector and $y \in \{0, 1\}$ is the corresponding binary label.*

Our work considers arbitrary binary classifier models (e.g. classification trees, support vector machines, neural networks) that operate on the same input feature vector $\mathbf{x}$. This approach is agnostic of the exact type of the model, we only assume that M can output a probability of

the data point having a positive label. The model owner does does not access the label, just a probability of $y = 1$ is output by the model.

Definition 6.2. ***Positive classification probability for model*** M**.** *Let $(\boldsymbol{x}, y)$ be a labeled data point and M be a binary classifier, $P(y = 1 \mid M, \boldsymbol{x})$ is the probability of the data point having a positive label output by classifier M.*

We are now ready to define the operation of an ensemble classifier consisting of multiple models.

Definition 6.3. ***Ensemble.*** *An ensemble is a set $\mathcal{M}$ of size m that consists of binary classifier models $M \in \mathcal{M}$ which can each output a probability for a data point $\boldsymbol{x} \in \mathbb{R}^d$ having a positive label. The ensemble sets the probability of a label as:*

$$P(y = 1 \mid \mathcal{M}, \boldsymbol{x}) = \sum_{M \in \mathcal{M}} P(y = 1 \mid M, \boldsymbol{x})/m.$$

And it makes decision about the label as:

$$\widehat{y} = \begin{cases} 1 & \text{if } P(y = 1 \mid \mathcal{M}, \boldsymbol{x}) \geq \gamma, \\ 0 & \text{otherwise.} \end{cases}$$

where $0 \leq \gamma \leq 1$ is the decision threshold and $\widehat{y}$ is the predicted label of the data point.

Definition 6.4. ***Sub-ensemble.*** *A sub-ensemble $\mathcal{S}$ is a subset $\mathcal{S} \subseteq \mathcal{M}$ of binary classifier models.*

Definition 6.5. ***Individual model weight.*** *The individual weight of the vote for M, in sub-ensemble $\mathcal{S} \subseteq \mathcal{M}$ for data point $(y, \boldsymbol{x})$ is defined as:*

$$w_M = \begin{cases} P(y = 1 \mid M, \boldsymbol{x})/m & \text{if } y = 1, \\ P(y = 0 \mid M, \boldsymbol{x})/m & \text{otherwise.} \end{cases}$$

Under this definition. the individual model weight of any binary classifier $M \in \mathcal{M}$ is bounded: $0 \leq w_M \leq 1/m$. Note that the weight of $M \in \mathcal{S}$ depends on m – the size of the larger ensemble and not on the size $|\mathcal{S}|$ of the sub-ensemble.

Definition 6.6. ***Ensemble game.*** *Let $\mathcal{M}$ be a set of binary classifiers. An ensemble game for a labeled data point $(y, \boldsymbol{x})$ is then a co-operative game $G = (\mathcal{M}, v)$ in which:*

$$v(\mathcal{S}) = \begin{cases} 1 & \text{if } w(\mathcal{S}) \geq \gamma, \\ 0 & \text{otherwise.} \end{cases}$$

where $w(\mathcal{S}) = \sum_{M \in \mathcal{S}} w_M$ for any sub-ensemble $\mathcal{S} \subseteq \mathcal{M}$ and threshold $0 \leq \gamma \leq 1$.

This definition is the central idea in our work. The models in the ensemble play a cooperative voting game to classify the data point correctly. When the data point is classified correctly the payoff is 1, an incorrect classification results in a payoff of 0. Each model casts a weighted vote about the data point and our goal is going to be to quantify the value of individual models in

the final decision. In other words, we would like to measure how individual binary classifiers *contribute on average* to the correct classification of a specific data point. This *solution concept is described in the next section(6.3)*.

We can consider a misclassification as a dual ensemble game:

Definition 6.7. ***Dual ensemble game.*** *Let $\mathcal{M}$ be a set of binary classifiers. A dual ensemble game for a labeled data point $(y, \boldsymbol{x})$ is then a co-operative game $G = (\mathcal{M}, \widetilde{v})$ in which:*

$$\widetilde{v}(\mathcal{S}) = \begin{cases} 1 & \text{if } \widetilde{w}(\mathcal{S}) \geq \widetilde{\gamma}, \\ 0 & \text{otherwise.} \end{cases}$$

for a binary classifier ensemble vote score $0 \leq \widetilde{w}(\mathcal{S}) \leq 1$ where $\widetilde{w}(\mathcal{S}) = \sum_{M \in \mathcal{S}}(1/m - w_M)$ for any sub-ensemble $\mathcal{S} \subseteq \mathcal{M}$ and inverse cutoff value $0 \leq \widetilde{\gamma} \leq 1$ defined by $\widetilde{\gamma} = 1 - \gamma$.

If the sum of classification weights for the binary classifiers is below the cutoff value the models in the ensemble misclassify the point, lose the ensemble game and as a consequence receive a payoff that is zero. In such scenarios it is interesting to ask: how can we describe the role of models in the misclassification? The dual ensemble game is derived from the original ensemble game in order to characterize this situation.

The classification game and its dual can be reframed simply as:

Definition 6.8. ***Simplified ensemble game.*** *An ensemble game in simplified form is described by the cutoff value – weight-vector tuple $(\gamma, [w_1, \ldots, w_m])$.*

Definition 6.9. ***Simplified dual ensemble game.*** *Given a simplified form ensemble game $(\gamma, [w_1, \ldots, w_m])$, the corresponding simplified dual ensemble game is defined by the cutoff value – weight vector tuple:*

$$(\widetilde{\gamma}, [\widetilde{w}_1, \ldots, \widetilde{w}_m]) = (1 - \gamma, [1/m - w_1, \ldots, 1/m - w_m])$$

The simplified forms of ensemble and dual ensemble games are compact data structures which can describe the game without the models themselves and the enumeration of every sub-ensemble.

Solution concepts for model valuation

We have defined the binary classification problem with an ensemble as a weighted voting game, which is a type of co-operative game. Now we will argue that *solution concepts* of co-operative games are suitable for the valuation of individual models which form the binary classifier ensemble.

Definition 6.10. ***Solution concept.*** *A solution concept defined for the ensemble game $G = (\mathcal{M}, v)$ is a function which assigns the real value $\Phi_M(\mathcal{M}, v) \in \mathbb{R}$ to each binary classifier $M \in \mathcal{M}$.*

The scalar Φ_M can be interpreted as the value of the individual binary classifier M in the ensemble $\mathcal{M}$. In the following we discuss axiomatic properties of solution concepts which are the desiderata for model valuation functions, and the implications of the axioms in the context of model valuation in binary ensemble games.

Axiom 6.11. ***Null classifier.*** *A solution concept has the null classifier property if* $\forall\ \mathcal{S} \subseteq \mathcal{M} : v(\mathcal{S} \cup \{M\}) = v(\mathcal{S})) \rightarrow \Phi_M(\mathcal{M}, v) = 0$.

Having the null classifier property means that a binary classifier which always has a zero marginal contribution in any sub-ensemble will have a zero payoff on its own. This also implies that the classifier never casts the deciding vote to correctly classify the data point when it is added to a sub-ensemble. Conversely, in the dual ensemble game the model never contributes to the misclassification of the data point.

Axiom 6.12. ***Efficiency.*** *A solution concept satisfies the efficiency property if* $v(\mathcal{M}) = \sum_{M \in \mathcal{M}} \Phi_M(\mathcal{M}, v)$.

That is, the value (loss or gain) of an ensemble can be split precisely into the contributed value of the constituent models.

Axiom 6.13. ***Symmetry.*** *A solution concept has the symmetry property if* $\forall \mathcal{S} \subseteq \mathcal{M} \setminus \{M', M''\} : v(\mathcal{S} \cup \{M'\}) = v(\mathcal{S} \cup \{M''\})$ *implies that* $\Phi_{M'}(\mathcal{M}, v) = \Phi_{M''}(\mathcal{M}, v)$.

Two binary classifiers which make equal marginal contribution to all sub-ensembles have the same value in the full ensemble.

Axiom 6.14. ***Linearity.*** *A solution concept has the linearity property if given any two ensemble games* $G = (\mathcal{M}, v)$ *and* $G' = (\mathcal{M}, v')$ *on the same set* $\mathcal{M}$*, the binary classifier* M *satisfies* $\Phi_M(\mathcal{M}, v + v') = \Phi_M(\mathcal{M}, v) + \Phi_M(\mathcal{M}, v')$.

That is, the value in the combined game is the sum of the values in individual games. This property will imply that valuations of a model for different datapoints, when added, leads to its valuation on the dataset.

The Shapley value

The Shapley value [180] of a classifier is the average marginal contribution of the model over the possible different permutations in which the ensemble can be formed [25]. It is a solution concept which satisfies Axioms 6.11-6.14 and the only solution concept which is uniquely characterized by Axioms 6.13 and 6.14.

Definition 6.15. ***Shapley value.*** *The Shapley value of binary classifier* M *in the ensemble* $\mathcal{M}$*, for the data point level ensemble game* $G = (\mathcal{M}, v)$ *is defined as*

$$\Phi_M(v) = \sum_{\mathcal{S} \subseteq \mathcal{M} \setminus \{M\}} \frac{|\mathcal{S}|!\ (|\mathcal{M}| - |\mathcal{S}| - 1)!}{|\mathcal{M}|!} (v(\mathcal{S} \cup \{M\}) - v(\mathcal{S})).$$

Calculating the exact Shapley value for every model in an ensemble game would take $\mathcal{O}(m!)$ time, or more, which is computationally unfeasible in large ensembles. We discuss a range of approximation approaches in detail which can give Shapley value estimates in $\mathcal{O}(m)$ and $O(m^2)$ time.

Multilinear extension (MLE) approximation of the Shapley value

The *MLE* approximation of the Shapley value in a voting game [46, 140] can be used to estimate the Shapley value in the ensemble game and its dual game. Let us define the expectation and the variation of the aggregated ensemble contributions for the remaining models as: $\mu_M = \sum_{i=1}^{m} w_j - w_M$ and $\nu_M = \sum_{i=1}^{m} w_j^2 - w_M^2$. For a classifier $M \in \mathcal{M}$ the multi-linear approximation of the unnormalized Shapley value is computed by:

$$\widehat{\Phi}_M \propto \int_{-\infty}^{\gamma} \frac{1}{\sqrt{2\pi\nu_M}} \exp\left(-\frac{(x-\mu_M)^2}{2\nu_M}\right) \mathrm{d}x - \int_{-\infty}^{\gamma-w_M} \frac{1}{\sqrt{2\pi\nu_M}} \exp\left(-\frac{(x-\mu_M)^2}{2\nu_M}\right) \mathrm{d}x.$$

This approximation assumes that the size of the game is large (many classifiers in the ensemble in our case) and also that μ has an *approximate normal distribution*. Calculating all of the approximate Shapley values by *MLE* takes $\mathcal{O}(m)$ time.

Monte Carlo (MC) approximation of the Shapley value

The MC approximation [122, 123] given the ensemble $\mathcal{M}$ estimates the Shapley value of the model $M \in \mathcal{M}$ by the average marginal contribution over uniformly sampled permutations.

$$\hat{\Phi}_M = \mathbb{E}_{\theta\sim\Theta}[v(\mathcal{S}_\theta^M \cup \{M\}) - v(\mathcal{S}_\theta^M)] \tag{6.1}$$

In Equation equation 6.1, Θ is a uniform distribution over the $m!$ permutations of the binary classifiers and $\mathcal{S}_\theta^M$ is the subset of models that appear before the classifier M in permutation θ. Approximating the Shapley value requires the generation of p classifier permutations (for a suitable p), and marginal contribution calculations with respect to those contributions – this takes $\mathcal{O}(mp)$ time.

Data: $[w_1, \ldots, w_m]$ – Weights of binary classifiers.
γ – Cutoff value.
δ – Numerical stability parameter.
μ – Expected value of weights.
ν – Variance of weights.
Result: $(\widehat{\Phi}_1, \ldots, \widehat{\Phi}_m)$ – Approximate Shapley values.

for $j \in \{1, values \ldots, m\}$ **do**
  $\widehat{\Phi}_j \leftarrow 0$
  for $k \in \{1, \ldots, m-1\}$ **do**
    $a \leftarrow (\gamma - w_j/k$
    $b \leftarrow (\gamma - \delta)/k$
    $\widehat{\Phi}_j \leftarrow \widehat{\Phi}_j + \frac{1}{\sqrt{2\pi\nu/k}} \int_a^b \exp(-k\frac{(x-\mu)^2}{2\nu})\mathrm{d}x$
  end
end

Algorithm 6: Expected marginal contribution approximation of Shapley values.

Voting game approximation of the Shapley value

The ensemble games introduced above are a variant of voting games [138], hence we can use the *Expected Marginal Contributions (EMC)* approximation [46]. This procedure sums the expected marginal contributions of a model to fixed size ensembles – it is described in the pseudo-code Algorithm 6. The algorithm iterates over the individual model weights and initializes Shapley values as zeros (lines 1-2). For each ensemble size it calculates the expected contribution of the model to the ensemble. This expected contribution is the probability that a classifier becomes the marginal voter. These marginal contributions are added to the approximate Shapley value (lines 3-8). Using this Shapley value approximation technique takes $\mathcal{O}(m^2)$ time. However, it is the most accurate Shapley value approximation technique in terms of absolute error of the Shapley value estimates [46].

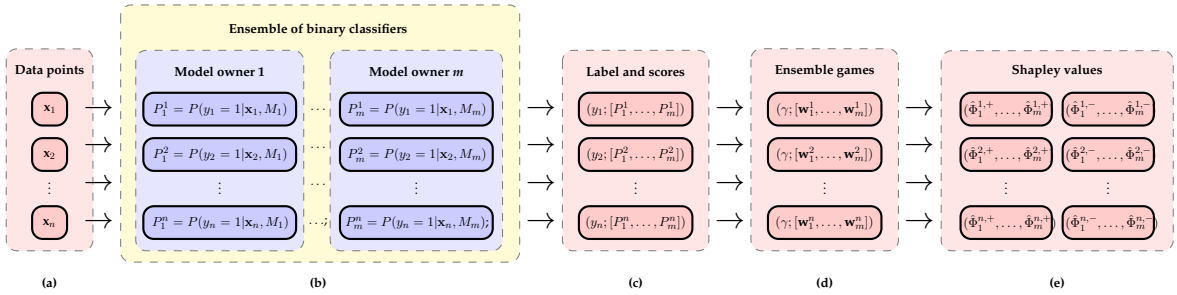


FIGURE 6.2: The approximate average Shapley value calculation pipeline. (a) Given the set of labeled datapoints the data owner sends the features of data points to the binary classifiers in the ensemble. (b) The models in the the ensemble score each of the data points independently and send the probability scores back to the oracle. (c) Given the label and the scores the oracle defines the ensemble game on each of the data points. (d-e) The ensemble games and the dual ensemble games are solved by the oracle and approximate Shapley values of models are calculated in each of the data point level ensemble games.

6.4 Model valuation with the Shapley value

In this section we present *Troupe*, the mechanism used for model valuation in ensembles. The chart in Figure 6.2 gives a high-level summary of our model valuation approach. We also point out theoretical properties of the proposed model valuation framework.

Data: $(\mathbf{x}, y)$ – Labeled data point.
$\{M_1, \ldots, M_m\}$ – Set of binary classifiers.
Result: $[w_1, \ldots, w_m]$ – Weights of individual models.
1 **for** $j \in \{1, \ldots, m\}$ **do**
2 **if** $y = 1$ **then**
3 $w_j \leftarrow P(y = 1 \mid M_j, \mathbf{x})/m$
4 **else**
5 $w_j \leftarrow P(y = 0 \mid M_j, \mathbf{x})/m$
6 **end**
7 **end**

Algorithm 7: Calculating the individual model weights in an ensemble game given a data point and a classifier ensemble.

The model valuation algorithm

The Shapley value based model evaluation procedure described by Algorithm 8 uses $\mathcal{D} = \{(\mathbf{x}_1, y_1); \ldots; (\mathbf{x}_n, y_n)\}$, a set of n datapoints with binary labels, for evaluating the ensemble $\mathcal{M}$ given a cutoff value $0 \leq \gamma \leq 1$ and a numerical stability parameter $\delta \in \mathbb{R}^{0+}$.

We set m the number of models and n the number of labeled data points used in the model valuation (lines 1-2). We iterate over all of the data points (line 3) and given a data point for all of the models we compute the individual model weights in an ensemble game using the label, features and the model itself (line 4) – see Algorithm 7. Using the individual model weights in the ensemble game we calculate the expected value and variance of the data point (lines 5-6).

If the sum of individual model weights is higher than the cutoff value (line 7) the Shapley values in the ensemble game are approximated by Algorithm 6 for the data point (line 8). The Shapley values in the dual ensemble game are defined by a vector of zeros (line 9). If the point is misclassified (line 10) a *dual ensemble game* is defined for the data point. This requires the redefined cutoff value $\widetilde{\gamma}$, expected value $\widetilde{\mu}$ and variance $\widetilde{\nu}$ of the individual model weight vector (lines 11-13). We exploit the linearity of expectation and the location invariance of the variance. The individual model weight vector is redefined (line 14) in order to quantify the role of models in the erroneous decision. The original ensemble game Shapley values are initialized with zeros (line 15) and using the new parametrization of the game we approximate the dual ensemble game Shapley values (line 16) by Algorithm 6. The algorithm outputs data point level Shapley values for each model.

Measuring ensemble heterogeneity

The data point level Shapley values of ensemble games and their dual can be aggregated to measure the importance of models in ensemble level decisions. Using the information entropy of the Shapley values we define heterogeneity metrics of ensembles.

The average conditional Shapley value

In order to quantify the role of models in classification and misclassification we calculate the average Shapley value of models in the ensemble and dual ensemble games conditional on the success of classification. The sets $\mathcal{N}^+$ and $\mathcal{N}^-$ contain the indices of classified and misclassified data points. Using the cardinality of these sets $n^+ = |\mathcal{N}^+|$, $n^- = |\mathcal{N}^-|$ and the data point level Shapley values output by Algorithm 6 we can estimate the *conditional* role of models in classification and misclassification by averaging the approximate Shapley values using Equations 6.2 and 6.3.

$$(\overline{\Phi}_1^+, \ldots, \overline{\Phi}_m^+) = \sum_{i \in \mathcal{N}^+} (\widehat{\Phi}_0^{i,+}, \ldots, \widehat{\Phi}_m^{i,+})/n^+ \tag{6.2}$$

$$(\overline{\Phi}_1^-, \ldots, \overline{\Phi}_m^-) = \sum_{i \in \mathcal{N}^-} (\widehat{\Phi}_0^{i,-}, \ldots, \widehat{\Phi}_m^{i,-})/n^- \tag{6.3}$$

Data: $\mathcal{D}$ – Labeled data points.
$\mathcal{M}$ – Set of binary classifiers.
γ – Cutoff value.
δ – Numerical stability parameter.
Result: $\underbrace{\left\{(\widehat{\Phi}_0^{1,+},\ldots,\widehat{\Phi}_m^{1,+}),\ldots,(\widehat{\Phi}_0^{n,+},\ldots,\widehat{\Phi}_m^{n,+})\right\}}_{\text{Shapley value vectors for ensemble games.}}$
$\underbrace{\left\{(\widehat{\Phi}_0^{1,-},\ldots,\widehat{\Phi}_m^{1,-}),\ldots,(\widehat{\Phi}_0^{n,+},\ldots,\widehat{\Phi}_m^{n,+})\right\}}_{\text{Shapley value vectors for dual ensemble games.}}$

$m \leftarrow |\mathcal{M}|$
$n \leftarrow |\mathcal{D}|$
for $i \in \{1,\ldots,n\}$ **do**
  $[w_1^i,\ldots,w_m^i] \leftarrow$ **Get Weights**$((\mathbf{x}_i, y_i); \{M_1,\ldots,M_m\})$
  $\mu \leftarrow \sum_{j=1}^{m} w_j^i/m$
  $\nu \leftarrow \sum_{j=1}^{m} (w_j^i - \mu)^2/m$
  if $\mu > \gamma/m$ **then**
    $(\widehat{\Phi}_0^{i,+},\ldots,\widehat{\Phi}_m^{i,+}) \leftarrow$ **Shapley** $\left([w_1^i,\ldots,w_m^i]; \gamma; \delta; \mu; \nu\right)$
    $(\widehat{\Phi}_0^{i,-},\ldots,\widehat{\Phi}_m^{i,-}) \leftarrow \mathbf{0}$
  else
    $\widetilde{\gamma} \leftarrow 1 - \gamma$
    $\widetilde{\mu} \leftarrow 1/m - \mu$
    $\widetilde{\nu} \leftarrow \nu$
    $[\widetilde{w}_1^i,\ldots,\widetilde{w}_m^i] \leftarrow [1/m - w_1^i,\ldots,1/m - w_m^i]$
    $(\widehat{\Phi}_0^{i,+},\ldots,\widehat{\Phi}_m^{i,+}) \leftarrow \mathbf{0}$
    $(\widehat{\Phi}_0^{i,-},\ldots,\widehat{\Phi}_m^{i,-}) \leftarrow$ **Shapley** $([\widetilde{w}_1,\ldots,\widetilde{w}_m]; \widetilde{\gamma}; \delta; \widetilde{\mu}; \widetilde{\nu})$
  end
end

Algorithm 8: *Troupe:* Calculating the approximate Shapley value of the classifiers in ensemble and dual ensemble games.

If a component of the average Shapley value vector in Equation 6.2 is large compared to other components the corresponding model has an important role in the correct classification decisions of the ensemble. A large component in Equation 6.3 corresponds to a model which is responsible for a large number of misclassifications.

The Shapley entropy

The Shapley values of the ensemble games can be interpreted as a probability distribution over the classifiers in the ensemble. Using the information entropy of average Shapley values of these games we define metrics that quantify the heterogeneity of ensembles with respect to classification and misclassification decisions – see Equations 6.4 and 6.5.

$$H^+ = -\sum_{j=1}^{m} \overline{\Phi}_j^+ \log(\overline{\Phi}_j^+) \tag{6.4}$$

$$H^- = -\sum_{j=1}^{m} \overline{\Phi}_j^- \log(\overline{\Phi}_j^-) \tag{6.5}$$

If H^+ is high it means that the all of the models are responsible for classification decisions to the same extent. A low H^+ value implies that a subset of models in the ensemble have a dominant role in correct classification decisions. The H^- score can be interpreted analogously with respect to misclassifications decisions.

Theoretical properties

Our framework utilizes labeled data instances to approximate the importance of classifiers in the ensemble and this has important implications. In the following we discuss how the size of the dataset and the number classifiers affects the Shapley value approximation error and the runtime of *Troupe*.

Bounding the average approximation error

Our discussion focuses on the average conditional Shapley value in ensemble games. However, analogous results can be obtained for the Shapley values computed from dual ensemble games.

Definition 6.16. ***Approximation error.*** *The Shapley value approximation error of model $M \in \mathcal{M}$ in an ensemble game is defined as $\Delta\Phi_M^+ = \widehat{\Phi}_M^+ - \Phi_M^+$.*

Definition 6.17. ***Average approximation error.*** *Let use denote the Shapley value approximation errors of model $M \in \mathcal{M}$ calculated from the dataset $\mathcal{D}$ of correctly classified points as $\Delta\Phi_M^{1,+}, \ldots, \Delta\Phi_M^{n,+}$. The average approximation error is defined by $\overline{\Delta\Phi}_M^+ = \sum_{i=1}^{n} \Delta\Phi_M^{i,+}/n$.*

Theorem 6.18. ***Average conditional Shapley value error bound.*** *If the Shapley value approximation errors $\Delta\Phi_M^{1,+}, \ldots, \Delta\Phi_M^{n,+}$ of model $M \in \mathcal{M}$ calculated by Algorithm 8 from the dataset $\mathcal{D}$ are independent random variables than for any $\varepsilon \in \mathbb{R}^+$ Inequality 6.6 holds.*

$$P(|\overline{\Delta\Phi}_M^+ - \mathbb{E}[\overline{\Delta\Phi}_M^+]| \geq \varepsilon) \leq 2\exp\left(-\sqrt{\frac{n^2 \cdot m \cdot \varepsilon^4 \cdot \pi}{8}}\right) \tag{6.6}$$

Proof. Let us first note the fact that every absolute Shapley approximation value is bounded by the inequality described in Lemma 6.19.

Lemma 6.19. ***Approximate Shapley value bound.*** *As [46] states the approximation error of the Shapley value in a single voting game is bounded by Inequality 6.7 when the expected marginal contributions approximation is used.*

$$-\sqrt{\frac{8}{m\pi}} \leq \Delta\Phi_M^+ \leq \sqrt{\frac{8}{m\pi}} \tag{6.7}$$

Using Lemma 6.19, the fact that *Troupe* is based on the expected marginal contributions approximation and that the Shapley values of a model in different ensemble games are independent random variables we can use Hoeffding's second inequality [77] for bounded non zero-mean random variables:

$$P(|\overline{\Delta\Phi}_M^+ - \mathbb{E}[\overline{\Delta\Phi}_M^+]| \geq \varepsilon) \leq 2\exp\left(-\frac{2\varepsilon^2 n^2}{\sum_{i=1}^{n}\left[\left(\sqrt{\frac{8}{m\pi}}\right) - \left(-\sqrt{\frac{8}{m\pi}}\right)\right]}\right)$$

$$P(|\overline{\Delta\Phi}_M^+ - \mathbb{E}[\overline{\Delta\Phi}_M^+]| \geq \varepsilon) \leq 2\exp\left(-\frac{2\varepsilon^2 n^2}{2n\sqrt{\frac{8}{m\pi}}}\right)$$

□

Theorem 6.20. ***Confidence interval of the expected average approximation error.*** *In order to acquire an $(1-\alpha)$-confidence interval of $\mathbb{E}[\overline{\Delta\Phi}_M^+] \pm \varepsilon$ one needs a labeled dataset $\mathcal{D}$ of correctly classified data points for which n the cardinality of $\mathcal{D}$, satisfies Inequality 6.8.*

$$n \geq \sqrt{\frac{8\ln^2\left(\frac{\alpha}{2}\right)}{\varepsilon^4 m\pi}} \tag{6.8}$$

Proof. The probability $P(|\overline{\Delta\Phi}_M^+ - \mathbb{E}[\overline{\Delta\Phi}_M^+]| \geq \varepsilon)$ in Theorem 6.18 equals to the level of significance for the confidence interval $\mathbb{E}[\overline{\Delta\Phi_M}^+] \pm \varepsilon$. Which means that Inequality 6.9 holds for the significance level α.

$$\alpha \leq 2\exp\left(-\sqrt{\frac{n^2 \cdot m \cdot \varepsilon^4 \cdot \pi}{8}}\right). \tag{6.9}$$

Solving inequality 6.9 for n yields the cardinality of the dataset (number of correctly classified data points) required for obtaining the confidence interval described in Theorem 6.20. □

The inequality presented in Theorem 6.20 has two important consequences regarding the bound:

1. Larger ensembles require less data in order to give confident estimates of the Shapley value for individual models.

2. The dataset size requirement is sublinear in terms of confidence level and quadratic in the precision of the Shapley value approximation.

Runtime and memory complexity.

The runtime and memory complexity of the proposed model valuation framework depends on the complexity of the main evaluation phases. We assume that our framework operates in a single-core non distributed setting.

Scoring and game definition Assuming that the scoring of a data point takes $\mathcal{O}(1)$ time, scoring the data point with all models takes $\mathcal{O}(m)$. Scoring the whole dataset and defining games both takes $\mathcal{O}(nm)$ time and $\mathcal{O}(nm)$ space respectively.

Approximation and overall complexity. Calculating the expected marginal contribution of a model to a fixed size ensemble takes $\mathcal{O}(1)$ time. Doing this for all of the ensemble sizes takes $O(m)$ time. Approximating the Shapley value for all models requires $O(m^2)$ time and $O(m)$ space. Given a dataset of n points this implies a need for $O(nm^2)$ time and $O(nm)$ space. This is also the overall time and space complexity of the proposed framework.

6.5 Experimental evaluation

In this section, we show that *Troupe* approximates the average of Shapley values precisely. We provide evidence that Shapley values are a useful decision metric for ensemble creation. Our results illustrate that model importance and complexity are correlated, and that Shapley values are able to identify adversarial models in the ensemble.

Our evaluation is based on various real world binary graph classification tasks [168]. Specifically, we use datasets collected from *Reddit*, *Twitch* and *GitHub* – the descriptive statistics of these datasets are enclosed as Table 6.2.

TABLE 6.2: Descriptive statistics of the binary graph classification datasets taken from [168] used for the evaluation of our proposed framework. These datasets are fairly balanced, while the graphs are hetereogeneous with respect to size, density and diameter.

	Classes		Nodes		Density		Diameter	
Dataset	**Positive**	**Negative**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Reddit	521	479	11	93	0.023	0.027	2	18
Twitch	520	480	14	52	0.039	0.714	2	2
GitHub	552	448	10	942	0.004	0.509	2	15

The precision of approximation

The Shapley value approximation performance of *Troupe* was compared to that of various other estimation schemes [122, 140] discussed earlier. We used an ensemble of logistic regressions where each classifier was trained on features extracted with a whole graph embedding technique [171, 202, 213]. We utilized 50% of the graphs for training and calculated the average conditional Shapley values of ensemble games and dual ensemble games using the remaining 50% of data.

Experimental details

The features of the graphs were extracted with whole graph embedding techniques implemented in the open source *Karate Club* framework [168]. Given a set of graphs $\mathcal{G} = (G_1, \ldots, G_n)$ whole graph embedding algorithms [171, 202] learn a mapping $g : \mathcal{G} \to \mathbb{R}^d$ which delineate the graphs $G \in \mathcal{G}$ to a d dimensional metric space. We utilized the following whole graph embedding and statistical finger printing techniques:

1. **FEATHER** [171] uses the characteristic function of topological features as a graph level statistical descriptor.
2. **Graph2Vec** [134] extracts tree features from the graph.
3. **GL2Vec** [27] distills tree features from the dual graph.
4. **NetLSD** [202] derives characteristic of graphs using the heat trace of the graph spectra.
5. **SF** [34] utilizes the largest eigenvalues of the graph Laplacian matrix as an embedding.
6. **LDP** [21] sketches the histogram of local degree distributions.
7. **GeoScattering** [51] applies the scattering transform to various structural features (e.g. degree centrality).
8. **IGE** [50] combines graph features from local degree distributions and scattering transforms.
9. **FGSD** [213] sketches the Moore-Penrose spectrum of the normalized graph Laplacian with a histogram.

The embedding techniques used the *default settings* of the *Karate Club* library, each embedding dimension was column normalized and the graph features were fed to the *scikit-learn* implementation of logistic regression. This classifier was trained with ℓ_2 penalty cost, we chose an SGD optimizer and the regularization coefficient λ was set to be 10^{-2}.

TABLE 6.3: Absolute percentage error of average conditional Shapley values obtained by approximation techniques (rows) for the graph classifiers (columns) in the ensemble game. Bold numbers note the lowest error on each dataset – classifier pair.

	Approximation	FEATHER	Graph2Vec	GL2Vec	NetLSD	SF	LDP	GeoScatter	IGE	FGSD
Reddit	**Troupe**	**1.23**	**2.35**	8.18	**0.99**	**2.64**	**2.31**	**1.64**	**4.85**	**1.49**
	MLE	3.20	23.61	32.62	4.19	5.34	7.97	7.12	5.42	7.61
	MC $p = 10^3$	12.57	30.94	13.26	8.67	32.76	12.32	11.62	16.36	12.78
	MC $p = 10^3$	4.71	5.38	**3.41**	1.67	3.03	5.51	4.34	4.97	3.82
Twitch	**Troupe**	**0.28**	**3.33**	**1.18**	**2.53**	**1.62**	**0.59**	**1.48**	**0.25**	1.19
	MLE	5.22	5.44	3.05	8.32	2.38	1.92	3.14	4.85	5.77
	MC $p = 10^2$	2.37	10.40	6.76	7.07	15.79	6.36	13.99	23.96	**0.39**
	MC $p = 10^3$	2.32	4.60	2.96	2.67	2.53	2.73	6.31	0.27	3.89
GitHub	**Troupe**	**2.68**	**0.18**	**2.61**	**1.41**	**1.49**	1.23	**1.88**	**2.36**	1.04
	MLE	9.22	5.12	3.76	3.27	9.71	7.46	5.08	4.04	0.73
	MC $p = 10^2$	5.91	9.37	4.67	9.82	8.78	8.34	13.66	28.95	**0.76**
	MC $p = 10^3$	3.35	6.09	7.70	3.26	2.84	**0.79**	6.67	2.51	1.12

Approximate model importance measurement

In Table 6.3 we summarized the absolute percentage error values (compared to exact Shapley values in ensemble games) obtained with the various approximations schemes. (i) Our empirical results support that *Troupe* consistently computes accurate estimates of the ground truth model evaluations across datasets and classifiers in the ensemble. (ii) The high quality of estimates suggests that the approximate Shapley values of models extracted by *Troupe* can serve as a proxy decision metric for ensemble building and model selection.

TABLE 6.4: Exact and approximate Shapley entropy values of ensemble and dual ensemble games on the whole graph embedding expert system based classification tasks. Bold numbers denote the best approximation.

		Exact	**Troupe**	**MLE**	**MC**$_{p=10^2}$	**MC**$_{p=10^3}$
Reddit	H^+	2.1860	**2.1864**	2.1971	2.1818	2.1929
	H^-	2.1907	**2.1911**	2.1972	2.1765	2.1860
Twitch	H^+	2.1963	**2.1962**	2.1972	2.1873	2.1949
	H^-	2.1953	**2.1956**	2.1972	2.1890	2.1947
GitHub	H^+	2.1953	**2.1954**	2.1972	2.1907	2.1944
	H^-	2.1961	**2.1961**	2.1971	2.1869	2.1966

Approximate ensemble heterogeneity measurement

Using the previous conditional average Shapley values of ensemble and dual ensemble games we calculated the Shapley entropy scores – see Table 6.4. Our empirical findings suggest that ensemble heterogeneity varies when it comes to correct and incorrect decisions. We also see evidence that: (i) *Troupe* gives the best estimates of the exact ensemble heterogeneity scores; (ii) *MLE* consistently underestimates the ensemble heterogeneity while *MC* overestimates the ensemble heterogeneity (which is a result of the non-exact computation).

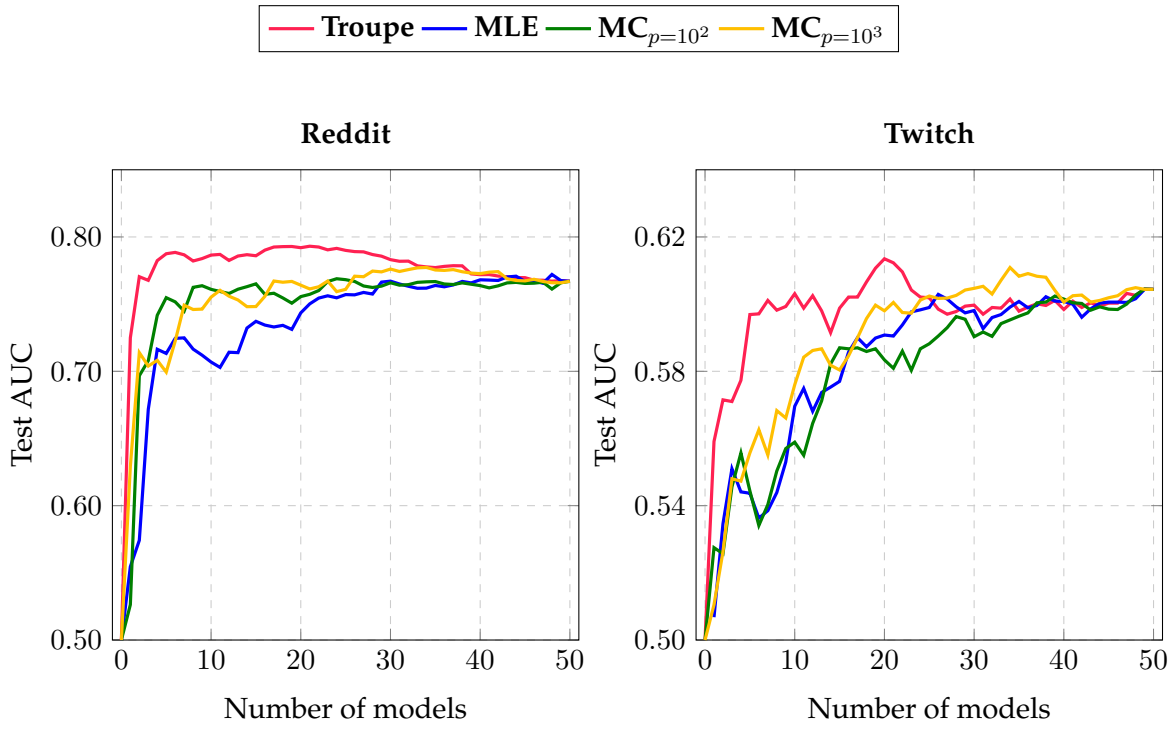


FIGURE 6.3: The test graph classification performance of binary classifier ensembles as a function of ensembles selected by our forward model building procedure. Models are added iteratively to the ensemble based on the average approximate Shapley values extracted from the ensemble games.

Ensemble building

Our earlier results suggested that the approximate Shapley values output by *Troupe* (and other estimation methods) can be used as decision metrics for ensemble building. We demonstrate this by selecting a high performance subset of a random forest in a forward fashion using the

estimated model valuation scores. The test performance (measured by the AUC scores) of these classifiers as a function of subensemble size is plotted on Figure 6.3. From each graph we extracted Weisfeiler-Lehman tree features [134] and kept those topological patterns which appeared in at least 5 graphs. Using the counts of these features in graphs we define statistical descriptors. Using 40% of the graphs we trained a random forest with 50 classification trees, each tree was trained on 20 randomly sampled Weisfeiler-Lehman features – we used the default settings of *scikit-learn*. We calculated the average conditional Shapley value of classifiers in the ensemble games defined on using 30% of the data. We ordered the classifiers by the approximate Shapley values in decreasing order, created subensembles in a forward fashion and calculated the predictive performance of the resulting subensembles on the remaining 30% of the graphs. Our results suggest that Troupe has a material advantage over competing approximation schemes on the Twitch and Reddit datasets.

Model complexity and influence

We fitted a voting expert which consisted of binary classifiers with heterogeneous model complexity. We extracted the Weisfeiler-Lehman features which appeared in at least 5 graphs in the datasets. The models were trained with 50% of the dataset and the average Shapley values were calculated from the remaining 50% of graphs.

Neural network ensembles

We created an ensemble of $m = 10^3$ neural networks using *scikit-learn* – each of these had a single hidden layer. Each model received 20 randomly selected frequency features as input and had a randomly chosen number of hidden layer neurons – we uniformly sampled this hyperparameter from $\{2^3, 2^4, 2^5, 2^6, 2^7\}$. Individual neural networks were trained by minimizing the binary cross-entropy with SGD for 200 epochs with a learning rate of 10^{-2}.

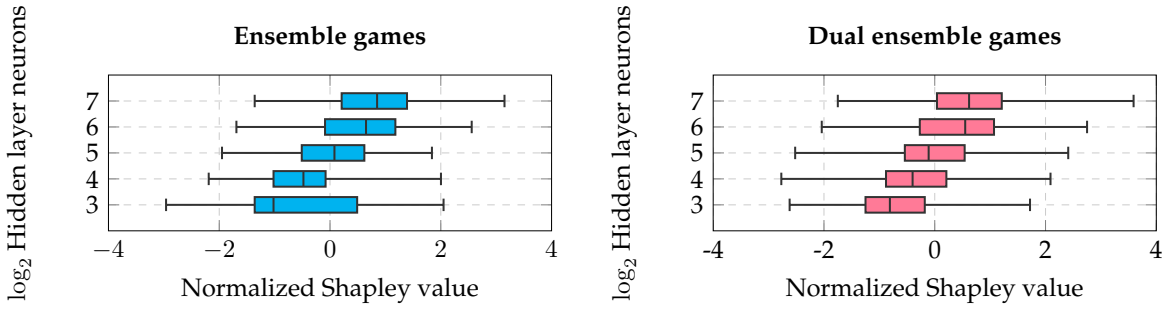


FIGURE 6.4: The distribution of normalized Shapley values for neural networks in ensemble and dual ensemble games conditional on the number of neurons (Reddit dataset).

The distribution of normalized average Shapley values are plotted on Figures 6.4 and 6.5 for the ensemble and dual ensemble games conditioned on the number of hidden layer neurons. These results imply that more complex models with a larger number of free parameters receive higher Shapley values in both classes of games. In simple terms complex models contribute to correct and incorrect classification decisions at a disproportionate rate.

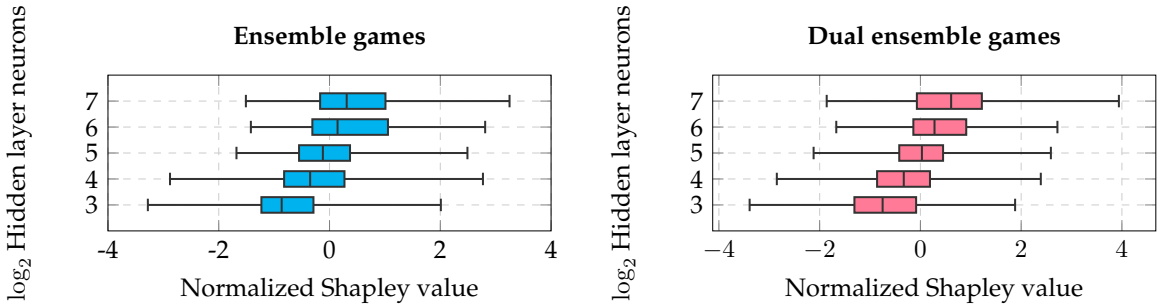


FIGURE 6.5: The distribution of normalized Shapley values for neural networks in ensemble and dual ensemble games conditional on the number of neurons (Twitch dataset).

Random forest ensembles

We created a random forest ensemble of $m = 10^3$ classification trees using *scikit-learn*. Each tree in the ensemble received 20 randomly selected Wesifeiler-Lehman count features as input. We used the default settings of *scikit-learn* except for the maximal depth which we fixed to be 4. Using the Reddit and Twitch datasets we plotted on Figures 6.6 and 6.7 the mean normalized Shapley value of classification trees obtained by *Troupe* in the ensemble games and dual ensemble games conditioned on the number of leaves that the trees have.

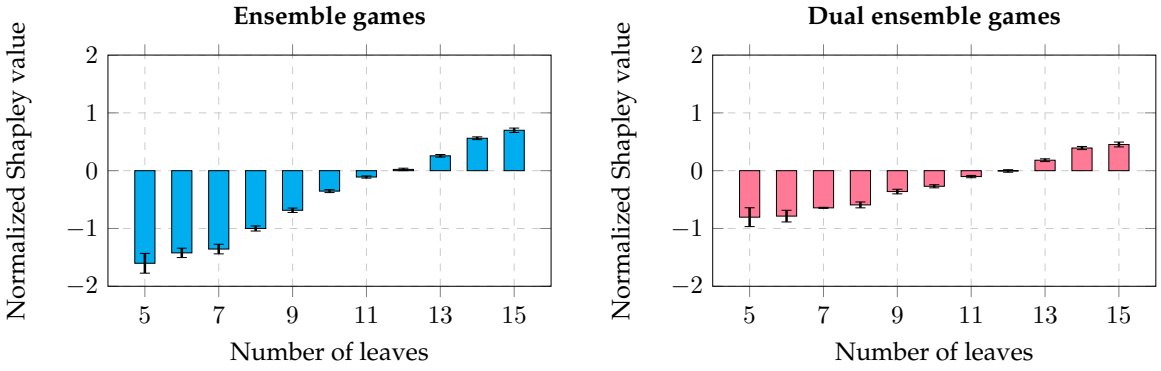


FIGURE 6.6: The mean and standard error of normalized Shapley values for classification trees in ensemble and dual ensemble games conditional on the number of leaves in the tree (Reddit dataset).

Our results support the claim made earlier that more complex models (higher number of tree leaves) contribute to correct and incorrect classification decisions with a higher probability. However, in this case the higher number of leaves might be a random artifact of sampling better quality features which are more discriminative.

Identifying adversarial models

Ensembles can be formed by the model owners submitting their classifiers voluntarily to a machine learning market. We investigated how adversarial behaviour of model owners affects the Shapley values of classifiers. We used Weisfeiler-Lehman tree features and trained a random forest ensemble of 20 classifiers using 50% of the data – each of these models utilized a random subset of 20 features and had a maximal tree depth of 4. The Shapley values were calculated from the remaining 50% of the data using *Troupe*. We artificially corrupted the predictions for 10

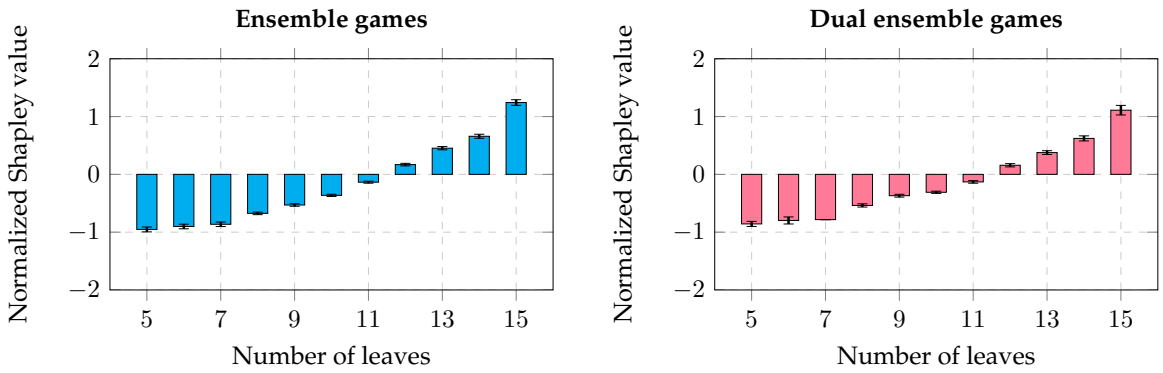


FIGURE 6.7: The mean and standard error of normalized Shapley values for classification trees in ensemble and dual ensemble games conditional on the number of leaves in the tree (Twitch dataset).

of the classification trees by mixing the outputted probability values with noise that had $\mathcal{U}(0,1)$ distribution. The corrupted predictions were a convex combination of the original prediction and the noise – we call the weight of the noise as the adversarial noise ratio. Given an adversarial noise ratio we calculated the mean Shapley value (with standard errors) for the adversarial and normally behaving classification trees in the ensemble.

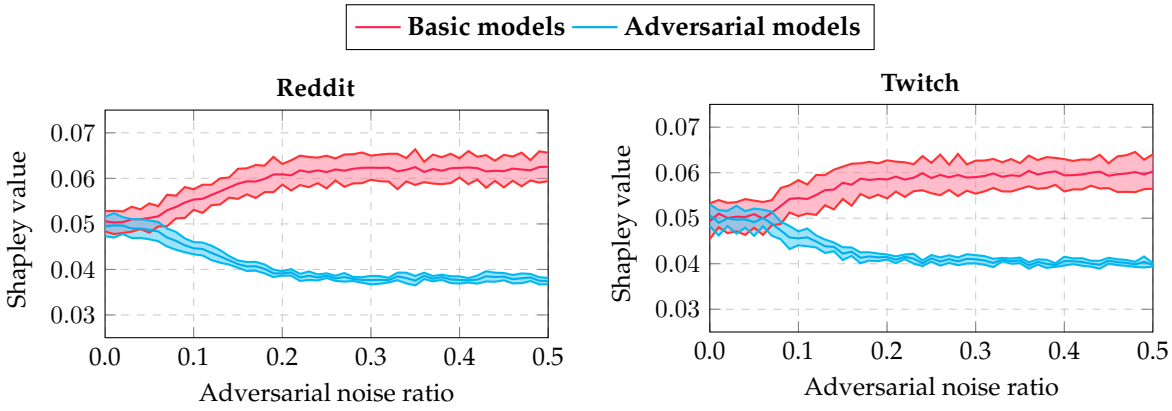


FIGURE 6.8: The mean Shapley value (standard errors added) of adversarial and base classifiers in ensemble games as a function of adversarial noise ratio mixed to predictions.

In Figure 6.8 we plotted the mean Shapley value of models which are adversarial and which are not in a scenario where a fraction of model owners mixed their predictions with noise. Even with a negligible amount of adversarial noise the Shapley values of adversarial models drop in the ensemble games considerably.

Scalability

We plotted the mean runtime of Shapley value approximations calculated from 10 experimental runs for an ensemble with $m = 2^5$ and dataset with $n = 2^8$ on Figure 6.9. All results were obtained with our open-source framework. These average runtimes of the approximation techniques are in line with the known and new theoretical results discussed in Sections 6.2 and 6.4. The precise estimates of Shapley values obtained with *Troupe* come at a time cost.

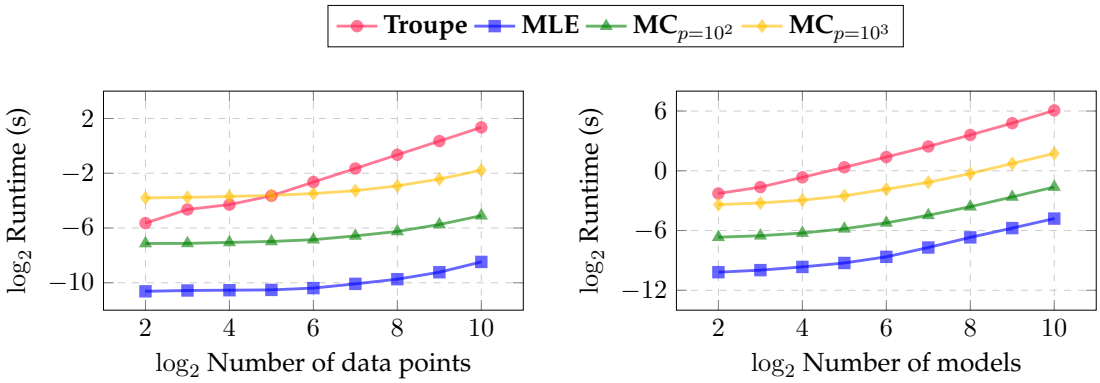


FIGURE 6.9: The runtime of Shapley value approximation in ensemble games as a function of dataset size and number of classifiers in the ensemble.

6.6 Conclusions and future directions

We proposed a new class of cooperative games called *ensemble games* in which binary classifiers cooperate in order to classify a data point correctly. We postulated that solving these games with the Shapley value results in a measure of individual classifier quality. We designed *Troupe* a voting game inspired approximation algorithm which computes the average of Shapley values for every classifier in the ensemble based on a dataset. Using these estimated model valuation scores we introduced metrics which describe the model heterogeneity of the ensemble with respect to predictive accuracy. We provided theoretical results about the sample size needed for precise estimates of the model quality.

We have demonstrated that our algorithm can provide accurate estimates of the Shapley value and the ensemble heterogeneity measures on real world social network data. We illustrated how the Shapley values of the models can be used to create small sized but highly accurate graph classification ensembles. We presented evidence that complex models have an important role in the classification decisions of ensembles. We showcased that our framework can identify adversarial models in the classification ensemble.

We think that our contribution opens up venues for novel theoretical and applied data mining research about ensembles. We theorize that our model valuation framework can be generalized to ensembles which consist of multi-class predictors using a one versus many approach. Our work provides an opportunity for the design and implementation of real world machine learning markets where the payoff of a model owner is a function of the individual model value in the ensemble.

Part II

Software for Graph Mining

Chapter 7

Karate Club: A Python Library for Unsupervised Learning on Graphs

Graphs encode important structural properties of complex systems. Machine learning on graphs has therefore emerged as an important technique in research and applications. We present *Karate Club* – a Python framework combining more than 30 state-of-the-art graph mining algorithms. These unsupervised techniques make it easy to identify and represent common graph features. The primary goal of the package is to make community detection, node and whole graph embedding available to a wide audience of machine learning researchers and practitioners. *Karate Club* is designed with an emphasis on a consistent application interface, scalability, ease of use, sensible out of the box model behaviour, standardized dataset ingestion, and output generation. This chapter discusses the design principles behind the framework with practical examples. We show *Karate Club*'s efficiency in learning performance on a wide range of real world clustering problems and classification tasks along with supporting evidence of its competitive speed.

7.1 Introduction

Techniques that extract features from graph data in an unsupervised manner [134, 147, 232] have seen an increasing success in the machine learning community. Features automatically extracted by these methods can serve as inputs for link prediction, node and graph classification, community detection and various other tasks tasks [134, 147, 148, 165, 226] in a wide range of real world research and application scenarios. Graph mining tools such as [68, 107, 146] have contributed to enhancement and development of machine learning pipelines. The need for complicated custom feature engineering is reduced by unsupervised mining techniques. This approach produces features that are naturally reusable on multiple types of tasks. Recent research [147, 148, 202] has made such feature extraction highly efficient and parallelizable.

The democratization of machine learning for tabular data was led by frameworks which made fast paced development possible. Tools such as [1, 20, 144, 145] are available in general purpose scripting languages with easy to use consistent interfaces. On the other hand, current graph based learning frameworks are less mature and of limited scope [68, 146]. For example, these packages host certain community detection algorithms, but none for whole graph or node embedding. In addition, some of these tools [107, 146] have significant barriers for the end users in terms of installing prerequisites and custom data structures used for representing graphs.

Present work. We propose *Karate Club*, an open source Python framework for unsupervised learning on graphs. We implemented *Karate Club* with consistent API oriented design principles in mind which makes the library end user friendly and modular. The name of the package is inspired by Zachary's Karate Club [242] – a network commonly used to demonstrate network algorithms. The design of this machine learning tool box was motivated by the principles used to create the widely used *scikit-learn* package [20].

The design entails a number of fundamental engineering patterns. Each algorithm has a sensible default hyperparameter setting which helps non expert practitioners. To further ease the use of our package, algorithms have a limited number of shared, publicly available methods (e.g. fit). Models ingest data structures from the scientific Python ecosystem [68, 214, 215] as input and the generated output also follows these formats. The inner model mechanics are always implemented as private methods using optimized back-end libraries [39, 156, 214, 215] for computing. These principles combined with the extensive documentation ensure that *Karate Club* is accessible to a wider audience than the currently available open-source graph mining frameworks.

Our empirical evaluation focuses on three common graph mining tasks: non-overlapping community detection, node and graph classification. We compare the learning performance of node and graph level algorithms implemented in our framework on various real world social, web and collaboration networks (collected from Deezer, Reddit, Facebook, Twitch, Wikipedia and GitHub). With respect to the runtime, models in *Karate Club* show excellent scalability which we demonstrate by the use of synthetic data.

Our contributions. Specifically our work makes the following key contributions:

1. We release *Karate Club*, a Python graph mining framework which provides a wide range of easy to use community detection, node and whole graph embedding procedures.
2. We demonstrate with code the main ideas of the API oriented framework design: hyperparameter encapsulation and inspection, available public methods, dataset ingestion, output generation, and interfacing with downstream learning algorithms and evaluation methods.
3. We evaluate the learning performance of the framework on real world community detection, node and graph classification problems. We validate the scalability of the algorithms implemented in our framework.
4. We open sourced with the framework a detailed documentation and released multiple large graph classification datasets.

The remainder of this paper is structured as follows. In Section 7.2 we discuss the covered graph mining techniques. We overview the main principles behind *Karate Club* in Section 7.3 where we included detailed examples to explain these design ideas. The learning performance and scalability of the algorithms in the package are evaluated in Section 7.4. We review related data mining libraries in Section 7.5. The paper concludes with Section 7.6 where we discuss future directions. The source code of *Karate Club* can be downloaded from `https://github.com/benedekrozemberczki/karateclub`; the Python package can be installed from the *Python Package Index*. Extensive documentation is available at `https://karateclub.readthedocs.io/en/latest/` with detailed examples.

7.2 Graph mining procedures in *Karate Club*

In this section, we briefly discuss the various graph mining algorithms which are available in the 1.0 release of the *Karate Club* package.

Community detection

Community detection techniques cluster the vertices of the graph into densely connected groups of nodes. This grouping can be the final result or an input for a downstream learning algorithm (e.g. node classification or link prediction).

Karate Club currently contains several methods for non-overlapping and overlapping community detection. Non-overlapping community detection is analogous to clustering, and assumes that a node can only belong to a single group; see, for example, [112, 153, 155, 166]. While overlapping community detection is analogous to fuzzy clustering as nodes have an affiliation with multiple clusters; look for example [99, 193, 218, 232, 238].

Node embedding

Node embeddings map vertices of a graph into an Euclidean space in which those that are deemed to be similar according to a certain notion will be in close proximity. The Euclidean representation makes it easier to apply standard machine learning libraries for node classification, link prediction, clustering etc.

Neighbourhood preserving embedding maintains the proximity of nodes in the graph when an embedding is created. These methods implicitly [147, 148, 170] or explicitly [22, 85, 154, 194] decompose a proximity matrix (e.g. powers of the adjacency matrix or a sum of these matrices) to create the node embedding.

Structural embedding conserves the structural roles of nodes in the embedding space [11, 41, 75]. Nodes with similar embeddings have a similar distribution of descriptive statistics (e.g. centrality and clustering coefficient) in their vicinity. Embeddings are distilled by the decomposition of matrices representing structural features of nodes.

Attributed embedding retains the neighbourhood structure and generic feature similarity of nodes when the embedding is learned. This learning involves the joint factorization of the node-node and node-feature matrices with a direct [227, 231] or implicit matrix decomposition technique [165, 244].

Meta embedding combines information from neighbourhood preserving, structural and attributed embeddings in order to create higher representation quality embeddings [228].

Whole graph embedding and summarization

Whole graph embedding and summarization techniques create fixed size representations of entire graphs as points in a Euclidean space. Those graphs which are close in the embedding space share structural patterns such as subtrees. These representations are used for a range of graph level tasks – graph classification, regression and whole graph clustering.

Spectral fingerprints extract statistics from the eigenvectors and eigenvalues of the graph Laplacian [34, 202, 213]. Vectors of the descriptive statistics are used as the whole graph representation.

Implicit factorization techniques create a graph – structural feature matrix [27, 134] by enumerating string features in the graphs. This matrix is decomposed in order to create whole graph descriptors and feature embeddings jointly.

7.3 Design principles

When we created *Karate Club*, we used an API oriented machine learning system design point of view [20, 145] in order to make an end-user friendly machine learning tool. This API oriented design principle entails a few simple ideas. In this section we discuss these ideas and their apparent advantages with appropriate illustrative examples in great detail.

Encapsulated model hyperparameters and inspection

An unsupervised *Karate Club* model instance is created by using the constructor of the appropriate Python object. This constructor has a *default hyperparameter setting* which allows for sensible out-of-the-box model usage. In simple terms this means that the end user does not need to understand the inner model mechanics in great detail to use the methods implemented in our framework. We set these default hyperparameters to provide a reasonable learning and runtime performance. If needed, these model hyperparameters can be modified at the model instance creation time with the appropriate re-parametrization of the constructor. The hyperparameters are stored as *public attributes* to allow the inspection of model settings.

We demonstrate the encapsulation of hyperparameters by the code snippet in Figure 7.1. First, we want to create an embedding for a *NetworkX* generated Erdos-Renyi graph (line 4) with the standard hyperparameter settings. When the model is constructed and fitted (lines 6-7) we do not change default hyperparameters and we can print the standard setting of the dimensions hyperparameter (line 8). Second, we decided to set a different number of dimensions, so we created and fitted a new model (lines 10-11) and we print the new value of the dimensions hyperparameter (line 12).

```
1 import networkx as nx
2 from karateclub import DeepWalk
3
4 graph = nx.gnm_random_graph(100, 1000)
5
6 model = DeepWalk()
7 model.fit(graph)
8 print(model.dimensions)
9
10 model = DeepWalk(dimensions=64)
11 model.fit(graph)
12 print(model.dimensions)
```

FIGURE 7.1: Creating a synthetic graph, using a DeepWalk model with standard and modified hyperparameter settings.

API consistency and non-proliferation of classes

Each unsupervised machine learning model in *Karate Club* is implemented as a separate class which inherits from the *Estimator* class. Algorithms implemented in our framework have a limited number of *public methods* as we do not assume that the end user is particularly interested in the algorithmic details related to a specific technique. All models are trained by the use of the *fit* method which takes the inputs (graph, node features) and calls the appropriate private methods to learn an embedding or clustering. Node and graph embeddings are returned by the *get_embedding* public method and cluster memberships are retrieved by calling *get_memberships*.

```
1 import networkx as nx
2 from karateclub import DeepWalk
3
4 graph = nx.gnm_random_graph(100, 1000)
5
6 model = DeepWalk()
7 model.fit(graph)
8 embedding = model.get_embedding()
```

FIGURE 7.2: Creating a synthetic graph, using the DeepWalk constructor, fitting the embedding and returning it.

We avoided the proliferation of classes with two specific strategies. First, the inputs used by our framework and the outputs generated do not rely on custom data classes. This helps to prevent the unnecessary growth of the number of classes and also helps with interfacing with downstream applications. Second, algorithms which use the same data pre-processing or algorithmic step (e.g. truncated random walk, Weisfeiler-Lehman hashing) were built on shared blocks.

In Figure 7.2 we create a random graph (line 4), and DeepWalk model with the default hyperparameters (line 6), we fit this model (line 7) using the public *fit* method (line 7) and return the embedding by calling the public *get_embedding* method (line 8).

The example in Figure 7.2 can be modified to create a *Walklets* embedding with minimal effort by changing the model import (line 2) and the constructor (line 6) – these modifications result in the snippet of Figure 7.3.

Looking at these two snippets the advantage of the API driven design is evident as we only needed to do a few modifications. First, we had to change the import of the embedding model. Second, we needed to modify the model construction and the default hyperparameters were already set. Third, the public methods provided by the *DeepWalk* and *Walklets* classes behave the same way. An embedding is learned with *fit* and it is returned by *get_embedding*. This allows for quick and minimal changes to the code when an upstream unsupervised model used for feature extraction performs poorly.

Standardized dataset ingestion

We designed *Karate Club* to use standardized dataset ingestion when a model is fitted. Practically this means that algorithms which have the same purpose use the same data types for model

```
import networkx as nx
from karateclub import Walklets

graph = nx.gnm_random_graph(100, 1000)

model = Walklets()
model.fit(graph)
embedding = model.get_embedding()
```

FIGURE 7.3: Creating a synthetic graph, using the Walklets constructor, fitting the embedding and returning it.

training. In detail:

- Neighbourhood based and structural node embedding techniques use a single *NetworkX* graph as input for the fit method.
- Attributed node embedding procedures take a *NetworkX* graph as input and the features are represented as a *NumPy* array or as a *SciPy* sparse matrix. In these matrices rows correspond to nodes and columns to features.
- Graph level embedding methods and statistical graph fingerprints take a list of *NetworkX* graphs as an input.
- Community detection methods use a *NetworkX* graph as an input.

High performance model mechanics

The underlying mechanics of the graph mining algorithms were implemented using widely available Python libraries which are not operation system dependent and do not require the presence of other external libraries like *TensorFlow* or *PyTorch* does [1, 144]. The internal graph representations in *Karate Club* use *NetworkX*. Dense linear algebra operations are done with *NumPy* and their sparse counterparts use *SciPy*. Implicit matrix factorization techniques [11, 147, 148, 165, 244] utilize the *GenSim* [156] package and methods which rely on graph signal processing use *PyGSP* [39].

Standardized output generation and downstream interfacing

The standardized output generation of *Karate Club* ensures that unsupervised learning algorithms which serve the same purpose always return the same type of output with a consistent data point ordering. There is a very important consequence of this design principle. When a certain type of algorithm is replaced with the same type of algorithm, the downstream code which uses the output of the upstream unsupervised model does not have to be changed. Specifically the outputs generated with our framework use the following data structures:

- *Node embedding algorithms* (neighbourhood preserving, attributed and structural) always return a *NumPy* float array when the *get_embedding* method is called. The number of rows

in the array is the number of vertices and the row index always corresponds to the vertex index. Furthermore, the number of columns is the number of embedding dimensions.

- *Whole graph embedding methods* (spectral fingerprints, implicit matrix factorization techniques) return a *NumPy* float array when the *get_embedding* method is called. The row index corresponds to the position of a single graph in the list of graphs inputted. In the same way, columns represent the embedding dimensions.
- *Community detection procedures* return a dictionary when the *get_memberships* method is called. Node indices are keys and the values corresponding to the keys are the community memberships of vertices. Certain graph clustering techniques create a node embedding in order to find vertex clusters. These return a *NumPy* float array when the *get_embedding* method is called. This array is structured like the ones returned by node embedding algorithms.

```
1 import community
2 import networkx as nx
3 from karateclub import LabelPropagation, SCD
4
5 graph = nx.gnm_random_graph(100, 1000)
6
7 model = SCD()
8 model.fit(graph)
9 scd_memberships = model.get_memberships()
10
11 model = LabelPropagation()
12 model.fit(graph)
13 lp_memberships = model.get_memberships()
14
15 print(community.modularity(scd_memberships, graph))
16 print(community.modularity(lp_memberships, graph))
```

FIGURE 7.4: Creating a synthetic graph, clustering with two community detection techniques and using an external library to evaluate the modularity of clusterings.

We demonstrate the standardized output generation and interfacing by the code fragment in Figure 7.4. We create clusterings of a random graph and return dictionaries containing the cluster memberships. Using the external community library we can calculate the modularity of these clusterings (lines 15-16). This shows that the standardized output generation makes interfacing with external graph mining and machine learning libraries easy.

Limitations

The current design of *Karate Club* has certain limitations and we make assumptions about the input. We assume that that the *NetworkX* graph is undirected and consists of a single strongly connected component. All algorithms assume that nodes are indexed with integers consecutively and the starting node index is 0. Moreover, we assume that the graph is not multipartite, nodes are homogeneous and edges are unweighted (each edge has a unit weight).

In case of the whole graph embedding algorithms [27, 34, 51, 134, 202, 213] all graphs in the set of graphs must amend the previously listed requirements with respect to the input. The Weisfeiler-Lehman feature based embedding techniques [27, 134] allow nodes to have a single string feature which can be accessed with the *feature* key. Without the presence of this key these algorithms default to the use of degree centrality as a node feature.

7.4 Experimental evaluation

In the experimental evaluation of *Karate Club* we will demonstrate two things. First, we will show that the implemented algorithms have a good performance with respect to embedding and extracted community quality on a variety of machine learning problems. Second, we support evidence that those algorithms which in theory scale linearly with the input size (number of nodes or number of graphs) scale linearly using our framework in practice. Throughout these experiments we will always use the standard hyperparameter settings of the 1.0 release of our package.

Learning performance

The evaluation of the representation quality focuses on three types of machine learning tasks. These are: community detection with ground truth communities, node classification with the node embeddings, and whole graph classification with graph level embeddings.

TABLE 7.1: Statistics of social networks used for node level algorithms.

	Wikipedia Crocodiles	**GitHub Developers**	**Twitch England**	**Facebook Page-Page**
Nodes	11,631	37,700	7,126	22,470
Density	0.003	0.001	0.002	0.001
Transitivity	0.026	0.013	0.042	0.232
Diameter	11	7	10	15
Features	13,183	4,005	2,545	4,714

Datasets

In order to evaluate the performance of vertex level algorithms (node embedding and community detection) we used attributed web, collaboration and social networks which are publicly available on *SNAP*[1] [107, 165]. We decided to use attributed networks because a large number of algorithms in *Karate Club* can exploit the presence of node features. These datasets are the following:

- *Wikipedia Crocodiles:* In this graph nodes represent Wikipedia pages and edges are mutual links. The vertex features describe the presence of nouns in the article and the binary target variable indicates the volume of traffic on the site.

[1] https://snap.stanford.edu/data/

- *GitHub Developers:* Vertices in this network are developers who use GitHub and edges represent mutual follower relationships between the users. Features are derived based on location, biography and other metadata, the binary target variable is whether someone is a machine learning or web developer.
- *Twitch England:* Nodes of this graph are Twitch users from England and edges are mutual friendships between them. Node features were extracted based on the streaming history of the users while the binary node class describes whether the user creates explicit content.
- *Facebook Page-Page:* A network of verified Facebook pages where nodes are pages and the links between nodes are mutual likes. Features are distilled from the page descriptions and the target is the category of the Facebook page (Politicians, Governments, Companies, TV Shows).

The descriptive statistics of these node level datasets are summarized in Table 7.1. As one can see these networks have a large variety of size, level of clustering and diameter.

TABLE 7.2: Statistics of graph datasets used for graph level algorithms.

		Nodes		**Density**		**Diameter**	
Dataset	**Graphs**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Reddit Threads	203,088	11	97	0.021	0.382	2	27
Twitch Egos	127,094	14	52	0.038	0.967	1	2
GitHub StarGazers	12,725	10	957	0.003	0.561	2	18
Deezer Egos	9,629	11	363	0.015	0.909	2	2

Graph level embedding algorithms were evaluated on a variety of web and social graph datasets which we collected specifically for this paper. We made these graph collections publicly available [2]. The graph collections used for predictive performance evaluation are the following:

- *Reddit Threads:* Discussion and non-discussion based threads from Reddit which we collected in May 2018. The task is to predict whether a thread is discussion based.
- *Twitch Egos:* The ego-nets of Twitch users who participated in the partnership program in April 2018. The binary classification task is to predict using the ego-net whether the central gamer plays a single or multiple games.
- *Github Stargazers:* The social networks of developers who starred popular machine learning and web development repositories until 2019 August. The task is to decide whether a social network belongs to a web or machine learning repository.
- *Deezer Egos:* The ego-nets of Eastern European users collected from the music streaming service Deezer in February 2020. The related task is the prediction of gender for the ego node in the graph.

[2] https://github.com/benedekrozemberczki/datasets

We listed the size of these datasets with the respective descriptive statistics in Table 7.2. It is worth noting that the *Reddit Threads* and *Twitch Egos* both have at least 10 fold more graphs than the social graph datasets which are widely used for graph classification evaluation [226]. We would also like to emphasize that the use of graph kernels would not be feasible on graph datasets which are this numerous.

TABLE 7.3: Mean NMI values with standard errors on the node level datasets calculated from 100 runs.

	Wikipedia Crocodiles	**GitHub Developers**	**Twitch England**	**Facebook Page-Page**
DANMF [238]	$.051 \pm .001$	$.083 \pm .001$	$.007 \pm .001$	$.164 \pm .001$
M-NMF [218]	$.063 \pm .001$	$.084 \pm .001$	$.004 \pm .001$	$.068 \pm .001$
NNSED [193]	$.063 \pm .001$	$.034 \pm .001$	$.004 \pm .001$	$.072 \pm .001$
SymmNMF [99]	$.062 \pm .001$	$.074 \pm .001$	$.007 \pm .001$	$.206 \pm .001$
Ego-Splitting [44]	$.157 \pm .001$	$\mathbf{.202 \pm .001}$	$\mathbf{.223 \pm .001}$	$.346 \pm .001$
EdMot [112]	$.085 \pm .001$	$.180 \pm .001$	$.008 \pm .001$	$.272. \pm .001$
LabelProp [155]	$.119 \pm .001$	$.090 \pm .002$	$.003 \pm .001$	$.320 \pm .004$
SCD [153]	$\mathbf{.181 \pm .001}$	$.189 \pm .001$	$.169 \pm .001$	$\mathbf{.386 \pm .001}$
GEMSEC[166]	$.102 \pm .001$	$.127 \pm .001$	$.008 \pm .002$	$.244 \pm .001$

Community detection

We evaluate the community detection performance by running the clustering algorithms on the node level datasets. In case of overlapping community detection algorithms [44, 99, 193, 218, 232, 238] we assigned each node to the cluster that has the strongest affiliation score with the node (ties were broken randomly). The metric used for the clustering performance measurement is the average normalized mutual information (henceforth NMI) score calculated between the cluster membership vector and the factual class memberships. We report in Table 7.3 the NMI averages with the standard errors calculated from 100 experimental runs.

Looking at Table 7.3 first we notice that the non-overlapping community detection techniques [44, 112, 153, 155, 166] materially outperform the overlapping models which create latent spaces [99, 193, 218, 232, 238] on every dataset in terms of NMI. Second, those algorithms which create clusters based on the presence of closed triangles (SCD [153], Ego-Splitting [44]) have a general strong performance. Finally, on problems where it can be assumed that the class membership vector is associated with structural properties (e.g. Wikipedia Crocodiles), the overlapping latent space creating community detection methods perform poorly in terms of NMI.

Graph classification

In each dataset we created representations for the graphs and use those as predictors for the downstream classification task. We repeated the feature distillation and supervised model training 100 times, used 80% of graphs for training and 20% for testing with seeded splits. Using the graph class vectors of the test set and class probabilities outputted by the logistic regression

classifier we calculated mean area under the curve (henceforth AUC) values which are presented in Table 7.4 along with their standard errors.

TABLE 7.4: Mean AUC values with standard errors on the graph level datasets calculated from 100 seed train-test splits.

	Reddit Threads	**Twitch Egos**	**GitHub StarGazers**	**Deezer Egos**
GL2Vec [27]	$.753 \pm .002$	$.664 \pm .002$	$.551 \pm .001$	$.504 \pm .001$
Graph2Vec [134]	$.804 \pm .002$	$.702 \pm .003$	$.585 \pm .001$	$.512 \pm .001$
SF [34]	$.814 \pm .002$	$.678 \pm .003$	$.558 \pm .001$	$.501 \pm .001$
NetLSD [202]	$\mathbf{.827 \pm .001}$	$.631 \pm .002$	$.632 \pm .001$	$.522 \pm .001$
FGSD [213]	$.825 \pm .002$	$.705 \pm .003$	$.656 \pm .001$	$.526 \pm .001$
GeoScattering [51]	$.800 \pm .001$	$.697 \pm .001$	$.546 \pm .003$	$.522 \pm .003$
FEATHER [171]	$\mathbf{.830 \pm .002}$	$\mathbf{.720 \pm .003}$	$\mathbf{.748 \pm .002}$	$\mathbf{.540 \pm .001}$

The results presented in Table 7.4 show that the representations created by implicit factorization [27, 134] and spectral finger printing [34, 202, 213] techniques are predictive on most problems. In addition, we see evidence that algorithms from the latter group create somewhat higher quality representations.

Node classification

In this series of experiments we evaluated the node classification performance on the node level datasets. For each graph we learned a node embedding and used the features of this node embedding as predictors for a downstream logistic (softmax) regression model. We repeated the embedding and supervised model training 100 times, used 80% of the nodes for training and 20% for testing with seeded splits. Using the target vectors of the test set and the class probabilities outputted by the downstream model we calculated mean AUC scores. These average AUC values are reported in Table 7.5 with standard errors. The results in Table 7.5 generally demonstrate that the included neighbourhood based [17, 22, 85, 139, 147, 148, 154, 170, 194], structural role preserving [11, 41], and attributed [15, 165, 227, 231, 234, 244] node embedding techniques all generate reasonable quality representations for this classification task. There are additional conclusions; (i) multi-scale node embeddings such as *GraRep* [22], *Walklets*, [148], and *MUSAE* [165] create high quality node features , (ii) combining neighbourhood and attribute information results in the best representations [165, 244], (iii) there is not a single model which is generally superior.

Scalability

We perform scalability tests for all three types of algorithms (community detection, node and whole graph embedding). For each of these categories we investigate the scalability of 4 chosen algorithms. We use Erdos-Renyi graphs where the input size and graph density can be manipulated directly.

TABLE 7.5: Mean AUC values with standard errors on the node level datasets calculated from 100 seed train-test splits.

	Wikipedia Crocodiles	**GitHub Developers**	**Twitch England**	**Facebook Page-Page**
BoostNE [85]	$.685 \pm .001$	$.845 \pm .001$	$.576 \pm .001$	$.752 \pm .001$
NodeSketch [230]	$.722 \pm .001$	$.631 \pm .001$	$.520 \pm .001$	$.579 \pm .001$
Diff2Vec [170]	$.832 \pm .001$	$.858 \pm .001$	$.589 \pm .001$	$.873 \pm .001$
NetMF [154]	$.866 \pm .001$	$.867 \pm .001$	$.629 \pm .002$	$.946 \pm .001$
Walklets [148]	$.875 \pm .001$	$.899 \pm .002$	$.622 \pm .001$	$.973 \pm .001$
HOPE [139]	$.870 \pm .001$	$.844 \pm .001$	$.612 \pm .001$	$.909 \pm .001$
GraRep [22]	$.888 \pm .002$	$.876 \pm .001$	$.609 \pm .001$	$.952 \pm .001$
DeepWalk [147]	$.850 \pm .001$	$.872 \pm .002$	$.597 \pm .002$	$.877 \pm .001$
NMF-ADMM [194]	$.747 \pm .001$	$.784 \pm .001$	$.619 \pm .001$	$.937 \pm .001$
LAP [17]	$.784 \pm .001$	$.529 \pm .001$	$.511 \pm .001$	$.501 \pm .001$
GraphWave [41]	$.517 \pm .001$	$.620 \pm .001$	$.583 \pm .001$	$.613 \pm .001$
Role2Vec [11]	$.845 \pm .001$	$.862 \pm .002$	$.601 \pm .002$	$.903 \pm .002$
BANE [231]	$.866 \pm .002$	$.570 \pm .001$	$.551 \pm .001$	$.970 \pm .002$
TENE [234]	$.907 \pm .001$	$.874 \pm .001$	$.615 \pm .001$	$.886 \pm .001$
TADW [227]	$.896 \pm .001$	$.817 \pm .001$	$.612 \pm .002$	$.871 \pm .001$
FSCNMF [15]	$.912 \pm .001$	$.856 \pm .002$	$.621 \pm .001$	$.891 \pm .001$
SINE [244]	$.904 \pm .001$	$\mathbf{.910 \pm .002}$	$\mathbf{.646 \pm .001}$	$.979 \pm .001$
MUSAE [165]	$\mathbf{.931 \pm .001}$	$.903 \pm .001$	$.628 \pm .001$	$\mathbf{.981 \pm .001}$

Figure 7.6 plots runtime against size and density of the clustered while the average number of edges is fixed to be 10. In the densification scenario we clustered a graph with 2^{12} nodes. Non-overlapping community detection techniques show a remarkable scalability with respect to graph size increase, and we also see that the densification of the graph results in longer runtimes.

We measured the same way how the average runtime of node embedding varies with input size changes and densification and plotted these in Figure 7.5. These results show that under no preferential attachment all of the included methods scale linearly with input size changes. Moreover, implicit factorization runtimes are unaffected by the densification of the graph.

In case of the whole graph representation we plotted the average runtime as a function of the number of graphs and their size on Figure 7.7. The base graph used for the first plot had 64 nodes and 5 edges per node and for the second plot we used 2^{10} graphs. First, a takeaway is that the runtime increases linearly with the size of the dataset assuming that size of the graphs is homogeneous. Second, the spectral fingerprinting techniques [34, 213] do not scale well when the size of the graphs is increased which was expected.

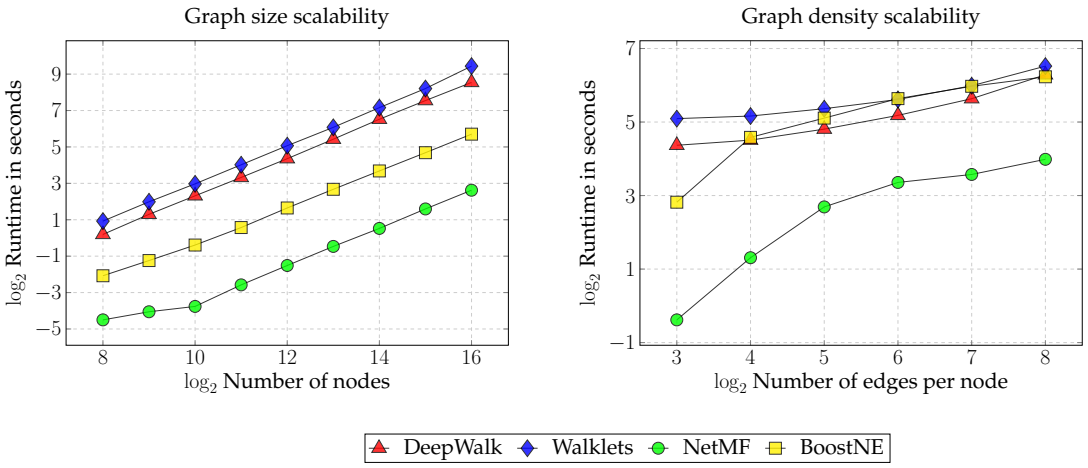


FIGURE 7.5: Scalability of node embedding procedures in Karate Club. We vary the number of nodes and the density of an Erdos-Renyi graph.

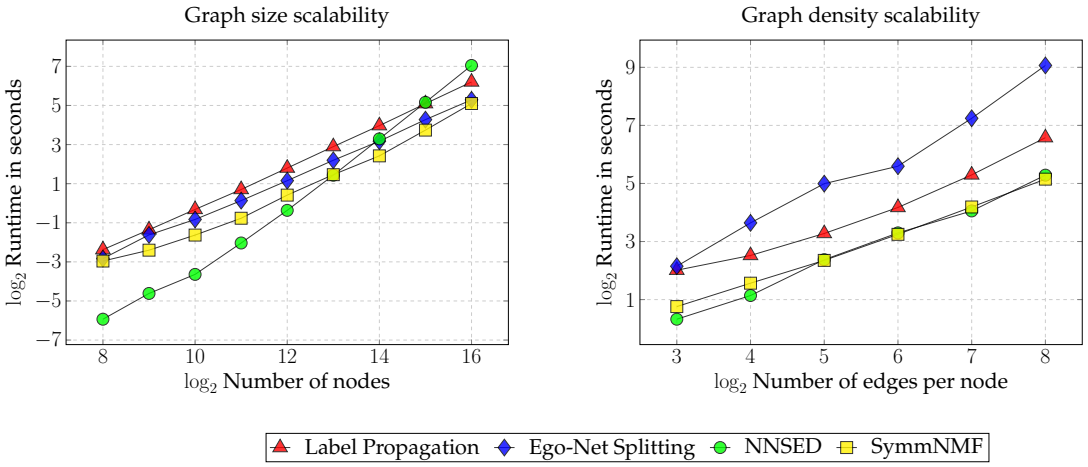


FIGURE 7.6: Scalability of the community detection procedures in Karate Club. We vary the number of nodes and the density of an Erdos-Renyi graph.

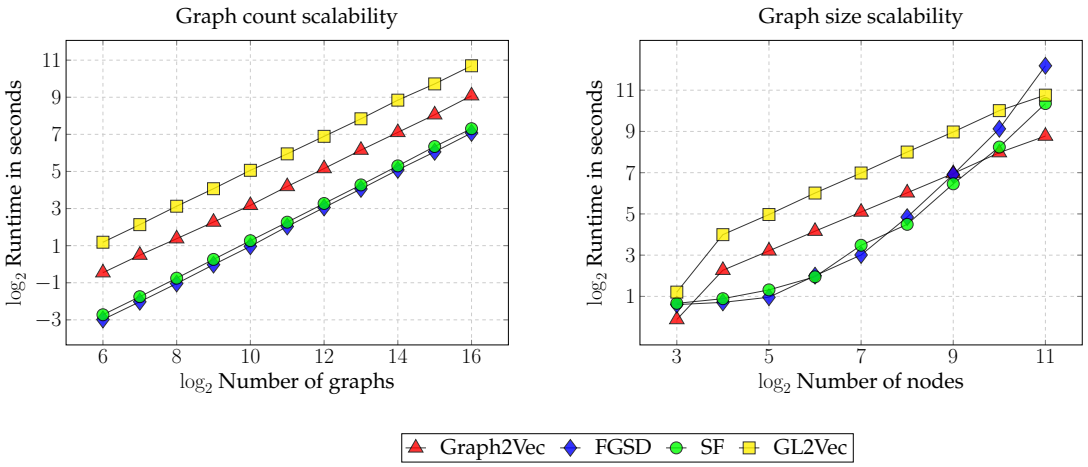


FIGURE 7.7: Scalability of graph embedding and summarization procedures in Karate Club. We vary the number of Erdos-Renyi graphs and their size.

7.5 Related work

In this section we discuss how the design of our framework is related to existing machine learning frameworks, what differentiates it from other graph mining tools.

API oriented machine learning frameworks

Scikit-learn [20, 145] is a machine learning framework with consistent and easy to use design. The *scikit-learn* models are characterised by models with a consistent API, their constructors have encapsulated sensible hyperparameters and utilize widely used Python data structures for data ingestion and output generation. This compositional design of the framework results in a low number of model classes, reusable model blocks and enables fast deployment. The *Karate Club* API draws heavily from the ideas of *scikit-learn* and the output generated is suitable as input for *scikit-learn*'s machine learning procedures.

Graph mining libraries

The *Karate Club* framework is differentiated from other graph mining libraries because of the lightweight prerequisites and the wide coverage of the learning techniques which we implemented. First, the *SNAP* and *GraphTool* packages both have C++ prerequisites which have to be pre-compiled and installed. Our framework only has Python dependencies and builds on top of the *NetworkX* project. Second, the *SNAP* [107] library only covers specific methods which were created by the authors of the framework. The *NetworkX* [68] and *GraphTool* [146] libraries only provide tools for community detection. Node and whole graph embedding is not supported by these frameworks.

7.6 Conclusion and future directions

In this work we described *Karate Club* a Python framework built on the open source packages *NetworkX* [68], *PyGSP* [39], *Gensim* [156], *NumPy* [215], and *SciPy Sparse* [214] which performs unsupervised learning on graph data. Specifically, it supports community detection, node embedding, and whole graph embedding techniques.

We discussed in detail the design principles which we followed when we created *Karate Club*, standard hyperparameter encapsulation, the assumptions about the format of input data and generated output, and the available public methods. In order to demonstrate these principles we included illustrative examples of code. In a series of experiments on real world datasets we validated that the machine learning models in *Karate Club* produce high quality clusters and embeddings. At the same time we demonstrated on synthetic data that the linear runtime algorithms scale well with increasing input size.

As discussed, *Karate Club* has certain limitations with regards to the types of graphs that it can handle. In the future we plan to extend it to operate on directed and weighted graphs. Another aim is to provide a general framework for unsupervised learning algorithms on heterogeneous, multiplex, temporal graphs and procedures for the hyperbolic embedding of nodes [176, 212].

Chapter 8

Little Ball of Fur: A Python Library for Graph Sampling

Sampling graphs is an important task in data mining. In this chapter, we describe *Little Ball of Fur* a Python library that includes more than twenty graph sampling algorithms. Our goal is to make node, edge, and exploration-based network sampling techniques accessible to a large number of professionals, researchers, and students in a single streamlined framework. We created this framework with a focus on a coherent application public interface which has a convenient design, generic input data requirements, and reasonable baseline settings of algorithms. Here we overview these design foundations of the framework in detail with illustrative code snippets. We show the practical usability of the library by estimating various global statistics of social networks and web graphs. Experiments demonstrate that *Little Ball of Fur* can speed up node and whole graph embedding techniques considerably with mildly deteriorating the predictive value of distilled features.

8.1 Introduction

Modern graph datasets such as social networks and web graphs are large and can be mined to extract detailed insights. However, the large size of the datasets pose fundamental computational challenges on graphs [61, 87]. Exploratory data analysis and computation of basic descriptive statistics can be time consuming on real world graphs. More advanced graph mining techniques such as node and edge classification or clustering can be completely intractable on full size datasets such as web graphs.

One of the fundamental techniques to deal with large datasets is sampling. On simple datasets such as point clouds, sampling preserves most of the distributional features of the data and forms the basis of machine learning [179]. However, graphs represent complex interrelations, so that naive sampling can destroy the salient features that constitute the value of the graph data. Graph sampling algorithms therefore need to be sensitive to the various features that are relevant to the downstream tasks. Such features include statistics such as diameter, clustering coefficient [42], transitivity or degree distribution. In more complex situations, graphs are used for community detection, classification, edge prediction etc [72]. A sampling algorithm should be representative with respect to such downstream learning tasks.

Various graph sampling procedures have been proposed with different objectives [79]. The implementation of the graph sampling technique and choice of its parameters used for the subgraph extraction can affect its utility for the task in question. A toolbox of well understood graph sampling techniques can make it easier for researchers and practitioners to easily perform graph sampling, and have consistent reproducible sampling across projects. Our goal is to make a large number of graph sampling techniques available to a large audience.

Present work. We release *Little Ball of Fur* – an open-source Python library for graph sampling. This is the first publicly available and documented Python graph sampling library. The general design of our framework is centered around an end-user friendly application public interface which allows for fast paced development and interfacing with other graph mining frameworks.

We achieve this by applying a few core software design principles consistently. Sampling techniques are implemented as individual classes and have pre-parametrized constructors with sensible default settings, which include the number of sampled vertices or edges, a random seed value and hyperparameters of the sampling method itself. Algorithmic details of the sampling procedures are implemented as private methods in order to shield the end-user from the inner mechanics of the algorithm. These concealed workings of samplers rely on the standard Python library and Numpy [215]. Practically, sampling techniques only provide a single shared public method (sample) which returns the sampled graph. Sampling procedures use NetworkX [68] and NetworKit [186] graphs as the input and the output adheres to the same widely used generic formats.

We demonstrate the practical applicability of our framework on various social networks and web graphs (e.g. Facebook, LastFM, Deezer, Wikipedia). We show that our package allows the precise estimation of macro level statistics such as average degree, transitivity, and degree correlation. We provide evidence that the use of sampling routines from *Little Ball of Fur* can reduce the runtime of node and whole graph embedding algorithms. Using these embeddings as input features for node and graph classification tasks we establish that the embeddings learned on the subsampled graphs extract high quality features.

Our contributions. Specifically, the main contributions of our work can be summarized as:

1. We present *Little Ball of Fur* a Python graph sampling library which includes various node, edge and exploration based subgraph sampling techniques.

2. We use code snippets to discuss the design principles which we applied when we developed our package. We examine the presence of standard hyperparameter settings, inner sampling mechanics which are implemented as private methods, the unified application public interface and the use of open-source scientific data structures.

3. We demonstrate on various real world social network and webgraph datasets how sampling a subgraph with *Little Ball of Fur* affects the estimation accuracy of graph-level macro statistics. We present real world graph mining case studies where using our sampling framework speeds up graph embedding.

4. We provide a detailed documentation of our sampling package with a tutorial and case studies with code snippets.

The rest of this chapter has the following structure. In Section 8.2 we overview the relevant literature about graph sampling. This discussion covers node, edge, and exploration sampling techniques, and the possible applications of sampling from graphs. The design principles which we followed when *Little Ball of Fur* was developed are discussed in Section 8.3 with samples of illustrative Python code. The subsampling techniques provided by our framework are evaluated in Section 8.4. We present results about network statistic estimation performance, machine learning case studies about node and graph classification. The chapter concludes with Section 8.5 where we discuss our main findings and point out directions for future work. We open sourced the software package and it can be downloaded from `https://github.com/benedekrozemberczki/littleballoffur`; the Python package can be installed via the *Python Package Index*. A comprehensive documentation can be accessed at `https://little-ball-of-fur.readthedocs.io/en/latest/` with a step-by-step tutorial.

8.2 Related work

In this section we briefly overview the types of graph subsampling techniques included in *Little Ball of Fur* and the node and graph level representation learning algorithms used for the experimental evaluation of the framework.

Graph sampling techniques

Graph subsampling procedures have three main groups – node, edge, and exploration-based techniques. We give a brief overview of these techniques in this section.

Node sampling

Techniques which sample nodes select a set of representative vertices and extract the induced subgraph among the chosen vertices. Nodes can be sampled uniformly without replacement (RN) [189], proportional to the degree centrality of nodes (RDN) [7] or according to the pre-calculated PageRank score of the vertices (PRN) [105]. All of these methods assume that the set of vertices and edges in the graph is known ex-ante.

Edge sampling

The simplest link sampling algorithm retains a randomly selected subset of edges by sampling those uniformly without replacement (RE) while another approach is to randomly select nodes and an edge that belongs to the chosen node (RNE) [98]. These techniques can be hybridized by alternating between node-edge sampling and random edge selection with a parametrized probability (HRNE) [98].

By randomly selecting a set of retained edges one implicitly samples nodes. Because of this the random edge selection can be followed up by an induction step (TIES) [10] in which the additional edges among chosen nodes are all added. This step can be a partial induction (PIES)

[10] if the edges were sampled in a streaming fashion and only edges with already sampled nodes are selected in the induction step.

Exploration based sampling

Node and edge sampling techniques do not extract representative subsamples of a graph by exploring the neighbourhoods of seed nodes. In comparison exploration based sampling techniques probe the neighborhood of a seed node or a set of seed vertices.

A group of exploration based sampling techniques uses search strategies on the graph to extract a subsample. The simplest search based strategies include classical traversal methods such as breadth first search (BFS) and depth first search (DFS) [40]. Snowball sampling (SB) [62] is a restricted version of BFS where a maximal fixed k number of neighbors is traversed. Forest fire (FF) sampling [106] is a parametrized stochastic version of SB sampling where the constraint on the maximum number of traversed neighbours only holds in expectation. A local greedy search based technique is community structure expansion sampling [121] (CSE) which starting with a seeding node adds new nodes to the sampled set which can reach the largest number of unknown nodes. Another simpler search based sampling technique is the random node-neighbor (RNN) [105] algorithm which randomly selects a set of seed nodes, takes the neighbors in a single hop and induces the edges of the resulting vertex set. Searching for shortest paths (SP) [157] between randomly sampled nodes can be used for selecting sequences of nodes and edges to induce a subsampled graph.

A large family of exploration based graph sampling strategies is based on random walks (RW) [60]. These techniques initiate a random walker on a seed node which traverses the graph and induces a subgraph which is used as the sample. Random walk based sampling has numerous shortcomings and a large number of sampling methods tries to correct for specific limitations.

One of the major limitations is that random walks are inherently biased towards visiting high degree nodes in the graph [79], Metropolis-Hastings random walk (MHRW) [82, 190] and its rejection constrained variant (RC-MHRW) [113] address this bias by making the walker prone to visit lower degree nodes.

Another major shortcoming of random walk based sampling is that the walker might get stuck in the closely knit community of the seed node. There are multiple ways to overcome this. The first one is the use of non-backtracking random walks (NBTRW) [101] which removes the tottering behaviour of random walks. The second one is circulating the neighbors of every node with a vertex specific queue (CNRW) [250]. A third strategy involves teleports - the random walker jumps (RWJ) [158] with a fixed probability to a random node from the current vertex. A fourth approach is to make the walker biased towards weak links by creating a common neighbor aware random walk sampler (CNARW) [114] which is biased towards neighbors with low neighborhood overlap. A fifth strategy is using multiple random walkers simultaneously which form a so called frontier of random walkers (FRW) [158]. These techniques can be combined with each other in a modular way to overcome the limitations of random walk based sampling.

There are other possible modifications to traditional random walks which we implemented in *Little Ball of Fur*. One example is random walk with restart (RWR) [105], which is similar to RWJ

sampling, but the teleport always ends with the seed node or loop erased random walks (LERW) [221] which can sample spanning trees from a source graph uniformly.

Node and whole graph embedding

Our experimental evaluation includes node and graph classification for which we use features extracted with proximity preserving node embeddings and whole graph embedding techniques.

Proximity preserving node embedding

Given a graph $G = (V, E)$ proximity preserving node embedding techniques [22, 66, 139, 147, 148, 170, 198] learn a function $f : V \to \mathbb{R}^d$ which maps the nodes $v \in V$ into a d dimensional Euclidean space. In this embedding space a pre-determined notion of node-node proximity is approximately preserved by learning the mapping. The vector representations created by the embedding procedure can be used as input features for node classifiers.

Whole graph embedding and statistical fingerprinting

Starting with a set of graphs $\mathcal{G} = (G_1, \ldots, G_n)$ whole graph embedding and statistical fingerprinting procedures [27, 134, 171, 202, 213] learn a function $h : \mathcal{G} \to \mathbb{R}^d$ which maps the graphs $G \in \mathcal{G}$ to a d dimensional Euclidean space. In this space those graphs which have similar structural patterns are close to each other. The vector representations distilled by these whole graph embedding techniques are useful inputs for graph classification algorithms.

8.3 Design principles

We overview the core design principles that we applied when we designed *Little Ball of Fur*. Each design principle is discussed with illustrative examples of Python code which we explain in detail.

Encapsulated hyperparameters, random seeding, and inspection

Graph sampling methods in *Little Ball of Fur* are implemented as individual classes which all inherit from the *Sampler* class. A *Sampler* object is created by using the constructor which has default out-of-the-box hyperparameter settings. These default settings are available in the documentation and can be customized by re-parametrizing the *Sampler* constructor. The hyperparameters of the sampling techniques are stored as *public attributes* of the *Sampler* instance which allows for inspection of the hyperparameter settings by the user. Each graph sampling procedure has a seed parameter – this value is used to set a random seed for the standard Python and NumPy random number generators. This way the subsample extracted from a specific graph with a fixed seed is always going to be the same.

The code snippet in Figure 8.1 illustrates the encapsulated hyperparameter and inspection features of the framework. We start the script by importing a simple random walk sampler from the package (line 1). We initialize the first random walk sampler instance without changing the

default hyperparameter settings (line 3). As the seed and hyperparameters are exposed we can print the seed parameter which is a public attribute of the sampler (line 4) and we can see the default value of the seed. We create a new instance with a parametrized constructor which sets a new seed (line 6) which modifies the value of the publicly available random seed (line 7).

```
1 from littleballoffur import RandomWalkSampler
2
3 sampler = RandomWalkSampler()
4 print(sampler.seed)
5
6 sampler = RandomWalkSampler(seed=41)
7 print(sampler.seed)
```

FIGURE 8.1: Re-parametrizing and initializing the constructor of a sampler by changing the seed.

Achieving API consistency and non proliferation of classes

The graph sampling procedures included in *Little Ball of Fur* are implemented with a consistent application public interface. As we discussed the parametrized constructor is used to create the sampler instance and the samplers all have a single available *public method*. The subsample of the graph is extract by the use of the *sample* method which takes the source graph and calls the private methods of the sampling algorithm.

We limited the number of classes and methods in *Little Ball of Fur* with a straightforward design strategy. First, the graph sampling procedures do not rely on custom data structures to represent the input and output graphs. Second, inheritance from the *Sampler* ensures that private methods which check the input format requirements do not have to be re-implemented on a case-by-case basis for each sampling procedure.

```
1 import networkx as nx
2 from littleballoffur import RandomWalkSampler
3
4 graph = nx.watts_strogatz_graph(1000, 10, 0)
5
6 sampler = RandomWalkSampler()
7 sampled_graph = sampler.sample(graph)
8
9 print(nx.transitivity(sampled_graph))
```

FIGURE 8.2: Using a random walk sampler on a Watts-Strogatz graph without changing the default sampler settings.

In Figure 8.2, first we import *NetworkX* and the random walk sampler from *Little Ball of Fur* (lines 1-2). Using these libraries we create a Watts-Strogatz graph (line 4) and a random walk sampler with the default hyperparameter settings of the sampling procedure (line 6). We sample a subgraph with the public *sample* method of the random walk sampler (line 7) and print the transitivity calculated for the sampled subgraph (line 8).

```
1 import networkx as nx
2 from littleballoffur import ForestFireSampler
3
4 graph = nx.watts_strogatz_graph(1000, 10, 0)
5
6 sampler = ForestFireSampler()
7 sampled_graph = sampler.sample(graph)
8
9 print(nx.transitivity(sampled_graph))
```

FIGURE 8.3: Using a forest fire sampler on a Watts-Strogatz graph without changing the default sampler settings.

The piece of code presented in Figure 8.2 can be altered seamlessly to perform Forest Fire sampling by modifying the sampler import (line 2) and changing the constructor (line 7) – these modifications result in the example in Figure 8.3.

These illustrative sampling pipelines presented in Figures 8.2 and 8.3 demonstrate the advantage of maintaining API consistency for the samplers. Changing the graph sampling technique that we used only required minimal modifications to the code. First, we replaced the import of the sampling technique from the *Little Ball of Fur* framework. Second, we used the constructor of the newly chosen sampling technique to create a sampler instance. Finally, we were able to use the shared *sample* method and the same pipeline to calculate the transitivity of the sampled graph.

Backend deployment, standardized dataset ingestion and limitations

Little Ball of Fur was implemented with a backend wrapper. Sampling procedures can be executed by the *NetworKit* [186] or *NetworkX* [68] backend libraries depending on the format of the input graph. Basic graph operations such as random neighbor or shortest path retrieval of the backend libraries have standardized naming conventions, data generation and ingestion methods. The generic backend wrapper based design allows for the future inclusion of other general graph manipulation backend libraries such as *GraphTool* [146] or *SNAP* [109].

The shared public *sample* method of the node, edge and exploration based sampling algorithms takes a *NetworkX*/*Networkit* graph as input and the returned subsample is also a *NetworkX*/*Networkit* graph. The subsampling does not change the indexing of the vertices.

The rich ecosystem of graph subsampling methods and the consistent API required that *Little Ball of Fur* was designed with a limited scope and we made restrictive assumptions about the input data used for sampling. Specifically, we assume that vertices are indexed numerically, the first index is 0 and indexing is consecutive. We assume that the graph that is passed to the sample method is undirected and unweighted (edges have unit weights). In addition, we assume that the graph forms a single strongly connected component and orphaned nodes are not present. Heterogeneous, multiplex, multipartite and attributed graphs are not handled by the 1.0 release of the sampling framework.

The sampler classes all inherit private methods that check whether the input graph violates the listed assumptions. These are called within the *sample* method before the sampling process

itself starts. When any of the assumptions is violated an error message is displayed for the end-user about the wrong input and the sampling is halted.

8.4 Experimental evaluation

To evaluate the sampling algorithms implemented in *Little Ball of Fur* we perform a number of experiments on real world social networks and webgraphs. Details about these datasets are discussed with descriptive statistics. We show how randomized spanning tree sampling can be used to speed up node embedding without reducing predictive performance. Our ablation study about graph classification demonstrates how connected graph subsampling can accelerate the application of whole graph embedding techniques. Finally, we present results about estimating graph level descriptive statistics.

Datasets

We use real world social network and webgraph datasets to compare the performance of sampling procedures and test their utility for speeding up classification tasks.

Node level datasets

The datasets used for graph statistic estimation and node classification are all available on SNAP [107], and descriptive statistics can be found in Table 8.1.

- **Facebook Page-Page** [165] is a webgraph of verified official Facebook pages. Nodes are pages representing politicians, governmental organizations, television shows and companies while the edges are links between the pages. The related task is multinomial node classification for the 4 page categories.
- **Wikipedia Crocodiles** [165] is a webgraph of Wikipedia pages about crocodiles. Nodes are the pages and edges are mutual links between the pages. The potential task is binary node classification
- **LastFM Asia** [171] is a social network of LastFM (English music streaming service) users who are located in Asian countries. Nodes are users and links are mutual follower relationships. The task on this dataset is multinomial node classification – one has to predict the location of the users.
- **Deezer Hungary** [166] is a social network of Hungarian Deezer (French music streaming service) users. Nodes are users located in Hungary and edges are friendships. The relevant task is multi-label multinomial node classification - one has to list the music genres liked by the users.

TABLE 8.1: Statistics of social networks used for comparing sampling and node classification algorithms.

	Facebook Page-Page	Wikipedia Crocodiles	LastFM Asia	Deezer Hungary
Nodes	22,470	11,631	7,624	47,538
Density	0.0007	0.0025	0.0010	0.0002
Transitivity	0.2323	0.0261	0.1786	0.0929
Diameter	15	11	15	12
Labels	4	2	18	84

Graph level datasets

Our classification study on subsampled sets of graphs utilized forum threads and small sized social networks of developers. The respective descriptive statistics of these datasets are in Table 8.2.

- **Reddit Threads 10K** [168] is a random subsample of 10 thousand graphs from the original Reddit threads datasets. Threads can be discussion and non-discussion based and the task is the binary classification of them according to these two categories.
- **GitHub StarGazers** [168] is a set of small sized social networks. Each social network is a community of developers who starred a specific machine learning or web development package on Github. The task is to predict the type of the repository based on the community graph.

TABLE 8.2: Descriptive statistics and size of the graph datasets for graph subsampling and whole graph classification.

		Nodes		Density		Diameter	
Dataset	**Graphs**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Reddit Threads 10K	10,000	11	97	0.021	0.291	2	22
GitHub StarGazers	12,725	10	957	0.003	0.561	2	18

Node classification with spanning tree embeddings

Node embedding vectors [147, 148] are useful compact descriptors of vertex neighbourhoods when it comes to solving classification problems. In traditional classification scenarios the whole graph is used to learn the node embedding vectors. In this experiment we study a situation where the embedding vectors are learned from a randomly sampled spanning tree of the original graph. We compare the predictive value of node embeddings learned on the whole graph with ones learned from spanning trees extracted with randomized BFS [98], DFS [98] and LERW [221]. The main advantage of randomized spanning trees is that storing the whole graph requires $\mathcal{O}(|E|)$ memory when we learn the node embedding. In contrast storing a sampled spanning tree only requires $\mathcal{O}(|V|)$ space.

Experimental settings

The experimental pipeline which we used for node classification has four stages.

1. *Graph sampling.* The BFS, DFS and LERW sample based embeddings start with the extraction of a random spanning tree with *Little Ball of Fur*. This sample is fed to the embedding procedure.

2. *Upstream model.* The sampled graph is fed to the unsupervised upstream models DeepWalk [147] and Walklets [148] which learn the neighbourhood preserving node embedding vectors. We used the Karate Club [168] implementation of these models with the default hyperparameter settings.

3. *Downstream model.* We fed the node embedding vectors as input features for a logistic regression classifier – we used the scikit-learn implementation [145] with the default hyperparameter settings. The downstream models were trained with a varying ratio of nodes.

4. *Evaluation.* We report average AUC values on the test set calculated from a 100 seeded sampling, embedding and downstream model training runs.

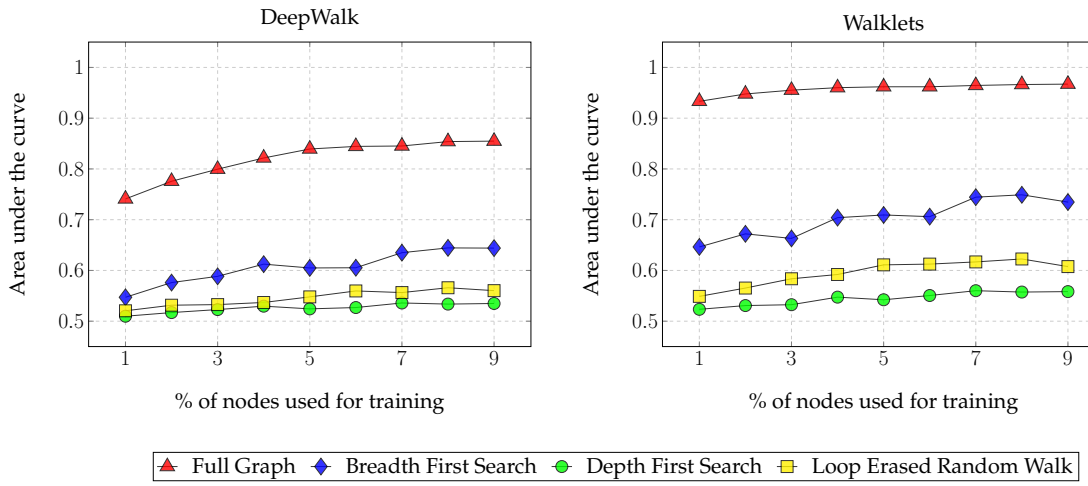


FIGURE 8.4: Node classification performance on the Facebook Page-Page graph [165] evaluated by average AUC scores on the test set calculated from a 100 seeded experimental runs.

Experimental results

We report the predictive performance for the Facebook Page-Page and LastFM Asia graphs respectively on Figures 8.4 and 8.5. First, we see that the features extracted from the BFS, DFS and LERW sampled spanning trees are less valuable for node classification based on the predictive performance on these two social networks. In plain words node embeddings of randomly sampled spanning trees produce inferior features. Second, the marginal gains of additional training data are smaller when the embedding is learned from a subsampled graph. Third, DFS sampled node embedding features have a considerably lower quality compared to the BFS and LERW sampled node embedding features. Finally, the Walklets based higher order proximity preserving embeddings have a superior predictive performance compared to the DeepWalk based

ones even when the graph being embedded is a randomly sampled spanning tree of the source graph.

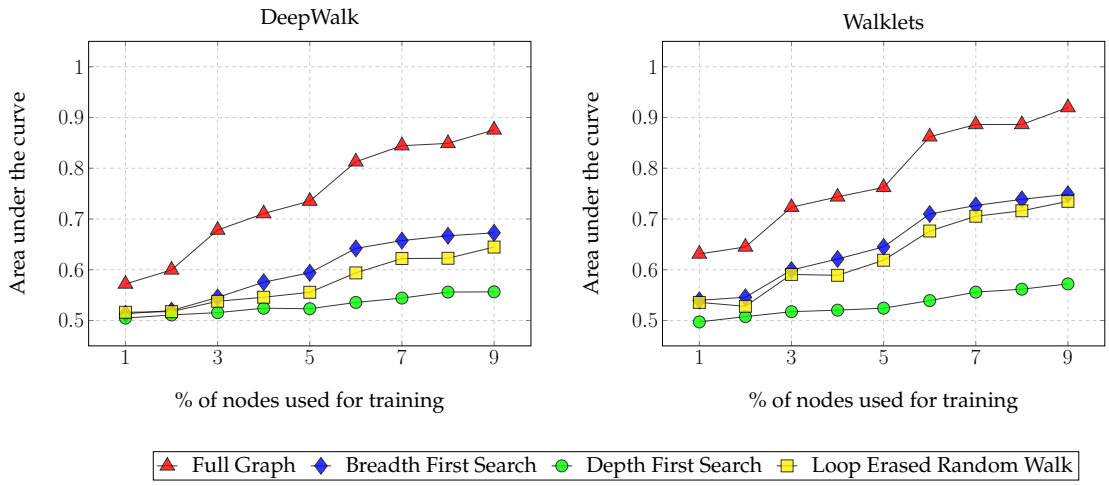


FIGURE 8.5: Node classification performance on the LastFM Asia graph [171] evaluated by average AUC scores on the test set calculated from a 100 seeded experimental runs.

An ablation study of graph classification

Graph classification procedures use the whole graph embedding vectors as input to discriminate between graphs based on structural patterns. Using subsamples of the graphs and extracting structural patterns from those can speed up this classification process. We will investigate how exploration based sampling techniques perform when they are used to obtain the samples used for the embedding.

Experimental settings

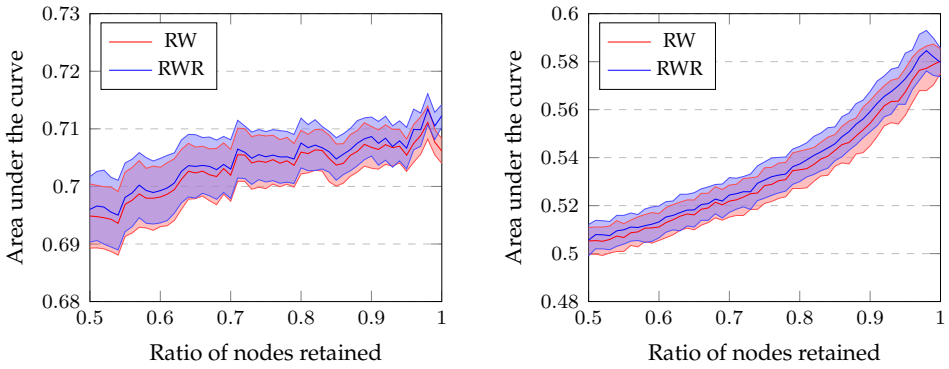
The data processing which we used for the evaluation of graph classification performance has four stages.

1. *Graph sampling.* We sample both datasets using the RW [60] and RWR [105] methods implemented in *Little Ball of Fur* 100 times for each retainment rate with different random seeds. Using these algorithms ensures that the graphs' connectivity patterns are unchanged.

2. *Upstream model.* Following the sampling, all of the samples are embedded using the Graph2Vec [134] algorithm. This procedure uses the presence of subtrees as structural patterns.

3. *Downstream model.* With the embedding vectors as covariates, we estimate a logistic regression for each dataset and retainment rate. We rely on the scikit-learn implementation [145] of the classifier with the default hyperparameter settings. We use 80% of the graphs to train the classifier.

4. *Evaluation.* The classification performance is evaluated using the AUC based on the remaining 20% of graphs which form the test set.

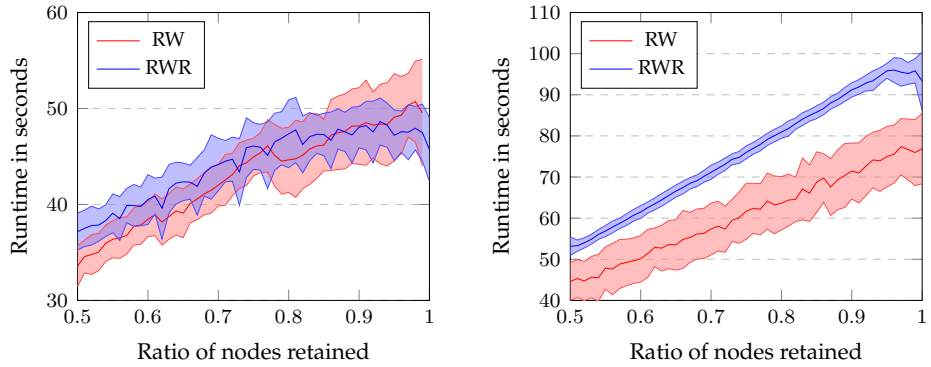


(A) Reddit Threads (B) Github Stargazers

FIGURE 8.6: Graph classification performance on the Reddit Threads and GitHub Stargazers graph datasets [168] evaluated by average AUC scores on the test set calculated from 100 seeded experimental runs. We also report standard deviations around the mean performance.

Experimental results

We report mean AUC values along with a standard deviation band in Figure 8.6 for the Reddit Threads and Github Stargazers datasets with the Random Walk and Random Walk with Restart sampling methods. Lower retainment rate is associated with a lower classification performance, as it can be expected. The more ragged, step function-like pattern observed in case of the Reddit threads dataset is likely to be due to the interplay of structural pattern downsampling and the generally smaller graphs in the dataset.



(A) Reddit Threads (B) Github Stargazers

FIGURE 8.7: Graph embedding runtime on the Reddit Threads and GitHub Stargazers graph datasets [168] calculated from 100 experimental runs. We also report standard deviations around the mean performance.

We report the mean runtime of the graph embedding process with a band of standard deviations in Figure 8.7. As we decrease the retainment rate, a significant decrease in runtime is prevalent. There is a clear trade-off between runtime and predictive performance. It is, however, worth noting that while the runtime associated with the 50% retainment rate is, in most cases close to half of the runtime using the whole graphs, the loss in classification power in case of the Reddit Threads dataset is less significant.

Estimating descriptive statistics

A traditional task for the evaluation of graph sampling algorithms is the estimation of graph level descriptive statistics. The graph level descriptive statistic is calculated for the sampled graph and it is compared to the ground truth value which is calculated based on the whole set of nodes and edges. A well performing sampling procedure is ought to give a precise estimate for the graph level quantity of interest with a reasonably small subsample of the graph.

Experimental settings

The pipeline used for estimating the graph level descriptive statistics had two stages.

1. *Graph sampling.* Node and exploration based sampling procedures sample 50% of vertices, while the edge sampling techniques select 50% of the edges to extract a subgraph.

2. *Descriptive statistic estimation.* We calculated the average of the clustering coefficient (transitivity), average node degree and the degree correlation for the sampled graphs. We did 10 seeded experimental runs to get an estimate of statistics and calculated standard errors.

TABLE 8.3: Ground truth and estimated descriptive statistics of the Facebook Page-Page and Wikipedia Crocodile networks. We calculated average statistics from 10 seeded experimental runs and included the standard errors below the mean with ground truth values based on the whole graph (first block) with estimates obtained with node (second block), edge (third block) and exploration (fourth and fifth blocks) sampling algorithms. Bold numbers denote for each category the best estimate for a given dataset.

	Facebook Page-Page			**Wikipedia Crocodiles**		
	Clustering Coefficient	**Average Degree**	**Degree Correlation**	**Clustering Coefficient**	**Average Degree**	**Degree Correlation**
Truth	0.232	15.205	0.085	0.026	29.365	−0.277
RN [189]	**0.229** 0.002	7.531 0.060	**0.070** 0.003	**0.028** 0.001	14.293 0.388	**-0.284** 0.008
DRN [7]	0.261 0.001	21.514 0.021	0.080 0.001	0.038 0.001	40.750 0.054	−0.324 0.001
PRN [105]	0.270 0.001	**16.228** 0.032	0.136 0.001	0.049 0.001	**32.370** 0.074	−0.290 0.001
RE [98]	0.116 0.001	8.470 0.004	**0.084** 0.001	0.013 0.001	15.462 0.009	**-0.277** 0.001
RNE [98]	0.092 0.001	7.602 0.001	−0.075 0.001	0.007 0.001	14.682 0.001	−0.231 0.001
HRNE [98]	0.081 0.001	7.194 0.002	−0.005 0.001	0.007 0.001	13.550 0.005	−0.234 0.001
TIES [10]	0.235 0.001	16.564 0.007	0.083 0.001	**0.026** 0.001	30.720 0.016	−0.278 0.001
PIES [10]	**0.231** 0.001	**15.357** 0.008	0.087 0.001	**0.026** 0.001	**29.142** 0.015	−0.283 0.001
RW [60]	0.255 0.001	22.535 0.022	0.073 0.001	0.036 0.001	41.648 0.196	−0.325 0.001
RWR [105]	0.253 0.003	20.282 0.293	0.092 0.007	0.043 0.003	38.967 0.549	−0.313 0.006
RWJ [158]	0.271 0.001	18.615 0.036	0.123 0.001	0.047 0.001	34.475 0.074	−0.297 0.001
MHRW [82, 190]	0.280 0.002	**17.903** 0.113	0.134 0.003	0.119 0.002	**29.914** 0.223	−0.146 0.004
RC-MHRW [113]	0.266 0.001	21.070 0.064	0.106 0.002	0.072 0.001	35.758 0.159	−0.254 0.002
FRW [158]	0.063 0.001	5.745 0.059	0.069 0.004	0.004 0.001	5.813 0.058	**-0.280** 0.003
CNRW [250]	0.255 0.001	22.590 0.038	0.072 0.001	0.037 0.001	41.645 0.104	−0.324 0.001
CNARW [114]	**0.239** 0.001	21.117 0.033	**0.082** 0.001	**0.026** 0.001	41.064 0.101	−0.348 0.001
NBT-RW [101]	0.257 0.001	22.353 0.048	0.076 0.001	0.038 0.001	41.264 0.144	−0.322 0.001
SB [62]	**0.238** 0.002	20.671 0.223	0.069 0.004	0.057 0.004	37.278 0.576	−0.292 0.009
FF [106]	**0.238** 0.001	19.219 0.089	**0.079** 0.002	0.074 0.002	**33.190** 0.262	−0.227 0.006
CSE [121]	0.229 0.002	**13.116** 0.046	0.070 0.003	**0.026** 0.001	20.314 0.345	**-0.290** 0.006
SP [157]	0.221 0.001	12.842 0.062	0.106 0.002	0.037 0.001	23.451 0.125	−0.292 0.001

Experimental results

The ground truth and estimated descriptive statistics are enclosed in Tables 8.3 and 8.4 for all of the node level datasets. Blocks of rows correspond to node, edge, random walk based and non random walk based exploration sampling algorithms. In each block of methods bold numbers denote the best performing sampling technique (closest to the ground truth) in a given category for a specific estimated descriptive statistic and dataset.

TABLE 8.4: Ground truth and estimated descriptive statistics of the LastFM Asia and Deezer Hungary networks. We calculated average statistics from 10 seeded experimental runs and included the standard errors below the mean with ground truth values based on the whole graph (first block) with estimates obtained with node (second block), edge (third block) and exploration (fourth and fifth blocks) sampling algorithms. Bold numbers denote for each category the best estimate for a given dataset.

	LastFM Asia			**Deezer Hungary**		
	Clustering Coefficient	**Average Degree**	**Degree Correlation**	**Clustering Coefficient**	**Average Degree**	**Degree Correlation**
Truth	0.179	7.294	0.017	0.093	9.377	0.207
RN [189]	**0.177** 0.004	3.642 0.032	**0.020** 0.010	**0.092** 0.001	4.699 0.010	0.190 0.002
DRN [7]	0.231 0.001	9.531 0.020	0.045 0.001	0.102 0.001	**8.551** 0.007	**0.211** 0.001
PRN [105]	0.236 0.001	**8.209** 0.022	0.064 0.002	0.098 0.001	7.251 0.007	0.231 0.001
RE [98]	0.090 0.001	4.422 0.009	**0.019** 0.002	0.046 0.001	5.018 0.001	0.183 0.001
RNE [98]	0.046 0.001	3.674 0.001	−0.108 0.001	0.042 0.001	4.701 0.001	0.056 0.001
HRNE [98]	0.046 0.001	3.562 0.003	−0.070 0.002	0.039 0.001	4.501 0.001	0.095 0.001
TIES [10]	0.190 0.001	8.218 0.010	0.027 0.001	**0.094** 0.001	**9.748** 0.001	0.204 0.001
PIES [10]	**0.186** 0.001	**7.247** 0.010	0.037 0.001	0.086 0.001	8.501 0.003	**0.209** 0.001
RW [60]	0.224 0.001	9.878 0.021	0.039 0.003	0.104 0.001	9.221 0.008	0.218 0.001
RWR [105]	0.222 0.003	9.078 0.087	0.036 0.004	0.098 0.001	9.122 0.082	**0.213** 0.003
RWJ [158]	0.233 0.001	9.012 0.032	0.067 0.003	0.102 0.001	8.351 0.016	0.233 0.002
MHRW [82, 190]	0.232 0.001	**8.854** 0.041	0.102 0.007	0.106 0.001	7.761 0.023	0.242 0.003
RC-MHRW [113]	0.232 0.001	9.594 0.031	0.078 0.003	0.103 0.001	**8.553** 0.021	0.237 0.002
FRW [158]	0.084 0.001	4.485 0.032	**0.018** 0.008	0.033 0.001	3.243 0.005	0.091 0.002
CNRW [250]	0.223 0.001	9.924 0.018	0.036 0.002	0.104 0.001	9.254 0.017	0.218 0.001
CNARW [114]	**0.220** 0.001	9.508 0.032	0.052 0.002	**0.094** 0.001	9.140 0.013	0.218 0.001
NBT-RW [101]	0.226 0.001	9.818 0.027	0.049 0.002	0.104 0.001	9.106 0.010	0.230 0.001
SB [62]	0.207 0.002	9.131 0.121	−0.008 0.005	**0.093** 0.001	**9.913** 0.103	0.122 0.003
FF [106]	0.204 0.001	9.034 0.025	0.051 0.001	0.096 0.001	10.120 0.013	0.197 0.001
CSE [121]	**0.191** 0.003	6.544 0.055	**0.006** 0.006	0.089 0.002	6.554 0.001	0.165 0.001
SP [157]	0.203 0.001	**7.258** 0.032	0.043 0.002	0.079 0.001	8.176 0.007	**0.209** 0.001

We can make a few generic observations about the quality of descriptive statistic estimates. First, there is not a clearly superior sampling technique. This holds generally and within all of the main categories of considered algorithms. Specifically, there is not a superior node, edge or expansion based sampling procedure. Second, the induction based edge sampling techniques (TIES and PIES) give a good estimate of the statistics, but the induction step includes more than 50% of the edges. Because of this, the obtained good estimation performance is somewhat misleading as the majority of edges is still retained after the induction step. Third, edge sampling algorithms sometimes fail to estimate the direction of the degree correlation properly. Finally, the random walk based techniques generally tend to overestimate the average degree. This is not

surprising considering that these are biased towards high degree nodes.

8.5 Conclusion and future directions

In this chapter we described *Little Ball of Fur* – an open-source Python graph sampling framework built on the widely used scientific computing libraries NetworkX [68], NetworKit [186], and NumPy [215]. In detail it provides techniques for node, edge, and exploration based graph sampling.

We reviewed the general conventions which we used for implementing graph sampling algorithms in *Little Ball of Fur*. The framework offers a limited number of public methods, ingests and outputs data in widely used graph formats, and embodies preset default hyperparameters. We presented practical implications of these design features with illustrative examples of Python code snippets. Using various social networks and web graphs we had shown that using sampled graphs extracted with *Little Ball of Fur* one can approximate ground truth graph level statistics such as transitivity and the degree correlation coefficient. We also found evidence that sampling subgraphs with our framework can accelerate node and graph classification without extremely reducing the predictive performance.

As we have emphasized *Little Ball of Fur* assumes that the inputted graph is undirected and unweighted. In the future we envision to relax these assumptions about the input. We plan to include additional high performance backend libraries such as SNAP [109] and GraphTool [146]. Furthermore, we aim to extend our framework by including multiplex [59], attributed, and heterogeneous [111, 229] graph sampling algorithms with new releases of the framework.

Chapter 9

Conclusions

This chapter begins with a summary of the significant contributions of this thesis to the field of graph mining including sub areas such as: node embeddings, statistical graph descriptors, graph neural networks, explainable machine learning, and graph mining software. We also highlight promising future research directions that can considerably extend the contributions presented in this thesis.

9.1 The significance of contributions

In this thesis we developed novel machine learning techniques and software for scalable, expressive and explainable graph mining. In Part I we focused on the design of unsupervised and supervised learning algorithms which can extract valuable features from graph structured data. Part II discussed the design principles and applications of research oriented graph mining software. Specifically the main contributions of the thesis are:

- **Graph Embedding with Self-Clustering.** In Chapter 2 we introduced a node embedding technique which uses implicit factorization to learn clustered latent space node representations. This algorithm was the first of its kind to separate the number of embedding dimensions and clusters; earlier approaches did not disentangle these hyperparameters from each other.

- **Multi-Scale Attributed Node Embedding.** In Chapter 3 we proposed the first attributed node embedding algorithm which can learn multi-scale vertex representation contextualized by vertex attributes in an unsupervised way. This line of research was the first to show the theoretical connection between attributed and proximity-preserving embedding techniques.

- **Characteristic Functions on Graphs.** In Chapter 4 we generalized the idea of characteristic functions to describe the distribution of features in neighbourhoods of nodes. Exploiting this generalization we introduced unsupervised node embeddings and discriminative parametric models that operate on graph structured data.

- **Pathfinder Discovery Network.** In Chapter 5 we described the general design of an end-to-end differentiable graph neural network layer that operates on multiplex graphs. Our contribution was the first to point out that a number of well-established graph neural network architectures are special corner cases of this general abstract design.

- **Ensemble Games.** In Chapter 6 we defined a novel class of co-operative games in which machine learning models make classification decisions in an ensemble. We solved these games with the Shapley value to quantify the value of classifiers in voting based ensembles.
- **Karate Club.** In Chapter 7 we described a machine learning library which provides node embedding, whole graph embedding and community detection algorithms. It is the only open-source graph mining library which provides tools for graph level embeddings and statistical descriptors.
- **Little Ball of Fur.** In Chapter 8 we overviewed the design of the first graph sampling library which made node, edge and exploration sampling techniques available for graph mining researchers.

9.2 Future research directions

The contributions presented in this thesis suggest a number of novel avenues for future research. We are particularly excited about certain potential ideas which can have high academic and industry impact. Hence, we are going to highlight additional interesting directions for future graph mining research. These ideas in themselves have sufficient depth to be formed into individual research papers and stand on their own.

Multi-scale node embedding with self-clustering

Early research into proximity-preserving node embedding did not consider proximity at different scales [66, 147, 198]. On the contrary, multi-scale node representation learning techniques [5, 22, 148, 165] are able to describe the location of vertices in the embedding space with features which have a varying level of coarseness. This heterogeneous coarseness had been proven to be a very expressive feature extraction strategy, as multi-scale node features are highly predictive as demonstrated by our results in Chapters 2, 3 and 4 had shown it. Creating a multi-scale node embedding technique which extends the techniques presented in Chapter 2 would be an important algorithmic contribution: it would be a node embedding procedure which learns a multi-scale parametric node embedding and a hierarchical clustering jointly.

Graph characteristic function generalizations

The characteristic functions defined on graphs presented in Chapter 4 are based on a flexible theoretical concept. In our work we only considered a specific case of characteristic functions, ones where the probability weights are defined by truncated random walk transition probabilities. Designing graph characteristic functions which generalize the basic concept that we introduced could lead to novel representation learning techniques. Specifically we see the following research directions:

- *Novel probability weight definitions:* Re-parametrizing the characteristic functions with proximity measures would be an important and straightforward extension of our work. Defining weights derived from generic vertex attributes would be another interesting path.

- *Pooling node features:* Characteristic functions can describe the shape of any distribution with a differentiable function. Hence, defining differentiable graph pooling operators could enrich the existing geometric deep learning literature on graph pooling functions.
- *Fast convolutions:* The convolution of vertex feature could be calculated by multiplying the characteristic functions of vertex features together. We did not exploit this particular property of characteristic functions in our research.

Theory of multiplex neural message passing

The Pathfinder Discovery Networks described in Chapter 5 do not create a general theory of neural message passing [58] on multiplex, multi-view and multi-layer graphs. These techniques describe a class of graph neural networks which can operate on graph data which exhibit multiplexity. However, using the generic ideas described in Chapter 5 and [58] one could create a general construct for neural message passing on these types of graphs with flexible definitions of edge scoring functions, message passing and message compression. Using a general framework of neural message passing most graph neural networks defined on multiplex graphs could be described with a single unified theory.

Generalization of ensemble games and the Shapley entropy

The ensemble games and dual ensemble games introduced in Chapter 6 were limited to binary classifier ensembles which weight individual models uniformly. These strong assumptions about the classifier could be relaxed with appropriate consideration. We postulate that the following generalizations are all viable directions of future research:

- *Multi-class generalization:* Using a voting game based design with a one versus many reformulation, ensemble games can be generalized to situations when a data point has a non-binary label. This way the value of models can be quantified in situations where the classification has more than two outcomes.
- *Custom model weighting:* The assumption about uniform model weighting can be relaxed if one assumes that the classification decisions are based on a convex combination of the classification probabilities. This variation of the ensemble game class would require only a slight modification to the proposed definitions and approximation algorithm.
- *Confusion matrix based extension:* The design of dual ensemble games does not consider the nature of false and correct classifications. It can be interesting to ask: Which models have a prime role in the ensemble when it comes to false positive and false negative classification decisions?

Designing and integrating new graph mining tools

The graph mining libraries *Karate Club* [168] and *Little Ball of Fur* [169] presented in Chapters 7 and 8 have been very well received in the graph mining research community. Multiple researchers started to integrate the source code of their own machine learning and graph sampling algorithms

to these frameworks [160]. Research projects which propose novel graph mining algorithms use these libraries [49, 160] for comparing the performance of their own methods to existing well established algorithms. This showcases that creating open-source machine learning frameworks fosters the development of new ideas and facilitates collaborations in the graph mining domain. Designing publicly available libraries which solves theses specific challenges listed would be an important contribution to the field:

- *Learning on multiplex graphs:* Currently there is no publicly available unified framework which provides community detection and node embedding techniques for multiplex, multi-layer and multi-view graph data.
- *Spatio-temporal signal processing:* Extracting knowledge from graph data with temporally changing attributes would require appropriate machine learning and signal processing libraries. Contributions in this domain could cover dynamic node embedding and spatio-temporal parametric models.

Appendix A

Multi-scale attributed node embedding convergence proofs

Lemma 3.1. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps* ***(i) after; or (ii) before*** *node $v \in \mathbb{V}$, as given by:*

$$\underset{l\to\infty}{plim} \frac{\#(v,f)_{\overrightarrow{r}}}{|\mathbb{D}_{\overrightarrow{r}}|} = c^{-1}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f}$$
$$\underset{l\to\infty}{plim} \frac{\#(v,f)_{\overleftarrow{r}}}{|\mathbb{D}_{\overleftarrow{r}}|} = c^{-1}(\boldsymbol{F}^\top\boldsymbol{D}\boldsymbol{P}^r)_{f,v}$$

Proof. The proof is analogous to that given for Theorem 2.1 in [154]. We show that the computed statistics correspond to sequences of random variables with finite expectation, bounded variance and covariances that tend to zero as the separation between variables within the sequence tends to infinity. The Weak Law of Large Numbers (S.N.Bernstein) then guarantees that the sample mean converges to the expectation of the random variable. We first consider the special case $n=1$, i.e. we have a single sequence $v_1, ..., v_l$ generated by a random walk (see Algorithm 2). For a particular node-feature pair (v, f), we let Y_i, $i \in \{1, ..., l-t\}$, be the indicator function for the event $v_i = v$ and $f \in \mathbb{F}_{i+r}$. Thus, we have:

$$\frac{\#(v,f)_{\overrightarrow{r}}}{|\mathbb{D}_{\overrightarrow{r}}|} = \frac{1}{l-t}\sum_{i=1}^{l-t} Y_i, \tag{A.1}$$

the sample average of the Y_is. We also have:

$$\mathbb{E}[Y_i] = \frac{deg(v)}{c}(\boldsymbol{P}^r\boldsymbol{F})_{v,f} = \frac{1}{c}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f}$$

$$\begin{aligned}\mathbb{E}[Y_i Y_j] &= \text{Prob}[v_i = v, f \in \mathbb{F}_{i+r}, v_j = v, f \in \mathbb{F}_{j+r}] \\ &= \underbrace{\frac{deg(v)}{c}}_{p(v_i=v)} \underbrace{\underbrace{\boldsymbol{P}^r_{:v}}_{p(v_{i+r}|v_i=v)} \underbrace{diag(\boldsymbol{F}_{:f})}_{p(f\in\mathbb{F}_{i+r}|v_{i+r})} \underbrace{\boldsymbol{P}^{j-(i+r)}_{:v}}_{p(v_j=v|v_{i+r})}}_{p(v_j=v, f\in\mathbb{F}_{i+r}|v_i=v)} \underbrace{\boldsymbol{P}^r_{v:}\boldsymbol{F}_{:f}}_{p(f\in\mathbb{F}_{j+r}|v_j=v)}\end{aligned}$$

for $j > i + r$. This allows us to compute the covariance:

$$\begin{aligned}\text{Cov}(Y_i, Y_j) &= \mathbb{E}[Y_i Y_j] - \mathbb{E}[Y_i]\,\mathbb{E}[Y_j] \\ &= \tfrac{deg(v)}{c}\boldsymbol{P}^r_{v:} diag(\boldsymbol{F}_{:f}) \underbrace{\left(\boldsymbol{P}^{j-(i+r)}_{:v} - \tfrac{deg(v)}{c}\mathbf{1}\right)}_{\text{tends to 0 as } j-i\to\infty} \boldsymbol{P}^r_{v:}\boldsymbol{F}_{:f},\end{aligned} \tag{A.2}$$

where $\mathbf{1}$ is a vector of ones. The difference term (indicated) tends to zero as $j - i \to \infty$ since then $p(v_j = v|v_{i+r})$ tends to the stationary distribution $p(v) = \frac{deg(v)}{c}$, regardless of v_{i+r}. Thus, applying the Weak Law of Large Numbers, the sample average converges in probability to the expected value, i.e.:

$$\tfrac{\#(v,f)^{\rightarrow}_r}{|\mathbb{D}^{\rightarrow}_r|} = \tfrac{1}{l-t}\sum_{i=1}^{l-t} Y_i \xrightarrow{p} \tfrac{1}{l-t}\sum_{i=1}^{l-t}\mathbb{E}[Y_i] = \tfrac{1}{c}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f}$$

A similar argument applies to $\frac{\#(v,f)^{\leftarrow}_r}{|\mathbb{D}^{\leftarrow}_r|}$, with expectation term $\frac{1}{c}(\boldsymbol{F}^\top\boldsymbol{D}\boldsymbol{P}^r)_{f,v}$. In both cases, the argument readily extends to the general setting where $n > 1$ with suitably defined indicator functions for each of the n random walks (see [154]). □

Lemma 3.2. *The empirical statistics of node-feature pairs obtained from random walks give unbiased estimates of the joint probability of observing feature $f \in \mathbb{F}$ r steps **either side** of node $v \in \mathbb{V}$, given by:*

$$\underset{l\to\infty}{plim}\tfrac{\#(v,f)_r}{|\mathbb{D}_r|} = c^{-1}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f}\ ,$$

Proof.

$$\begin{aligned}\tfrac{\#(v,f)_r}{|\mathbb{D}_r|} &= \tfrac{\#(v,f)^{\rightarrow}_r}{|\mathbb{D}_r|} + \tfrac{\#(v,f)^{\leftarrow}_r}{|\mathbb{D}_r|} \\ &= \tfrac{1}{2}\big(\tfrac{\#(v,f)^{\rightarrow}_r}{|\mathbb{D}^{\rightarrow}_r|} + \tfrac{\#(v,f)^{\leftarrow}_r}{|\mathbb{D}^{\leftarrow}_r|}\big) \\ &\xrightarrow{p} \tfrac{1}{2}\big(\tfrac{1}{c}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f} + \tfrac{1}{c}(\boldsymbol{F}^\top\boldsymbol{D}\boldsymbol{P}^r)_{f,v}\big) \\ &= \tfrac{1}{2c}\big(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F} + \boldsymbol{P}^{r\top}\boldsymbol{D}\boldsymbol{F}\big)_{v,f} \\ &= \tfrac{1}{2c}\big((\boldsymbol{D}\boldsymbol{P}^r + (\boldsymbol{A}^\top\boldsymbol{D}^{-1})^r\boldsymbol{D})\boldsymbol{F}\big)_{v,f} \\ &= \tfrac{1}{2c}\big((\boldsymbol{D}\boldsymbol{P}^r + \boldsymbol{D}(\boldsymbol{D}^{-1}\boldsymbol{A}^\top)^r)\boldsymbol{F}\big)_{v,f} \\ &= \tfrac{1}{c}(\boldsymbol{D}\boldsymbol{P}^r\boldsymbol{F})_{v,f}\ .\end{aligned}$$

The final step follows by symmetry of $\boldsymbol{A}$, indicating how the Lemma can be extended to directed graphs. □

Bibliography

[1] ABADI, MARTÍN AND BARHAM, PAUL AND CHEN, JIANMIN AND CHEN, ZHIFENG AND DAVIS, ANDY AND DEAN, JEFFREY AND DEVIN, MATTHIEU AND GHEMAWAT, SANJAY AND IRVING, GEOFFREY AND ISARD, MICHAEL AND OTHERS. Tensorflow: A System for Large-Scale Machine Learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)* (2016), pp. 265–283.

[2] ABU-EL-HAIJA, S., ALIPOURFARD, N., HARUTYUNYAN, H., KAPOOR, A., AND PEROZZI, B. A Higher-Order Graph Convolutional Layer. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NIPS 2018). NIPS* (2018).

[3] ABU-EL-HAIJA, S., KAPOOR, A., PEROZZI, B., AND LEE, J. N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification. In *Proceedings of Machine Learning Research* (Tel Aviv, Israel, 22–25 Jul 2020), R. P. Adams and V. Gogate, Eds., vol. 115, PMLR, pp. 841–851.

[4] ABU-EL-HAIJA, S., PEROZZI, B., AL-RFOU, R., AND ALEMI, A. A. Watch Your Step: Learning Node Embeddings via Graph Attention. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 9198–9208.

[5] ABU-EL-HAIJA, S., PEROZZI, B., KAPOOR, A., ALIPOURFARD, N., LERMAN, K., HARUTYUNYAN, H., STEEG, G. V., AND GALSTYAN, A. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *International Conference on Machine Learning* (2019).

[6] ADAMIC, L. A., AND ADAR, E. Friends and Neighbors on the Web. *Social Networks 25*, 3 (2003), 211–230.

[7] ADAMIC, L. A., LUKOSE, R. M., PUNIYANI, A. R., AND HUBERMAN, B. A. Search in Power-Law Networks. *Physical Review E 64*, 4 (2001), 046135.

[8] AHMED, A., SHERVASHIDZE, N., NARAYANAMURTHY, S., JOSIFOVSKI, V., AND SMOLA, A. J. Distributed Large-Scale Natural Graph Factorization. In *Proceedings of the 22nd International Conference on World Wide Web* (2013), pp. 37–48.

[9] AHMED, N., ROSSI, R. A., LEE, J., WILLKE, T., ZHOU, R., KONG, X., AND ELDARDIRY, H. Role-based Graph Embeddings. *IEEE Transactions on Knowledge and Data Engineering* (2020), 1–1.

[10] AHMED, N. K., NEVILLE, J., AND KOMPELLA, R. Network Sampling: From Static to Streaming Graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD) 8*, 2 (2013), 1–56.

[11] AHMED, N. K., ROSSI, R., LEE, J. B., KONG, X., WILLKE, T. L., ZHOU, R., AND ELDARDIRY, H. Learning Role-based Graph Embeddings. *Proceedings of the 26th IJCAI Conference on Artificial Intelligence - Statistical Relational AI Workshop* (2018).

[12] ALLEN, C., BALAŽEVIĆ, I., AND HOSPEDALES, T. What the Vec? Towards Probabilistically Grounded Embeddings. In *Advances in Neural Information Processing Systems* (2019).

[13] ANCONA, M., OZTIRELI, C., AND GROSS, M. Explaining Deep Neural Networks with a Polynomial Time Algorithm for Shapley Value Approximation. In *Proceedings of the 36th International Conference on Machine Learning* (2019), vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 272–281.

[14] BACKSTROM, L., HUTTENLOCHER, D., KLEINBERG, J., AND LAN, X. Group Formation in Large Social Networks: Membership, Growth, and Evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2006), ACM, pp. 44–54.

[15] BANDYOPADHYAY, S., KARA, H., KANNAN, A., AND MURTY, M. N. FSCNMF: Fusing Structure and Content via Non-Negative Matrix Factorization for Embedding Information Networks. *arXiv preprint arXiv:1804.05313* (2018).

[16] BARABÁSI, A.-L. Network Science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 371*, 1987 (2013), 20120375.

[17] BELKIN, M., AND NIYOGI, P. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Advances in Neural Information Processing Systems* (2002), pp. 585–591.

[18] BOJCHEVSKI, A., KLICPERA, J., PEROZZI, B., KAPOOR, A., BLAIS, M., RÓZEMBERCZKI, B., LUKASIK, M., AND GÜNNEMANN, S. Scaling Graph Neural Networks with Approximate PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2020), KDD '20, p. 2464–2473.

[19] BROMBERGER, S., AND CONTRIBUTORS, O. Juliagraphs/lightgraphs.jl, September 2017.

[20] BUITINCK, L., LOUPPE, G., BLONDEL, M., PEDREGOSA, F., MUELLER, A., GRISEL, O., NICULAE, V., PRETTENHOFER, P., GRAMFORT, A., GROBLER, J., LAYTON, R., VANDERPLAS, J., JOLY, A., HOLT, B., AND VAROQUAUX, G. API Design for Machine Learning Software: Experiences from the Scikit-Learn Project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning* (2013), pp. 108–122.

[21] CAI, C., AND WANG, Y. A Simple Yet Effective Baseline for Non-Attributed Graph Classification. *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds* (2018).

[22] CAO, S., LU, W., AND XU, Q. GraRep: Learning Graph Representations with Global Structural Information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (2015), ACM, pp. 891–900.

[23] CASTRO, J., GÓMEZ, D., AND TEJADA, J. Polynomial Calculation of the Shapley Value Based on Sampling. *Computers and Operations Research 36*, 5 (2009), 1726 – 1730.

[24] CAVALLARI, S., ZHENG, V. W., CAI, H., CHANG, K. C.-C., AND CAMBRIA, E. Learning Community Embedding with Community Detection and Node Embedding on Graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (2017), pp. 377–386.

[25] CHALKIADAKIS, G., ELKIND, E., AND WOOLDRIDGE, M. Computational Aspects of Cooperative Game Theory. *Synthesis Lectures on Artificial Intelligence and Machine Learning 5*, 6 (2011), 1–168.

[26] CHAMI, I., ABU-EL-HAIJA, S., PEROZZI, B., RÉ, C., AND MURPHY, K. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. *arXiv preprint arXiv:2005.03675* (2020).

[27] CHEN, H., AND KOGA, H. GL2Vec: Graph Embedding Enriched by Line Graphs with Edge Features. In *International Conference on Neural Information Processing* (2019), Springer, pp. 3–14.

[28] CHIANG, W.-L., LIU, X., SI, S., LI, Y., BENGIO, S., AND HSIEH, C.-J. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *International Conference on Knowledge Discovery and Data Mining* (2019).

[29] COHEN, E., DELLING, D., PAJOR, T., AND WERNECK, R. F. Distance-Based Influence in Networks: Computation and Maximization. *Machine Learning on Graphs Workshop* (2014).

[30] CSARDI, G., NEPUSZ, T., ET AL. The igraph Software Package for Complex Network Research. *InterJournal, Complex Systems 1695*, 5 (2006), 1–9.

[31] DA SAN MARTINO, G., NAVARIN, N., AND SPERDUTI, A. Tree-Based Kernel for Graphs with Continuous Attributes. *IEEE Transactions on Neural Networks and Learning Systems* (2017).

[32] DARABOS, D., OLAH, G., GABOR, H., HERSKOVICS, D., NEMETH, A., ERBEN, P., AND BALOGH, Z. LynxKite: The Complete Graph Data Science Platform. `https://github.com/lynxkite/lynxkite`, 2020.

[33] DASGUPTA, A. *Characteristic Functions and Applications*. 2011, pp. 293–322.

[34] DE LARA, N., AND EDOUARD, P. A Simple Baseline Algorithm for Graph Classification. In *Advances in Neural Information Processing Systems* (2018).

[35] DE SA, C., GU, A., RÉ, C., AND SALA, F. Representation Tradeoffs for Hyperbolic Embeddings. *Proceedings of Machine Learning Research 80* (2018), 4460.

[36] DE SOUSA, C. A. R., REZENDE, S. O., AND BATISTA, G. E. Influence of Graph Construction on Semi-Supervised Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2013), Springer, pp. 160–175.

[37] DEFAZIO, A., BACH, F., AND LACOSTE-JULIEN, S. SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives. In *Advances in neural information processing systems* (2014), pp. 1646–1654.

[38] DEFFERRARD, M., BRESSON, X., AND VANDERGHEYNST, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems* (2016), pp. 3111–3119.

[39] DEFFERRARD, M., MARTIN, L., PENA, R., AND PERRAUDIN, N. PyGSP: Graph Signal Processing in Python.

[40] DOERR, C., AND BLENN, N. Metric Convergence in Social Network Sampling. In *Proceedings of the 5th ACM Workshop on HotPlanet* (2013), ACM, pp. 45–50.

[41] DONNAT, C., ZITNIK, M., HALLAC, D., AND LESKOVEC, J. Learning Structural Node Embeddings via Diffusion Wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2018), ACM, pp. 1320–1329.

[42] EASLEY, D., KLEINBERG, J., ET AL. *Networks, Crowds, and Markets*, vol. 8. Cambridge University Press, 2010.

[43] EGGHE, L., AND LEYDESDORFF, L. The Relation Between Pearson's Correlation Coefficient R and Salton's Cosine Measure. *Journal of the American Society for information Science and Technology 60*, 5 (2009), 1027–1036.

[44] EPASTO, A., LATTANZI, S., AND PAES LEME, R. Ego-Splitting Framework: From Non-Overlapping to Overlapping Clusters. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), ACM, pp. 145–154.

[45] FAKCHAROENPHOL, J., RAO, S., AND TALWAR, K. A Tight Bound on Approximating Arbitrary Metrics by Tree Metrics. *Journal of Computer and System Sciences 69*, 3 (2004), 485–497.

[46] FATIMA, S. S., WOOLDRIDGE, M., AND JENNINGS, N. R. A Linear Approximation Method for the Shapley Value. *Artificial Intelligence 172*, 14 (2008), 1673–1699.

[47] FEY, M., AND LENSSEN, J. E. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[48] FINCHAM, E., RÓZEMBERCZKI, B., KOVANOVIĆ, V., JOKSIMOVIĆ, S., JOVANOVIĆ, J., AND GAŠEVIĆ, D. Persistence and performance in co-enrollment network embeddings: An empirical validation of tinto's student integration model. *IEEE Transactions on Learning Technologies 14*, 1 (2021), 106–121.

[49] FREITAS, S., DONG, Y., NEIL, J., AND CHAU, D. H. A Large-Scale Database for Graph Representation Learning, 2020.

[50] GALLAND, A., AND LELARGE, M. Invariant Embedding for Graph Classification. In *ICML Workshop on Learning and Reasoningwith Graph-Structured Data* (2019).

[51] GAO, F., WOLF, G., AND HIRN, M. Geometric Scattering for Graph Data Analysis. In *Proceedings of the 36th International Conference on Machine Learning* (2019), vol. 97, pp. 2122–2131.

[52] GAO, H., AND JI, S. Graph U-nets. In *Proceedings of The 36th International Conference on Machine Learning* (2019).

[53] GÄRTNER, T., FLACH, P., AND WROBEL, S. On Graph Kernels: Hardness Results and Efficient Alternatives. In *Learning theory and kernel machines*. Springer, 2003, pp. 129–143.

[54] GASHLER, M., GIRAUD-CARRIER, C., AND MARTINEZ, T. Decision Tree Ensemble: Small Heterogeneous is Better than Large Homogeneous. In *Seventh International Conference on Machine Learning and Applications* (2008), IEEE, pp. 900–905.

[55] GHORBANI, A., AND ZOU, J. Data Shapley: Equitable Valuation of Data for Machine Learning. In *International Conference on Machine Learning* (2019), pp. 2242–2251.

[56] GHORBANI, A., AND ZOU, J. Neuron Shapley: Discovering the Responsible Neurons. *arXiv preprint arXiv:2002.09815* (2020).

[57] GHOSH, A., ROZEMBERCZKI, B., RAMAMOORTHY, S., AND SARKAR, R. Topological signatures for fast mobility analysis. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2018), pp. 159–168.

[58] GILMER, J., SCHOENHOLZ, S. S., RILEY, P. F., VINYALS, O., AND DAHL, G. E. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1263–1272.

[59] GJOKA, M., BUTTS, C. T., KURANT, M., AND MARKOPOULOU, A. Multigraph Sampling of Online Social Networks. *IEEE Journal on Selected Areas in Communications 29*, 9 (2011), 1893–1905.

[60] GJOKA, M., KURANT, M., BUTTS, C. T., AND MARKOPOULOU, A. Walking in Facebook: A Case Study of Unbiased Sampling of Online Social Networks. In *Proceedings of the 2010 IEEE INFOCOM Conference on Computer Communications* (2010), IEEE, pp. 1–9.

[61] GONZALEZ, J. E., LOW, Y., GU, H., BICKSON, D., AND GUESTRIN, C. Powergraph: Distributed Graph-Parallel Computation on Natural Graphs. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)* (2012), pp. 17–30.

[62] GOODMAN, L. A. Snowball Sampling. *The Annals of Mathematical Statistics* (1961), 148–170.

[63] GOYAL, P., AND FERRARA, E. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems 151* (2018), 78–94.

[64] GRATTAROLA, D., AND ALIPPI, C. Graph Neural Networks in TensorFlow and Keras with Spektral. *IEEE Computational Intelligence Magazine 16*, 1 (2021), 99–106.

[65] GREGORY, S. Finding Overlapping Communities in Networks by Label Propagation. *New Journal of Physics 12*, 10 (2010), 103018.

[66] GROVER, A., AND LESKOVEC, J. Node2Vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), pp. 855–864.

[67] GUTMANN, M., AND HYVARINEN, A. Noise-Contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (2010), pp. 297–304.

[68] HAGBERG, A., SWART, P., AND S CHULT, D. Exploring Network Structure, Dynamics, and Function Using NetworkX. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[69] HALCROW, J., MOSOI, A., RUTH, S., AND PEROZZI, B. Grale: Designing Networks for Graph Learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2020), KDD '20, p. 2523–2532.

[70] HALKO, N., MARTINSSON, P. G., AND TROPP, J. A. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review 53*, 2 (2011), 217–288.

[71] HAMILTON, W., YING, Z., AND LESKOVEC, J. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems* (2017), pp. 1024–1034.

[72] HAMILTON, W. L., YING, R., AND LESKOVEC, J. Representation Learning on Graphs: Methods and Applications. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* (2017).

[73] HAN, X., CAO, S., XIN, L., LIN, Y., LIU, Z., SUN, M., AND LI, J. OpenKE: An Open Toolkit for Knowledge Embedding. In *Proceedings of EMNLP* (2018).

[74] HAUSSLER, D. Convolution Kernels on Discrete Structures. Tech. rep., Technical Report, Department of Computer Science, University of California at Santa Cruz, 1999.

[75] HENDERSON, K., GALLAGHER, B., ELIASSI-RAD, T., TONG, H., BASU, S., AKOGLU, L., KOUTRA, D., FALOUTSOS, C., AND LI, L. RolX: Structural Role Extraction and Mining in Large Graphs. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2012), ACM, pp. 1231–1239.

[76] HENDERSON, K., GALLAGHER, B., LI, L., AKOGLU, L., ELIASSI-RAD, T., TONG, H., AND FALOUTSOS, C. It's Who You Know: Graph Mining Using Recursive Structural Features. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data mining* (2011), ACM, pp. 663–671.

[77] HOEFFDING, W. Probability Inequalities for Sums of Bounded Random Variables. In *The Collected Works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[78] HORVÁTH, T., GÄRTNER, T., AND WROBEL, S. Cyclic Pattern Kernels for Predictive Graph Mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), pp. 158–167.

[79] HU, P., AND LAU, W. C. A Survey and Taxonomy of Graph Sampling. *arXiv preprint arXiv:1308.5865* (2013).

[80] HUANG, K., NI, C.-C., SARKAR, R., GAO, J., AND MITCHELL, J. S. Bounded Stretch Geographic Homotopic Routing in Sensor Networks. In *Proceedings of the 2014 IEEE INFOCOM Conference on Computer Communications* (2014), IEEE, pp. 979–987.

[81] HUANG, X., LI, J., AND HU, X. Accelerated Attributed Network Embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining* (2017), SIAM, pp. 633–641.

[82] HÜBLER, C., KRIEGEL, H.-P., BORGWARDT, K., AND GHAHRAMANI, Z. Metropolis Algorithms for Representative Subgraph Sampling. In *Proceedings of the 8th IEEE International Conference on Data Mining* (2008), IEEE, pp. 283–292.

[83] JIA, L., GAÜZÈRE, B., AND HONEINE, P. Graph Kernels Based on Linear Patterns: Theoretical and Experimental Comparisons. *Pattern Recognition Letters* (2019).

[84] JIA, R., DAO, D., WANG, B., HUBIS, F. A., HYNES, N., GÜREL, N. M., LI, B., ZHANG, C., SONG, D., AND SPANOS, C. J. Towards Efficient Data Valuation Based on the Shapley Value. In *Proceedings of Machine Learning Research* (2019), pp. 1167–1176.

[85] JUNDONG LI, LIANG WU, H. L. Multi-Level Network Embedding with Boosted Low-Rank Matrix Approximation. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019* (2019), ACM, pp. 50–56.

[86] KANG, U., AND FALOUTSOS, C. Big Graph Mining: Algorithms and Discoveries. *ACM SIGKDD Explorations Newsletter 14*, 2 (2013), 29–36.

[87] KANG, U., TSOURAKAKIS, C. E., AND FALOUTSOS, C. Pegasus: A Peta-Scale Graph Mining System Implementation and Observations. In *Proceedings of the 9th IEEE international conference on data mining* (2009), IEEE, pp. 229–238.

[88] KARYPIS, G., AND KUMAR, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing 20*, 1 (1998), 359–392.

[89] KASHIMA, H., TSUDA, K., AND INOKUCHI, A. Marginalized Kernels between Labeled Graphs. In *Proceedings of the 20th International Conference on Machine Learning* (2003), pp. 321–328.

[90] KHAN, M. R., AND BLUMENSTOCK, J. E. Multi-GCN: Graph Convolutional Networks for Multi-View Networks, with Applications to Global Poverty. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 606–613.

[91] KINGMA, D., AND BA, J. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations* (2015).

[92] KIPF, T. N., AND WELLING, M. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations* (2017).

[93] KLICPERA, J., BOJCHEVSKI, A., AND GÜNNEMANN, S. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations* (2019).

[94] KLICPERA, J., WEISSENBERGER, S., AND GÜNNEMANN, S. Diffusion Improves Graph Learning. In *Advances in Neural Information Processing Systems* (2019), pp. 13354–13366.

[95] KRIEGE, N., AND MUTZEL, P. Subgraph Matching Kernels for Attributed Graphs. In *Proceedings of the 29th International Coference on International Conference on Machine Learning* (2012), p. 291–298.

[96] KRIEGE, N. M., GISCARD, P.-L., AND WILSON, R. On Valid Optimal Assignment Kernels and Applications to Graph Classification. In *Advances in Neural Information Processing Systems* (2016), pp. 1623–1631.

[97] KRIEGE, N. M., JOHANSSON, F. D., AND MORRIS, C. A Survey on Graph Kernels. *Applied Network Science 5*, 1 (2020), 1–42.

[98] KRISHNAMURTHY, V., FALOUTSOS, M., CHROBAK, M., LAO, L., CUI, J.-H., AND PERCUS, A. G. Reducing Large Internet Topologies for Faster Simulations. In *International Conference on Research in Networking* (2005), Springer, pp. 328–341.

[99] KUANG, D., DING, C., AND PARK, H. Symmetric Nonnegative Matrix Factorization for Graph Clustering. In *Proceedings of the 2012 SIAM International Conference on Data Mining* (2012), SIAM, pp. 106–117.

[100] LAZAREVIC, A., AND OBRADOVIC, Z. Effective Pruning of Neural Network Classifier Ensembles. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)* (2001), vol. 2, IEEE, pp. 796–801.

[101] LEE, C.-H., XU, X., AND EUN, D. Y. Beyond Random Walk and Metropolis-Hastings Samplers: Why You Should Not Backtrack for Unbiased Graph Sampling. *ACM SIGMETRICS Performance Evaluation Review 40*, 1 (2012), 319–330.

[102] LEE, J., LEE, I., AND KANG, J. Self-Attention Graph Pooling. In *Proceedings of the 36th International Conference on Machine Learning* (09–15 Jun 2019).

[103] LEECH, D. Computing Power Indices for Large Voting Games: A New Algorithm. Tech. rep., University of Warwick, 1998.

[104] LEECH, D. Computing Power Indices for Large Voting Games. *Management Science 49*, 6 (2003), 831–837.

[105] LESKOVEC, J., AND FALOUTSOS, C. Sampling From Large Graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2006), pp. 631–636.

[106] LESKOVEC, J., KLEINBERG, J., AND FALOUTSOS, C. Graphs Over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2005), ACM, pp. 177–187.

[107] LESKOVEC, J., AND KREVL, A. SNAP Datasets: Stanford Large Network Dataset Collection. `http://snap.stanford.edu/data`, June 2014.

[108] LESKOVEC, J., RAJARAMAN, A., AND ULLMAN, J. D. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[109] LESKOVEC, J., AND SOSIČ, R. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology 8*, 1 (2016), 1.

[110] LEVY, O., AND GOLDBERG, Y. Neural Word Embedding as Implicit Matrix Factorization. In *Advances in Neural Information Processing Systems* (2014), pp. 2177–2185.

[111] LI, J.-Y., AND YEH, M.-Y. On Sampling Type Distribution From Heterogeneous Social Networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2011), Springer, pp. 111–122.

[112] LI, P.-Z., HUANG, L., WANG, C.-D., AND LAI, J.-H. EdMot: An Edge Enhancement Approach for Motif-aware Community Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019), ACM, pp. 479–487.

[113] LI, R.-H., YU, J. X., QIN, L., MAO, R., AND JIN, T. On Random Walk Based Graph Sampling. In *Proceedings of the 31st IEEE International Conference on Data Engineering* (2015), IEEE, pp. 927–938.

[114] LI, Y., WU, Z., LIN, S., XIE, H., LV, M., XU, Y., AND LUI, J. C. Walking with Perception: Efficient Random Walk Sampling via Common Neighbor Awareness. In *Proceedings of the 35th IEEE International Conference on Data Engineering* (2019), IEEE, pp. 962–973.

[115] LI, Y., YU, R., SHAHABI, C., AND LIU, Y. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations* (2018).

[116] LIAO, L., HE, X., ZHANG, H., AND CHUA, T.-S. Attributed Social Network Embedding. *IEEE Transactions on Knowledge and Data Engineering 30*, 12 (2018), 2257–2270.

[117] LIPOVETSKY, S., AND CONKLIN, M. Analysis of Regression in Game Theory Approach. *Applied Stochastic Models in Business and Industry 17*, 4 (2001), 319–330.

[118] LU, Q., AND GETOOR, L. Link-based Classification. In *International Conference on Machine Learning* (2003).

[119] LUNDBERG, S. M., AND LEE, S.-I. A Unified Approach to Interpreting Model Predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), Curran Associates Inc., p. 4768–4777.

[120] MA, X., LI, Z., XU, L., SONG, G., LI, Y., AND SHI, C. Learning Discrete Adaptive Receptive Fields for Graph Convolutional Networks.

[121] MAIYA, A. S., AND BERGER-WOLF, T. Y. Sampling Community Structure. In *Proceedings of the 19th International Conference on World Wide Web* (2010), pp. 701–710.

[122] MALEKI, S., TRAN-THANH, L., HINES, G., RAHWAN, T., AND ROGERS, A. Bounding the Estimation Error of Sampling-Based Shapley Value Approximation. *arXiv preprint arXiv:1306.4265* (2013).

[123] MANN, I., AND SHAPLEY, L. S. Values of Large Games, IV: Evaluating the Electoral College by Monte Carlo Techniques. Tech. rep., Rand Corporation, 1960.

[124] MARA, A. C. EvalNE : A Framework for Evaluating Network Embeddings on Link Prediction. In *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019), OpenReview.

[125] MARTÍNEZ-MUÑOZ, G., AND SUÁREZ, A. Pruning in Ordered Bagging Ensembles. In *Proceedings of the 23rd International Conference on Machine Learning* (2006), pp. 609–616.

[126] MATOUŠEK, J. *Lectures on Discrete Geometry*, vol. 108. Springer, 2002.

[127] MATSUNO, R., AND MURATA, T. MELL: Effective Embedding Method for Multiplex Networks. In *Companion Proceedings of the The Web Conference 2018* (2018), International World Wide Web Conferences Steering Committee, pp. 1261–1268.

[128] MENICHETTI, G., REMONDINI, D., PANZARASA, P., MONDRAGÓN, R. J., AND BIANCONI, G. Weighted Multiplex Networks. *PloS one 9*, 6 (2014).

[129] MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of the 1st International Conference on Learning Representations* (2013).

[130] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed Representations of Words and Phrases and Their Compositionality. In *Advances in Neural Information Processing Systems* (2013), pp. 3111–3119.

[131] MOKDAD, F., BOUCHAFFRA, D., ZERROUKI, N., AND TOUAZI, A. Determination of an Optimal Feature Selection Method Based on Maximum Shapley Value. In *2015 15th International Conference on Intelligent Systems Design and Applications (ISDA)* (2015), IEEE, pp. 116–121.

[132] NAIR, V., AND HINTON, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning* (2010), pp. 807–814.

[133] NAMATA, G., LONDON, B., GETOOR, L., HUANG, B., AND EDU, U. Query-Driven Active Surveying for Collective Classification. In *International Workshop on Mining and Learning with Graphs* (2012).

[134] NARAYANAN, A., CHANDRAMOHAN, M., VENKATESAN, R., CHEN, L., AND LIU, Y. Graph2Vec: Learning Distributed Representations of Graphs. In *Machine Learning on Graphs Workshop (KDD)* (2017).

[135] NEUMANN, L. J., MORGENSTERN, O., ET AL. *Theory of Games and Economic Behavior*, vol. 60. Princeton University Press Princeton, 1947.

[136] ONNELA, J.-P., SARAMÄKI, J., HYVÖNEN, J., SZABÓ, G., LAZER, D., KASKI, K., KERTÉSZ, J., AND BARABÁSI, A.-L. Structure and Tie Strengths in Mobile Communication Networks. *Proceedings of the National Academy of Sciences 104*, 18 (2007), 7332–7336.

[137] OPSAHL, T., AGNEESSENS, F., AND SKVORETZ, J. Node Centrality in Weighted Networks: Generalizing Degree and Shortest Paths. *Social networks 32*, 3 (2010), 245–251.

[138] OSBORNE, M. J., AND RUBINSTEIN, A. *A Course in Game Theory*. MIT press, 1994.

[139] OU, M., CUI, P., PEI, J., ZHANG, Z., AND ZHU, W. Asymmetric Transitivity Preserving Graph Embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), ACM, pp. 1105–1114.

[140] OWEN, G. Multilinear Extensions of Games. *Management Science 18*, 5-part-2 (1972), 64–79.

[141] PAPADOPOULOS, S., KOMPATSIARIS, Y., VAKALI, A., AND SPYRIDONOS, P. Community Detection in Social Media. *Data Mining and Knowledge Discovery 24*, 3 (2012), 515–554.

[142] PARK, C., KIM, D., HAN, J., AND YU, H. Unsupervised Attributed Multiplex Network Embedding. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence, AAAI 2020*, AAAI Press, pp. 5371–5378.

[143] PASCAL, P., AND LATAPY, M. *In International Symposium on Computer and Information Sciences*. Springer Berlin Heidelberg, 2005, ch. Computing Communities in Large Networks Using Random Walks, pp. 284–293.

[144] PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., ET AL. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems* (2019), pp. 8024–8035.

[145] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., ET AL. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research 12*, Oct (2011), 2825–2830.

[146] PEIXOTO, T. P. The Graph-Tool Python Library. *figshare* (2014).

[147] PEROZZI, B., AL-RFOU, R., AND SKIENA, S. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* (2014), ACM.

[148] PEROZZI, B., KULKARNI, V., CHEN, H., AND SKIENA, S. Don't Walk, Skip!: Online Learning of Multi-scale Network Embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017* (2017), ACM, pp. 258–265.

[149] PEROZZI, B., AND SKIENA, S. Exact Age Prediction in Social Networks. In *Proceedings of the 24th International Conference on World Wide Web* (2015), pp. 91–92.

[150] PIMENTEL, T., VELOSO, A., AND ZIVIANI, N. Unsupervised and Scalable Algorithm for Learning Node Representations. *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2017).

[151] PINTÉR, M. Regression Games. *Annals of Operations Research 186*, 1 (2011), 263–274.

[152] POSTĂVARU, Ş., TSITSULIN, A., DE ALMEIDA, F. M. G., TIAN, Y., LATTANZI, S., AND PEROZZI, B. InstantEmbedding: Efficient Local Node Representations. *arXiv preprint arXiv:2010.06992* (2020).

[153] PRAT-PÉREZ, A., DOMINGUEZ-SAL, D., AND LARRIBA-PEY, J.-L. High Quality, Scalable and Parallel Community Detection for Large Real Graphs. In *Proceedings of the 23rd International Conference on World Wide Web* (2014), pp. 225–236.

[154] QIU, J., DONG, Y., MA, H., LI, J., WANG, K., AND TANG, J. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2Vec. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining* (2018), ACM, pp. 459–467.

[155] RAGHAVAN, U. N., ALBERT, R., AND KUMARA, S. Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks. *Physical Review E 76*, 3 (2007), 036106.

[156] REHUREK, R., AND SOJKA, P. Gensim — Statistical Semantics in Python. *Retrieved from genism. org* (2011).

[157] REZVANIAN, A., AND MEYBODI, M. R. Sampling Social Networks Using Shortest Paths. *Physica A: Statistical Mechanics and its Applications 424* (2015), 254–268.

[158] RIBEIRO, B., AND TOWSLEY, D. Estimating and Sampling Graphs with Multidimensional Random Walks. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (2010), ACM, pp. 390–403.

[159] RIBEIRO, L. F., SAVERESE, P. H., AND FIGUEIREDO, D. R. Struc2Vec: Learning Node Representations from Structural Identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), ACM, pp. 385–394.

[160] RICAUD, B., ASPERT, N., AND MIZ, V. Spikyball Sampling: Exploring Large Networks via an Inhomogeneous Filtered Diffusion. *Algorithms 13*, 11 (2020), 275.

[161] RÍOS, S. A., AND VIDELA-CAVIERES, I. F. Generating Groups of Products Using Graph Mining Techniques. *Procedia Computer Science 35* (2014), 730–738.

[162] ROSSETTI, G. DyNetx: Dynamic Network Analysis Library. `https://github.com/GiulioRossetti/dynetx`, 2020.

[163] ROSSETTI, G., MILLI, L., AND CAZABET, R. CDLIB: A Python Library to Extract, Compare and Evaluate Communities from Complex Networks. *Applied Network Science 4*, 1 (2019), 52.

[164] ROSSETTI, G., MILLI, L., RINZIVILLO, S., SÎRBU, A., PEDRESCHI, D., AND GIANNOTTI, F. NDlib: A Python Library to Model and Analyze Diffusion Processes Over Complex Networks. *International Journal of Data Science and Analytics 5*, 1 (2018), 61–79.

[165] ROZEMBERCZKI, B., ALLEN, C., AND SARKAR, R. Multi-scale attributed node embedding. *Journal of Complex Networks 9*, 2 (2021), cnab014.

[166] ROZEMBERCZKI, B., DAVIES, R., SARKAR, R., AND SUTTON, C. GEMSEC: Graph Embedding with Self Clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019* (2019), ACM, pp. 65–72.

[167] ROZEMBERCZKI, B., ENGLERT, P., KAPOOR, A., BLAIS, M., AND PEROZZI, B. Pathfinder Discovery Networks for Neural Message Passing. In *Proceedings of the 2021 World Wide Web Conference* (2021).

[168] ROZEMBERCZKI, B., KISS, O., AND SARKAR, R. Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)* (2020), ACM.

[169] ROZEMBERCZKI, B., KISS, O., AND SARKAR, R. Little Ball of Fur: A Python Library for Graph Sampling. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)* (2020), ACM.

[170] ROZEMBERCZKI, B., AND SARKAR, R. Fast Sequence-Based Embedding with Diffusion Graphs. In *International Workshop on Complex Networks* (2018), Springer, pp. 99–107.

[171] ROZEMBERCZKI, B., AND SARKAR, R. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)* (2020), ACM.

[172] ROZEMBERCZKI, B., AND SARKAR, R. The shapley value of classifiers in ensemble games. *arXiv preprint arXiv:2101.02153* (2021).

[173] ROZEMBERCZKI, B., AND SARKAR, R. Twitch Gamers: a Dataset for Evaluating Proximity Preserving and Structural Role-based Node Embeddings, 2021.

[174] ROZEMBERCZKI, B., SCHERER, P., HE, Y., PANAGOPOULOS, G., RIEDEL, A., ASTEFANOAEI, M., KISS, O., BERES, F., LOPEZ, G., COLLIGNON, N., AND SARKAR, R. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models, 2021.

[175] ROZEMBERCZKI, B., SCHERER, P., KISS, O., SARKAR, R., AND FERENCI, T. Chickenpox cases in hungary: a benchmark dataset for spatiotemporal signal processing with graph neural networks. *arXiv preprint arXiv:2102.08100* (2021).

[176] SARKAR, R. Low Distortion Delaunay Embedding of Trees in Hyperbolic Plane. In *International Symposium on Graph Drawing* (2011), Springer, pp. 355–366.

[177] SCHIAVINATO, M., GASPARETTO, A., AND TORSELLO, A. Transitive Assignment Kernels for Structural Classification. In *International Workshop on Similarity-Based Pattern Recognition* (2015), Springer, pp. 146–159.

[178] SCHVANEVELDT, R. W., DURSO, F. T., AND DEARHOLT, D. W. Network Structures in Proximity Data. In *Psychology of Learning and Motivation*, vol. 24. Elsevier, 1989, pp. 249–284.

[179] SHALEV-SHWARTZ, S., AND BEN-DAVID, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[180] SHAPLEY, L. S. A Value for n-Person Games. *Contributions to the Theory of Games 2*, 28 (1953), 307–317.

[181] SHCHUR, O., MUMME, M., BOJCHEVSKI, A., AND GÜNNEMANN, S. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS 2018* (2018).

[182] SHERVASHIDZE, N., SCHWEITZER, P., VAN LEEUWEN, E. J., MEHLHORN, K., AND BORGWARDT, K. M. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research 12*, 77 (2011), 2539–2561.

[183] SHI, Y., HAN, F., HE, X., HE, X., YANG, C., LUO, J., AND HAN, J. MVN2Vec: Preservation and Collaboration in Multi-view Network Embedding. *arXiv preprint arXiv:1801.06597* (2018).

[184] SIGLIDIS, G., NIKOLENTZOS, G., LIMNIOS, S., GIATSIDIS, C., SKIANIS, K., AND VAZIRGIANNIS, M. GraKeL: A Graph Kernel Library in Python. *Journal of Machine Learning Research 21*, 54 (2020), 1–5.

[185] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[186] STAUDT, C. L., SAZONOVS, A., AND MEYERHENKE, H. NetworKit: A Tool Suite for Large-Scale Complex Network Analysis. *Network Science 4*, 4 (2016), 508–530.

[187] STIER, J., GIANINI, G., GRANITZER, M., AND ZIEGLER, K. Analysing Neural Network Topologies: A Game Theoretic Approach. *Procedia Computer Science 126* (2018), 234–243.

[188] STRUMBELJ, E., AND KONONENKO, I. An Efficient Explanation of Individual Classifications Using Game Theory. *The Journal of Machine Learning Research 11* (2010), 1–18.

[189] STUMPF, M. P., WIUF, C., AND MAY, R. M. Subnets of Scale-Free Networks are not Scale-Free: Sampling Properties of Networks. *Proceedings of the National Academy of Sciences 102*, 12 (2005), 4221–4224.

[190] STUTZBACH, D., REJAIE, R., DUFFIELD, N., SEN, S., AND WILLINGER, W. On Unbiased Sampling for Unstructured Peer-to-Peer Networks. *IEEE/ACM Transactions on Networking 17*, 2 (2008), 377–390.

[191] SUGIYAMA, M., AND BORGWARDT, K. Halting in Random Walk Kernels. In *Advances in Neural Information Processing Systems* (2015), pp. 1639–1647.

[192] SUGIYAMA, M., GHISU, M. E., LLINARES-LÓPEZ, F., AND BORGWARDT, K. Graphkernels: R and Python Packages for Graph Comparison. *Bioinformatics 34*, 3 (2017), 530–532.

[193] SUN, B.-J., SHEN, H., GAO, J., OUYANG, W., AND CHENG, X. A Non-Negative Symmetric Encoder-Decoder Approach for Community Detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (2017), ACM, pp. 597–606.

[194] SUN, D. L., AND FEVOTTE, C. Alternating Direction Method of Multipliers for Non-Negative Matrix Factorization with the Beta-Divergence. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing* (2014), IEEE, pp. 6201–6205.

[195] SUN, X., LIU, Y., LI, J., ZHU, J., LIU, X., AND CHEN, H. Using Cooperative Game Theory to Optimize the Feature Selection Problem. *Neurocomputing 97* (2012), 86–93.

[196] SUNDARARAJAN, M., AND NAJMI, A. The Many Shapley Values for Model Explanation. In *International Conference on Machine Learning* (2020), PMLR, pp. 9269–9278.

[197] TAKIMOTO, E., AND WARMUTH, M. K. Path Kernels and Multiplicative Updates. *Journal of Machine Learning Research 4*, Oct (2003), 773–818.

[198] TANG, J., QU, M., WANG, M., ZHANG, M., YAN, J., AND MEI, Q. LINE: Large-Scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web* (2015), pp. 1067–1077.

[199] TANG, L., AND LIU, H. Relational Learning via Latent Social Dimensions. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data mining* (2009), pp. 817–826.

[200] TANG, L., AND LIU, H. Scalable Learning of Collective Behavior Based on Sparse Social Dimensions. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management* (2009), pp. 1107–1116.

[201] TSENG, V. S., YING, J.-C., HUANG, C.-W., KAO, Y., AND CHEN, K.-T. Fraudetector: A Graph Mining Based Framework for Fraudulent Phone Call Detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), pp. 2157–2166.

[202] TSITSULIN, A., MOTTIN, D., KARRAS, P., BRONSTEIN, A., AND MÜLLER, E. NetLSD: Hearing the Shape of a Graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2018), ACM, pp. 2347–2356.

[203] TSITSULIN, A., MOTTIN, D., KARRAS, P., AND MÜLLER, E. VERSE: Versatile Graph Embeddings from Similarity Measures. In *Proceedings of the 2018 World Wide Web Conference* (2018), pp. 539–548.

[204] TSITSULIN, A., MUNKHOEVA, M., AND PEROZZI, B. Just SLaQ When You Approximate: Accurate Spectral Distances for Web-Scale Graphs. In *Proceedings of The Web Conference 2020* (2020), WWW '20, p. 2697–2703.

[205] TU, C., YAO, Y., ZHANG, Z., CUI, G., WANG, H., TIAN, C., ZHOU, J., AND YANG, C. OpenNE: An Open Source Toolkit for Network Embedding. `https://github.com/thunlp/OpenNE`, 2018.

[206] TUMER, K., AND GHOSH, J. Error Correlation and Error Reduction in Ensemble Classifiers. *Connection Science 8*, 3-4 (1996), 385–404.

[207] VAN DER MAATEN, L. Accelerating t-SNE Using Tree-Based Algorithms. *Journal of Machine Learning Research 15*, 93 (2014), 3221–3245.

[208] VAN DER MAATEN, L., AND HINTON, G. Visualizing Data Using t-SNE . *Journal of Machine Learning Research 9* (2008), 2579–2605.

[209] VAN LAARHOVEN, T., AND MARCHIORI, E. Robust Community Detection Methods with Resolution Parameter for Complex Detection in Protein Protein Interaction Networks. *Pattern Recognition in Bioinformatics* (2012), 1–13.

[210] VELIČKOVIĆ, P., CUCURULL, G., CASANOVA, A., ROMERO, A., LIÒ, P., AND BENGIO, Y. Graph Attention Networks. In *International Conference on Learning Representations* (2018).

[211] VELIČKOVIĆ, P., FEDUS, W., HAMILTON, W. L., LIÒ, P., BENGIO, Y., AND HJELM, R. D. Deep Graph Infomax. In *International Conference on Learning Representations* (2019).

[212] VERBEEK, K., AND SURI, S. Metric Embedding, Hyperbolic Space, and Social Networks. In *Proceedings of the 30th ACM Annual Symposium on Computational Geometry* (2014), ACM, pp. 501–510.

[213] VERMA, S., AND ZHANG, Z.-L. Hunt for the Unique, Stable, Sparse and Fast Feature Learning on Graphs. In *Advances in Neural Information Processing Systems* (2017), pp. 88–98.

[214] VIRTANEN, P., GOMMERS, R., OLIPHANT, T. E., HABERLAND, M., REDDY, T., COURNAPEAU, D., BUROVSKI, E., PETERSON, P., WECKESSER, W., BRIGHT, J., VAN DER WALT, S. J., BRETT, M., WILSON, J., JARROD MILLMAN, K., MAYOROV, N., NELSON, A. R. J., JONES, E., KERN, R., LARSON, E., CAREY, C., POLAT, İ., FENG, Y., MOORE, E. W., VAND ERPLAS, J., LAXALDE, D., PERKTOLD, J., CIMRMAN, R., HENRIKSEN, I., QUINTERO, E. A., HARRIS, C. R., ARCHIBALD, A. M., RIBEIRO, A. H., PEDREGOSA, F., VAN MULBREGT, P., AND CONTRIBUTORS, S. . . SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272.

[215] WALT, S. V. D., COLBERT, S. C., AND VAROQUAUX, G. The NumPy Array: a Structure for Efficient Numerical Computation. *Computing in Science & Engineering 13*, 2 (2011), 22–30.

[216] WANG, G., YING, R., HUANG, J., AND LESKOVEC, J. Improving Graph Attention Networks with Large Margin-based Constraints. *NeurIPS Workshop on Graph Representation Learning* (2019).

[217] WANG, M., YU, L., ZHENG, D., GAN, Q., GAI, Y., YE, Z., LI, M., ZHOU, J., HUANG, Q., MA, C., HUANG, Z., GUO, Q., ZHANG, H., LIN, H., ZHAO, J., LI, J., SMOLA, A. J., AND ZHANG, Z. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019).

[218] WANG, X., CUI, P., WANG, J., PEI, J., ZHU, W., AND YANG, S. Community Preserving Network Embedding. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence* (2017), AAAI Press, pp. 203–209.

[219] WATSON, L., ROZEMBERCZKI, B., AND SARKAR, R. Stability enhanced privacy and applications in private stochastic gradient descent. *arXiv preprint arXiv:2006.14360* (2020).

[220] WATTS, D. J., AND STROGATZ, S. H. Collective Dynamics of 'Small-World' Networks. *Nature 393*, 6684 (1998), 440–442.

[221] WILSON, D. B. Generating Random Spanning Trees More Quickly Than the Cover Time. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing* (1996), ACM, pp. 296–303.

[222] WU, F., SOUZA, A., ZHANG, T., FIFTY, C., YU, T., AND WEINBERGER, K. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning* (2019), pp. 6861–6871.

[223] WU, X., ZHAO, L., AND AKOGLU, L. A Quest for Structure: Jointly Learning the Graph Structure and Semi-Supervised Classification, 2019.

[224] XU, B., SHEN, H., CAO, Q., QIU, Y., AND CHENG, X. Graph Wavelet Neural Network. In *International Conference on Learning Representations (ICLR)* (2019).

[225] YANARDAG, P., AND VISHWANATHAN, S. A Structural Smoothing Framework For Robust Graph Comparison. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2134–2142.

[226] YANARDAG, P., AND VISHWANATHAN, S. Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2015), p. 1365–1374.

[227] YANG, C., LIU, Z., ZHAO, D., SUN, M., AND CHANG, E. Network Representation Learning with Text Information. In *Proceedings of the 24th IJCAI Conference on Artificial Intelligence* (2015).

[228] YANG, C., SUN, M., LIU, Z., AND TU, C. Fast Network Embedding Enhancement via High Order Proximity Approximation. In *Proceedings of the 26th IJCAI Conference on Artificial Intelligence* (2017), pp. 3894–3900.

[229] YANG, C.-L., KUNG, P.-H., CHEN, C.-A., AND LIN, S.-D. Semantically Sampling in Heterogeneous Social Networks. In *Proceedings of the 22nd International Conference on World Wide Web* (2013), pp. 181–182.

[230] YANG, D., ROSSO, P., LI, B., AND CUDRE-MAUROUX, P. NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019), ACM, pp. 1162–1172.

[231] YANG, H., PAN, S., ZHANG, P., CHEN, L., LIAN, D., AND ZHANG, C. Binarized Attributed Network Embedding. In *Proceedings of the 18th IEEE International Conference on Data Mining* (2018), IEEE, pp. 1476–1481.

[232] YANG, J., AND LESKOVEC, J. Overlapping Community Detection at Scale: a Nonnegative Matrix Factorization Approach. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining* (2013), ACM, pp. 587–596.

[233] YANG, S., REN, Y., BAO, C., ZHUO, Y., HU, C., YE, T., AND ZIHANG, Y. Euler: A Distributed Graph Deep Learning Framework. `https://github.com/alibaba/euler`, 2019.

[234] YANG, S., AND YANG, B. Enhanced Network Embedding with Text Information. In *Proceedings of the 24th IEEE International Conference on Pattern Recognition* (2018), IEEE, pp. 326–331.

[235] YANG, S. Y., LIU, F.-C., ZHU, X., AND YEN, D. C. A Graph Mining Approach to Identify Financial Reporting Patterns: An Empirical Examination of Industry Classifications. *Decision Sciences 50*, 4 (2019), 847–876.

[236] YANG, Z., COHEN, W. W., AND SALAKHUTDINOV, R. Revisiting Semi-supervised Learning with Graph Embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning* (2016), pp. 40–48.

[237] YAO, L., MAO, C., AND LUO, Y. Graph Convolutional Networks for Text Classification. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 7370–7377.

[238] YE, F., CHEN, C., AND ZHENG, Z. Deep Autoencoder-Like Nonnegative Matrix Factorization for Community Detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018), ACM, pp. 1393–1402.

[239] YING, R., HE, R., CHEN, K., EKSOMBATCHAI, P., HAMILTON, W. L., AND LESKOVEC, J. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2018), ACM, pp. 974–983.

[240] YING, Z., YOU, J., MORRIS, C., REN, X., HAMILTON, W., AND LESKOVEC, J. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems* (2018), pp. 4800–4810.

[241] YU, X., BAN, X., ZENG, W., SARKAR, R., GU, X., AND GAO, J. Spherical Representation and Polyhedron Routing for Load Balancing in Wireless Sensor Networks. In *Proceedings of the 2011 IEEE INFOCOM Conference on Computer Communications* (2011), IEEE, pp. 621–625.

[242] ZACHARY, W. W. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research 33*, 4 (1977), 452–473.

[243] ZENG, W., SARKAR, R., LUO, F., GU, X., AND GAO, J. Resilient Routing for Sensor Networks Using Hyperbolic Embedding of Universal Covering Space. In *Proceedings of the 2010 IEEE INFOCOM Conference on Computer Communications* (2010), IEEE, pp. 1–9.

[244] ZHANG, D., YIN, J., ZHU, X., AND ZHANG, C. SINE: Scalable Incomplete Network Embedding. In *Proceedings of the 18th International Conference on Data Mining* (2018), IEEE, pp. 737–746.

[245] ZHANG, H., QIU, L., YI, L., AND SONG, Y. Scalable Multiplex Network Embedding. In *IJCAI* (2018), vol. 18, pp. 3082–3088.

[246] ZHANG, H., QIU, L., YI, L., AND SONG, Y. Scalable Multiplex Network Embedding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (2018), IJCAI'18, AAAI Press, p. 3082–3088.

[247] ZHANG, M., CUI, Z., NEUMANN, M., AND CHEN, Y. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI* (2018), pp. 4438–4445.

[248] ZHANG, Y., XIONG, Y., KONG, X., LI, S., MI, J., AND ZHU, Y. Deep Collective Classification in Heterogeneous Information Networks. In *Proceedings of the 2018 World Wide Web Conference* (2018), International World Wide Web Conferences Steering Committee, pp. 399–408.

[249] ZHOU, T., LÜ, L., AND ZHANG, Y.-C. Predicting Missing Links via Local Information. *The European Physical Journal B 71*, 4 (2009), 623–630.

[250] ZHOU, Z., ZHANG, N., AND DAS, G. Leveraging History for Faster Sampling of Online Social Networks. *Proceedings of the VLDB Endowment 8*, 10 (2015).

[251] ZLOTKIN, G., AND ROSENSCHEIN, J. S. Coalition, Cryptography, and Stability: Mechanisms for Coalition Formation in Task Oriented Domains. In *AAAI* (1994), pp. 432–437.