Karate Club: An API Oriented Open-Source Python Framework for Unsupervised Learning on Graphs

Benedek Rozemberczki
The University of Edinburgh
Edinburgh, United Kingdom
benedek.rozemberczki@ed.ac.uk

Oliver Kiss
Central European University
Budapest, Hungary
kiss_oliver@phd.ceu.edu

Rik Sarkar
The University of Edinburgh
Edinburgh, United Kingdom
rsarkar@inf.ed.ac.uk

ABSTRACT

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 paper 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.

KEYWORDS

network embedding, graph embedding, representation learning, network analytics, graph mining

ACM Reference Format:
Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. 2020. Karate Club: An API Oriented Open-Source Python Framework for Unsupervised Learning on Graphs. In *Proceedings of 16th International Workshop on Mining and Learning with Graphs (San Diego '20).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Techniques that extract features from graph data in an unsupervised manner [19, 24, 47] 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 [19, 24, 25, 30, 42] in a wide range of real world research and application scenarios. Graph mining tools such as [13, 17, 23] have contributed to enhancement and development of machine learning pipelines. This approach produces features that are naturally reusable on multiple types of tasks. The need for complicated custom feature engineering is reduced by unsupervised mining techniques. Recent research [24, 25, 36] 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, 5, 21, 22] 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 [13, 23]. For example, these packages host certain community detection algorithms, but none for whole graph or node embedding. In addition, some of these tools [17, 23] 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 [50] – 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 [5].

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 [13, 39, 40] 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 [9, 29, 39, 40] 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:

San Diego '20, 08-24, 2020, San Diego, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

(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 article is structured as follows. In Section 2 we discuss the covered graph mining techniques. We overview the main principles behind *Karate Club* in Section 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 4. We review related data mining libraries in Section 5. The paper concludes with Section 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.

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.

2.1 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, [18, 26, 28, 31]. While overlapping community detection is analogous to fuzzy clustering as nodes have an affiliation with multiple clusters; look for example [16, 34, 41, 47, 49].

2.2 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 [24, 25, 32] or explicitly [6, 15, 27, 35] decompose the 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 [2, 10, 14]. Nodes with similar embeddings have a similar distribution of centrality and clustering coefficient in their vicinity. Embeddings are distilled by the decomposition of matrices representing structural feature versus 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 [43, 46] or implicit matrix decomposition technique [30, 51].

Meta embedding combines information from neighbourhood preserving, structural and attributed embeddings in order to create higher representation quality embeddings [44].

2.3 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 [8, 36, 38]. Vectors of the descriptive statistics are used as the whole graph representation.

Implicit factorization techniques create a graph – structural feature matrix [7, 19] by enumerating string features in the graphs. This matrix is decomposed in order to create whole graph descriptors and feature embeddings jointly.

3 DESIGN PRINCIPLES

When we created *Karate Club*, we used an API oriented machine learning system design point of view [5, 22] 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.

3.1 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 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 1: Creating a synthetic graph, using a DeepWalk model with standard and modified hyperparameter settings.

3.2 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 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 step or algorithmic step (e.g. truncated random walk, Weisfeiler-Lehman hashing) were built on shared blocks.

In Figure 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 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 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.

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

Figure 3: Creating a synthetic graph, using the Walklets constructor, fitting the embedding and returning it.

3.3 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 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.

3.4 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, 21]. 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 [2, 24, 25, 30, 51] utilize the *GenSim* [29] package and methods which rely on graph signal processing use *PyGSP* [9].

3.5 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 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 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.

3.6 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 [7, 8, 12, 19, 36, 38] 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 [7, 19] 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.

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.

4.1 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 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

4.1.1 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] [17, 30]. 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.
- *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 1. As one can see these networks have a large variety of size, level of clustering and diameter.

Table 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.

We listed the size of these datasets with the respective descriptive statistics in Table 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 [42]. We would also like to emphasize that the use of graph kernels would not be feasible on graph datasets which are this numerous.

Table 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 [49]	.051 ± .001	.083 ± .001	.007 ± .001	.164 ± .001
M-NMF [41]	.063 ± .001	.084 ± .001	.004 ± .001	.068 ± .001
NNSED [34]	.063 ± .001	.034 ± .001	.004 ± .001	.072 ± .001
SymmNMF [16]	.062 ± .001	.074 ± .001	.007 ± .001	.206 ± .001
Ego-Splitting [11]	.157 ± .001	**.202 ± .001**	**.223 ± .001**	.346 ± .001
EdMot [18]	.085 ± .001	.180 ± .001	.008 ± .001	.272. ± .001
LabelProp [28]	.119 ± .001	.090 ± .002	.003 ± .001	.320 ± .004
SCD [26]	**.181 ± .001**	.189 ± .001	.169 ± .001	**.386 ± .001**
GEMSEC[31]	.102 ± .001	.127 ± .001	.008 ± .002	.244 ± .001

4.1.2 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 [11, 16, 34, 41, 47, 49] 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 3 the NMI averages with the standard errors calculated from 100 experimental runs.

Looking at Table 3 first we notice that the non-overlapping community detection techniques [11, 18, 26, 28, 31] materially outperform the overlapping models which create latent spaces [16, 34, 41, 47, 49] on every dataset in terms of NMI. Second, those algorithms which create clusters based on the presence of closed triangles (SCD [26], Ego-Splitting [11]) 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.

4.1.3 Graph classification. In each dataset we created representations for the graphs and use those as predictors for the downstream

[1] https://snap.stanford.edu/data/
[2] https://github.com/benedekrozemberczki/datasets

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 4 along with their standard errors.

Table 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 [7]	.753 ± .002	.664 ± .002	.551 ± .001	.504 ± .001
Graph2Vec [19]	.804 ± .002	.702 ± .003	.585 ± .001	.512 ± .001
SF [8]	.814 ± .002	.678 ± .003	.558 ± .001	.501 ± .001
NetLSD [36]	**.827 ± .001**	.631 ± .002	.632 ± .001	.522 ± .001
FGSD [38]	.825 ± .002	**.705 ± .003**	**.656 ± .001**	**.526 ± .001**
GeoScattering [12]	.800 ± .001	.697 ± .001	.546 ± .003	.522 ± .003

The results presented in Table 4 show that the representations created by implicit factorization [7, 19] and spectral finger printing [8, 36, 38] techniques are predictive on most problems. In addition, we see evidence that algorithms from the latter group create somewhat higher quality representations.

Table 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 [15]	.685 ± .001	.845 ± .001	.576 ± .001	.752 ± .001
NodeSketch [45]	.722 ± .001	.631 ± .001	.520 ± .001	.579 ± .001
Diff2Vec [32]	.832 ± .001	.858 ± .001	.589 ± .001	.873 ± .001
NetMF [27]	.866 ± .001	.867 ± .001	.629 ± .002	.946 ± .001
Walklets [25]	.875 ± .001	.899 ± .002	.622 ± .001	.973 ± .001
HOPE [20]	.870 ± .001	.844 ± .001	.612 ± .001	.909 ± .001
GraRep [6]	.888 ± .002	.876 ± .001	.609 ± .001	.952 ± .001
DeepWalk [24]	.850 ± .001	.872 ± .002	.597 ± .002	.877 ± .001
NMF-ADMM [35]	.747 ± .001	.784 ± .001	.619 ± .001	.937 ± .001
LAP [4]	.784 ± .001	.529 ± .001	.511 ± .001	.501 ± .001
GraphWave [10]	.517 ± .001	.620 ± .001	.583 ± .001	.613 ± .001
Role2Vec [2]	.845 ± .001	.862 ± .002	.601 ± .002	.903 ± .002
BANE [46]	.866 ± .002	.570 ± .001	.551 ± .001	.970 ± .002
TENE [48]	.907 ± .001	.874 ± .001	.615 ± .001	.886 ± .001
TADW [43]	.896 ± .001	.817 ± .001	.612 ± .002	.871 ± .001
FSCNMF [3]	.912 ± .001	.856 ± .002	.621 ± .001	.891 ± .001
SINE [51]	.904 ± .001	**.910 ± .002**	**.646 ± .001**	.979 ± .001
MUSAE [30]	**.931 ± .001**	.903 ± .001	.628 ± .001	**.981 ± .001**

4.1.4 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 5 with standard errors. The results in Table 5 generally demonstrate that the included neighbourhood based [4, 6, 15, 20, 24, 25, 27, 32, 35], structural role preserving [2, 10], and attributed [3, 30, 43, 46, 48, 51] node embedding techniques all generate reasonable quality representations for this classification task. There are additional conclusions; (i) multi-scale node embeddings such as *GraRep* [6], *Walklets*, [25], and *MUSAE* [30] create high quality node features , (ii) combining neighbourhood and attribute information results in the best representations [30, 51], (iii) there is not a single model which is generally superior.

4.2 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.

Figure 5 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.

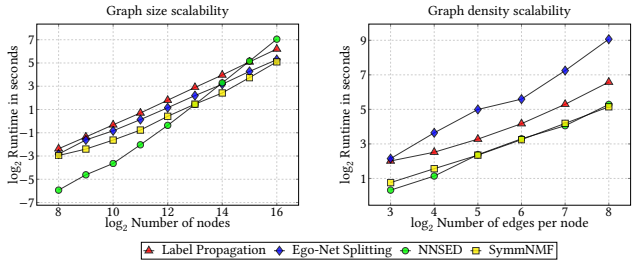

Figure 5: Scalability of the community detection procedures in Karate Club. We vary the number of nodes and the density of an Erdos-Renyi graph.

We measured the same way how the average runtime of node embedding varies with input size changes and densification and plotted these in Figure 6. 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. 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

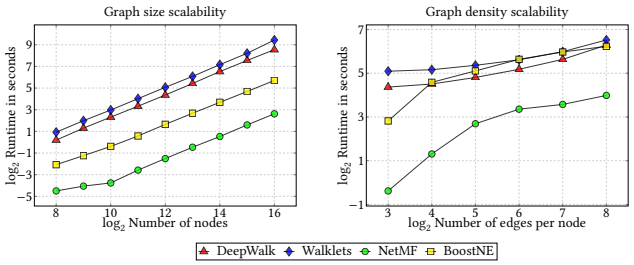

Figure 6: Scalability of node embedding procedures in Karate Club. We vary the number of nodes and the density of an Erdos-Renyi graph.

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 [8, 38] do not scale well when the size of the graphs is increased which was expected.

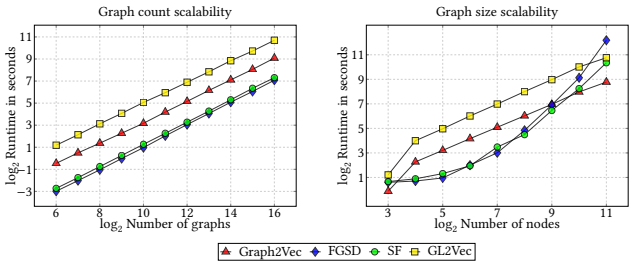

Figure 7: Scalability of graph embedding and summarization procedures in Karate Club. We vary the number of Erdos-Renyi graphs and their size.

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.

5.1 API oriented machine learning frameworks

Scikit-learn [5, 22] 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.

5.2 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* [17] library only covers specific methods which were created by the authors of the framework. The *NetworkX* [13] and *GraphTool* [23] libraries only provide tools for community detection. Node and whole graph embedding is not supported by these frameworks.

6 CONCLUSION AND FUTURE DIRECTIONS

In this work we described *Karate Club* a Python framework built on the open source packages *NetworkX* [13], *PyGSP* [9], *Gensim* [29], *NumPy* [40], and *SciPy Sparse* [39] 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 [33, 37].

ACKNOWLEDGEMENTS

Benedek Rozemberczki was supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Nesreen K Ahmed, Ryan A Rossi, John Boaz Lee, Theodore L Willke, Rong Zhou, Xiangnan Kong, and Hoda Eldardiry. 2019. role2vec: Role-based network embeddings. In *Proc. DLG KDD*.

[3] Sambaran Bandyopadhyay, Harsh Kara, Aswin Kannan, and M Narasimha Murty. 2018. Fscnmf: Fusing structure and content via non-negative matrix factorization for embedding information networks. *arXiv preprint arXiv:1804.05313* (2018).

[4] Mikhail Belkin and Partha Niyogi. 2002. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in neural information processing systems*. 585–591.

[5] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jacob VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. *ArXiv* abs/1309.0238 (2013).

[6] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 891–900.

[7] Hong Chen and Hisashi Koga. 2019. GL2vec: Graph Embedding Enriched by Line Graphs with Edge Features. In *International Conference on Neural Information Processing*. Springer, 3–14.

[8] Nathan de Lara and Pineau Edouard. 2018. A simple baseline algorithm for graph classification. In *Advances in Neural Information Processing Systems*.

[9] Michaël Defferrard, Lionel Martin, Rodrigo Pena, and Nathanaël Perraudin. [n.d.]. PyGSP: Graph Signal Processing in Python. https://doi.org/10.5281/zenodo.1003157

[10] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1320–1329.

[11] Alessandro Epasto, Silvio Lattanzi, and Renato Paes Leme. 2017. Ego-Splitting Framework: From Non-Overlapping to Overlapping Clusters. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. 145–154.

[12] Feng Gao, Guy Wolf, and Matthew Hirn. 2019. Geometric Scattering for Graph Data Analysis. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. 2122–2131.

[13] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[14] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1231–1239.

[15] Huan Liu Jundong Li, Liang Wu. 2019. 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*. ACM, 50–56.

[16] Da Kuang, Chris Ding, and Haesun Park. 2012. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 106–117.

[17] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[18] Pei-Zhen Li, Ling Huang, Chang-Dong Wang, and Jian-Huang Lai. 2019. EdMot: An Edge Enhancement Approach for Motif-aware Community Detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. 479–487.

[19] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, and Yang Liu. 2017. graph2vec: Learning distributed representations of graphs. (2017).

[20] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 1105–1114.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. 8024–8035.

[22] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[23] Tiago P Peixoto. 2014. The graph-tool python library. *figshare* (2014).

[24] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[25] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. 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*. ACM, 258–265.

[26] Arnau Prat-Pérez, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. 2014. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd international conference on World wide web*. 225–236.

[27] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 459–467.

[28] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks. *Physical review E* 76, 3 (2007), 036106.

[29] Radim Rehurek and Petr Sojka. 2011. Gensim—statistical semantics in python. *Retrieved from genism. org* (2011).

[30] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. *arXiv preprint arXiv:1909.13021* (2019).

[31] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. 2019. GEMSEC: Graph Embedding with Self Clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2019*. ACM, 65–72.

[32] Benedek Rozemberczki and Rik Sarkar. 2018. Fast Sequence-Based Embedding with Diffusion Graphs. In *International Workshop on Complex Networks*. Springer, 99–107.

[33] Rik Sarkar. 2011. Low distortion delaunay embedding of trees in hyperbolic plane. In *International Symposium on Graph Drawing*. Springer, 355–366.

[34] Bing-Jie Sun, Huawei Shen, Jinhua Gao, Wentao Ouyang, and Xueqi Cheng. 2017. A non-negative symmetric encoder-decoder approach for community detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 597–606.

[35] Dennis L Sun and Cedric Fevotte. 2014. Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 6201–6205.

[36] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. 2018. Netlsd: hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2347–2356.

[37] Kevin Verbeek and Subhash Suri. 2014. Metric embedding, hyperbolic space, and social networks. In *Proceedings of the thirtieth annual symposium on Computational geometry*. 501–510.

[38] Saurabh Verma and Zhi-Li Zhang. 2017. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *Advances in Neural Information Processing Systems*. 88–98.

[39] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2019. SciPy 1.0–fundamental algorithms for scientific computing in Python. *arXiv preprint arXiv:1907.10121* (2019).

[40] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering* 13, 2 (2011), 22–30.

[41] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. 2017. Community Preserving Network Embedding. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17)*. 203–209.

[42] Pinar Yanardag and S.V.N. Vishwanathan. 2015. Deep Graph Kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1365–1374.

[43] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[44] Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. 2017. Fast network embedding enhancement via high order proximity approximation.. In *IJCAI*. 3894–3900.

[45] Dingqi Yang, Paolo Rosso, Bin Li, and Philippe Cudre-Mauroux. 2019. NodeSketch: Highly-Efficient Graph Embeddings via Recursive Sketching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1162–1172.

[46] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. 2018. Binarized attributed network embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1476–1481.

[47] Jaewon Yang and Jure Leskovec. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 587–596.

[48] Shuang Yang and Bo Yang. 2018. Enhanced Network Embedding with Text Information. In *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 326–331.

[49] Fanghua Ye, Chuan Chen, and Zibin Zheng. 2018. Deep Autoencoder-like Nonnegative Matrix Factorization for Community Detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*. 1393–1402.

[50] Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of anthropological research* 33, 4 (1977), 452–473.

[51] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. SINE: Scalable Incomplete Network Embedding. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 737–746.