AUTOGL: A LIBRARY FOR AUTOMATED GRAPH LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent years have witnessed an upsurge of research interests and applications of machine learning on graphs. Automated machine learning (AutoML) on graphs is on the horizon to automatically design the optimal machine learning algorithm for a given graph task. However, all current libraries cannot support AutoML on graphs. To tackle this problem, we present Automated Graph Learning (AutoGL), the first library for automated machine learning on graphs. AutoGL is **open-source**, **easy to use**, and **flexible to be extended**. Specifically, We propose an automated machine learning pipeline for graph data containing four modules: auto feature engineering, model training, hyper-parameter optimization, and auto ensemble. For each module, we provide numerous state-of-the-art methods and flexible base classes and APIs, which allow easy customization. We further provide experimental results to showcase the usage of our AutoGL library.

1 INTRODUCTION

Machine learning on graphs has drawn increasing popularity in the last decade (Zhang et al., 2020c). For example, graph neural networks (GNNs) (Zhou et al., 2018; Wu et al., 2020) are the recent rising star and are shown successful in a wide range of applications such as recommendation (Ma et al., 2019; Li et al., 2021), traffic forecasting (Jiang & Luo, 2021), physical simulations (Shlomi et al., 2020), geometric data analysis (Bronstein et al., 2017), bioinformatics (Su et al., 2020), and combinatorial optimization (Bengio et al., 2020).

However, as the literature booms and the graph tasks are diverse, it becomes increasingly difficult to manually design the optimal machine learning algorithm for a given graph task. Therefore, there is an urgent need and recent research interest in automated machine learning (AutoML) on graphs. Essentially, AutoML on graphs combines the strengths of graph-based machine learning and AutoML techniques (Yao et al., 2018) to automate the design of graph-based machine learning algorithms. Initial successes have been shown in hyper-parameter optimization (HPO) (Tu et al., 2019) and neural architecture search (NAS) (Gao et al., 2020) for graph learning algorithms.

There exists a bunch of libraries and toolkits for machine learning on graphs, such as PyTorch Geometric (Fey & Lenssen, 2019), Deep Graph Library (Wang et al., 2019), GraphNets (Battaglia et al., 2018), AliGraph (Zhu et al., 2019), and PBG (Lerer et al., 2019). Besides, AutoML libraries such as AutoKeras (Jin et al., 2019), AutoSklearn (Feurer et al., 2019), Hyperopt (Bergstra et al., 2013), and NNI (Zhang et al., 2020a) are also available. Unfortunately, integrating these libraries is non-trivial due to the uniqueness and complexity of graphs. Currently, there is a lack of libraries for AutoML on graphs.

To tackle this problem, we present Automated Graph Learning (AutoGL), the first dedicated library for automated machine learning on graphs. The overall framework of AutoGL is shown in Figure 1. We summarize and abstract the pipeline of AutoML on graphs into four modules: auto feature engineering, model training, hyper-parameter optimization, and auto ensemble. For each module, we provide plenty of state-of-the-art algorithms and standardized base classes with high-level APIs for easy and flexible customization. The AutoGL library is built upon PyTorch Geometric (PyG) (Fey & Lenssen, 2019), a widely adopted graph learning library with high performance and efficiency. AutoGL has the following key characteristics:

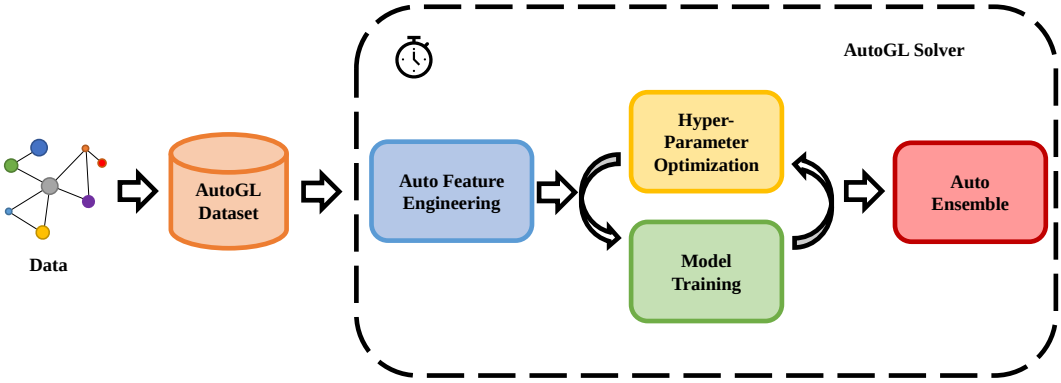

Figure 1: The overall framework of AutoGL.

- **Open source:** The codes[1] and detailed documentations[2] are available online.
- **Easy to use:** AutoGL is designed to be user-friendly. Users can conduct quick AutoGL experiments with less than 10 lines of codes.
- **Flexible to be extended:**. The modular design, high-level base class APIs, and extensive documentations of AutoGL allow flexible and customized extensions.

2 Automated Graph Learning

In this section, we elaborate on the detailed designs of AutoGL. AutoGL is designed in a modular and object-oriented fashion to enable clear logic flows, easy usages, and flexibility in extending. All the APIs exposed to users are abstracted and designed in a high-level fashion to avoid redundant re-implementation of cumbersome models, algorithms, and train/evaluation protocols.

As illustrated in Figure 1, we summarize and abstract the core problem of AutoML on graphs into a pipeline with four modules: auto feature engineering, model training, hyper-parameter optimization, and auto ensemble. All modules are designed while considering the specialty of graph structures.

AutoGL Dataset Before presenting methodology, we briefly introduce our dataset management. AutoGL Dataset is currently inherited from `Dataset` from PyTorch Geometric, covering common benchmarks for node and graph classification. Besides, we add supports for the recent Open Graph Benchmark (Hu et al., 2020). For a complete list of datasets, please refer to Appendix A. Users can also easily add customized datasets following our documentations. We expect our dataset design to support smooth transitions from conducting machine learning on graphs to AutoML on graphs.

Auto Feature Engineering The graph data is first processed by the auto feature engineering module, where useful features of nodes, edges, and graph are automatically added while useless ones are deleted. Graph topological features can also be extracted to better utilize graph structures.

Currently, we support 24 atom feature engineering operations abstracted into three categories: generators, selectors, and sub-graph generators. The generators aim to create new node and edge features based on the current node features and graph structures. The selectors automatically filter out and compress useless and meaningless features so that features can stay compact and informative. Sub-graph generators focus on generating graph-level features of the given data. For a complete list, please refer to Appendix B. We also provide convenient wrappers that support feature engineering operations in PyTorch Geometric (Fey & Lenssen, 2019) and NetworkX (Hagberg et al., 2008). Users can customize feature engineering methods by simply overwriting the class `BaseGenerator`, `BaseSelector`, and `BaseSubgraph`, or directly rewrite `BaseFeatureAtom` if the methods do not fit in the current categorization.

[1]To be released after the anonymous review period.
[2]To be released after the anonymous review period.

Table 1: The results of node classification

Model	Cora	CiteSeer	PubMed
GCN	80.9 ± 0.7	70.9 ± 0.7	78.7 ± 0.6
GAT	82.3 ± 0.7	71.9 ± 0.6	77.9 ± 0.4
GraphSAGE	74.5 ± 1.8	67.2 ± 0.9	76.8 ± 0.6
AutoGL	$\mathbf{83.2 \pm 0.6}$	$\mathbf{72.4 \pm 0.6}$	$\mathbf{79.3 \pm 0.4}$

Table 2: The results of graph classification

Model	MUTAG	PROTEINS	IMDB-B
Top-K Pooling	76.3 ± 7.5	72.7 ± 4.1	72.5 ± 4.6
GIN	85.7 ± 7.7	72.1 ± 5.1	72.8 ± 4.5
AutoGL	$\mathbf{92.7 \pm 3.6}$	$\mathbf{79.3 \pm 4.1}$	$\mathbf{76.4 \pm 4.6}$

Table 3: The results of HPO for node classification

Method	Trails	Cora		CiteSeer		PubMed	
		GCN	GAT	GCN	GAT	GCN	GAT
None		80.9 ± 0.7	82.3 ± 0.7	70.9 ± 0.7	71.9 ± 0.6	78.7 ± 0.6	77.9 ± 0.4
random	1	81.0 ± 0.6	81.4 ± 1.1	70.4 ± 0.7	70.1 ± 1.1	78.3 ± 0.8	76.9 ± 0.8
	10	82.0 ± 0.6	82.5 ± 0.7	71.5 ± 0.6	$\mathbf{72.2 \pm 0.7}$	79.1 ± 0.3	78.2 ± 0.3
	50	81.8 ± 1.1	$\mathbf{83.2 \pm 0.7}$	71.1 ± 1.0	72.1 ± 1.0	$\mathbf{79.2 \pm 0.4}$	78.2 ± 0.4
TPE	1	81.8 ± 0.6	81.9 ± 1.0	70.1 ± 1.2	71.0 ± 1.2	78.7 ± 0.6	77.7 ± 0.6
	10	82.0 ± 0.7	82.3 ± 1.2	71.2 ± 0.6	72.1 ± 0.7	79.0 ± 0.4	$\mathbf{78.3 \pm 0.4}$
	50	$\mathbf{82.1 \pm 1.0}$	83.2 ± 0.8	$\mathbf{72.4 \pm 0.6}$	71.6 ± 0.8	79.1 ± 0.6	78.1 ± 0.4

Model Training The model training module handles the training and evaluation process of graph tasks on the pre-processed dataset. It contains two functional sub-modules: model and trainer. Model handles the construction of graph machine learning models, e.g., GNNs, by defining learnable parameters and the forward pass under the current deep learning paradigm. Trainer controls the optimization process for the given model. Common optimization methods are packaged as high-level APIs to provide neat and clean interfaces. More advanced training control and regularization methods in graph tasks like early stopping and weight decay are also supported.

The model training module supports both node-level and graph-level tasks, e.g., node classification and graph classification. Commonly used models for node classification such as GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), and GraphSAGE (Hamilton et al., 2017), GIN (Xu et al., 2019), and pooling methods such as Top-K Pooling (Gao & Ji, 2019) are supported. Users can easily implement their own graph models by inheriting the `BaseModel` class. If needed, user can add customized tasks or optimization methods by overwriting `BaseTrainer`.

Hyper-Parameter Optimization This module aims at automatically selecting the best hyper-parameters of a specified model and the training process, including but not limited to the number of layers, the dimensionality of node representations, the dropout rate, the activation function, the learning rate, the weight decay, the number of epochs, etc. The hyper-parameters, their types (e.g., integer, numerical, or categorical), and feasible ranges can be set by the model training module.

We have supported various hyper-parameter optimization algorithms, including algorithms specified for graph data like AutoNE (Tu et al., 2019) and general-purpose algorithms like random search (Bergstra & Bengio, 2012), Tree Parzen Estimator (Bergstra et al., 2011), etc. Users can customize hyper-parameter optimization algorithms by inheriting the `BaseHPOptimizer` class.

Auto Ensemble The optimized individual models with promising performance can be automatically integrated by the auto ensemble module to form a more powerful final model. Currently, we have implemented two kinds of ensemble methods: voting and stacking. Voting is a simple yet powerful ensemble method that directly averages the output of individual models, while stacking trains another meta-model that learns to combine the output of models in a more principled way. Both general linear model (GLM) and gradient boosting machine (GBM) are supported as meta-models.

AutoGL Solver On top of the four modules, we provide another high-level API Solver to control the overall AutoML pipeline. In Solver, the four modules are integrated systematically to form the final AutoML solution for graph data. Solver also provides global controls of the whole AutoGL pipeline: the time budget can be explicitly set to restrict the maximum time cost, and the training/evaluation protocols can be selected from plain train/val settings or cross-validation.

Table 4: The results of HPO for graph classification

HPO	MUTAG		PROTEINS		IMDB-B	
	Top-K Pooling	GIN	Top-K Pooling	GIN	Top-K Pooling	GIN
None	76.3 ± 7.5	85.7 ± 7.7	72.7 ± 4.1	72.1 ± 5.1	72.5 ± 4.6	72.8 ± 4.5
random	90.6 ± 5.0	$\mathbf{93.3 \pm 5.4}$	79.1 ± 3.8	$\mathbf{77.3 \pm 4.4}$	$\mathbf{74.6 \pm 3.2}$	$\mathbf{76.4 \pm 4.6}$
TPE	$\mathbf{92.2 \pm 5.7}$	92.7 ± 3.6	$\mathbf{79.3 \pm 4.1}$	75.1 ± 4.5	72.4 ± 3.0	75.0 ± 3.4

3 EVALUATION

In this section, we provide experimental results to showcase the usage of AutoGL as well as main functional modules. For node classification, we use Cora, CiteSeer, and PubMed with the default dataset splits from (Kipf & Welling, 2017). For graph classification, we follow (Yanardag & Vishwanathan, 2015) and report the average accuracy of 10-fold cross-validation on MUTAG, PROTEINS, and IMDB-B.

Table 5: The performance of ensemble modules of AutoGL for the node classification task.

Base Model	Cora	CiteSeer	PubMed
GCN	81.1 ± 0.9	69.6 ± 1.1	$\mathbf{78.5 \pm 0.4}$
GAT	82.0 ± 0.5	70.4 ± 0.6	77.7 ± 0.5
Ensemble	$\mathbf{82.2 \pm 0.4}$	$\mathbf{70.8 \pm 0.5}$	$\mathbf{78.5 \pm 0.4}$

AutoGL Results We turn on all the functional modules in AutoGL, and report the fully automated results in Table 1 and 2. We use the best single model for graph classification under the cross-validation setting. We observe that in all the benchmark datasets, AutoGL achieves better results than vanilla models, demonstrating the importance of AutoML for graph machine learning and the effectiveness of the proposed pipeline in the released library.

Hyper-Parameter Optimization Table 3 reports the results of two implemented HPO methods, i.e., random search and TPE, for the semi-supervised node classification taks. As shown in the table, with larger number of trials, both HPO methods tend to lead to better results. In general, TPE algorithm performs similar to random search, both of which achieve better performances than vanilla models without HPO. We further try these HPO methods with 10 trails on the graph classification task, and report the results in Table 4. The results show quite significant improvements over not conducting HPO on all datasets. Besides, different models prefer different HPO methods. GIN seems to prefer random search while Top-K Pooling has better performance using TPE.

Auto Ensemble Table 5 report the performance of ensemble module as well as its base learners for the node classification task. We use voting as the example ensemble method and choose GCN and GAT as the base learners. As shown in Table 5, the ensemble module shows better performance than both the base learners it uses, demonstrating the effectiveness of the implemented ensemble module.

4 CONCLUSION AND FUTURE PLANS

We present AutoGL, the first library for automated machine learning on graphs, which is open source, easy to use, and flexible to be extended. Currently, we are actively developing AutoGL and plan to support the following functionalities within a short time:

- Neural architecture search.
- Support for large-scale graphs.
- More graphs and tasks, e.g., heterogeneous graphs and spatial temporal graphs.
- Support for more graph library backends, e.g., Deep Graph Library.

REFERENCES

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24, 2011.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pp. 115–123, 2013.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21:i47–i56, 2005.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pp. 113–134. Springer, Cham, 2019.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 1403–1409, 7 2020.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, 2008.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *arXiv preprint arXiv:2101.11174*, 2021.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1946–1956, 2019.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, 2019.

Haoyang Li, Xin Wang, Ziwei Zhang, Jianxin Ma, Peng Cui, and Wenwu Zhu. Intention-aware sequential recommendation with structured intent transition. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Jianxin Ma, Chang Zhou, Peng Cui, Hongxia Yang, and Wenwu Zhu. Learning disentangled representations for recommendation. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. Deep inductive graph representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 32(3):438–452, 2018.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2020.

Chang Su, Jie Tong, Yongjun Zhu, Peng Cui, and Fei Wang. Network embedding in biomedical data science. *Briefings in bioinformatics*, 21(1):182–197, 2020.

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

Ke Tu, Jianxin Ma, Peng Cui, Jian Pei, and Wenwu Zhu. Autone: Hyperparameter optimization for massive network embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 216–225, 2019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv:1909.01315*, 2019.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, and Lidong Zhou. Retiarii: A deep learning exploratory-training framework. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pp. 919–936, 2020a.

Ziwei Zhang, Peng Cui, Jian Pei, Xin Wang, and Wenwu Zhu. Eigen-gnn: A graph structure preserving plug-in for gnns. *arXiv:2006.04330*, 2020b.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020c.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: A comprehensive graph neural network platform. *Proc. VLDB Endow.*, 12 (12):2094–2105, 2019.

A DATASETS

Currently, we provide off-the-shelf node classification datasets including Cora, CiteSeer, PubMed (Sen et al., 2008), Amazon Computers, Amazon Photo, Coauthor CS, Coauthor Physics (Shchur et al., 2018), Reddit (Hamilton et al., 2017), and graph classification datasets MUTAG (Debnath et al., 1991), PROTEINS (Borgwardt et al., 2005), IMDB-B, IMDB-M, COLLAB (Yanardag & Vishwanathan, 2015), Datasets from Open Graph Benchmark (Hu et al., 2020) including node classification datasets ogbn-products, ogbn-proteins, ogbn-arxiv, and ogbn-papers100M, and graph classification datasets ogbg-molhiv, ogbg-molpcba, ogbg-ppa, and ogbg-code. Table A.1 summarizes the statistics of the supported datasets.

B AUTO FEATURE ENGINEERING

We summarize the supported generators in Table B.2, including Graphlets (Milo et al., 2002), EigenGNN (Zhang et al., 2020b), PageRank (Brin & Page, 1998), local degree profile, normalization, one-hot degrees, and one-hot node IDs. For selectors summarized in Table B.3, GBDT (Ke et al., 2017) and FilterConstant are supported. An automated feature engineering method DeepGL (Rossi et al., 2018) is also supported, functioning as both a generator and a selector. For subgraph, Netlsd (Tsitsulin et al., 2018) and a set of subgraph feature extractors implemented in NetworkX (Hagberg et al., 2008) are wrapped, i.e., `NxLargeCliqueSize`, `NxAverageClusteringApproximate`, `NxDegreeAssortativityCoefficient`, `NxDegreePearsonCorrelationCoefficient`, `NxHasBridge`, `NxGraphCliqueNumber`, `NxGraphNumberOfCliques`, `NxTransitivity`, `NxAverageClustering`, `NxIsConnected`, `NxNumberConnectedComponents`, `NxIsDistanceRegular`, `NxLocalEfficiency`, `NxGlobalEfficiency`, and `NxIsEulerian`.

Table A.1: The statistics of the supported datasets. For datasets with more than one graph, #Nodes and #Edges are the average number of all the graphs. #Features correspond to node features by default and edges features are specified.

Dataset	Task	#Graphs	#Nodes	#Edges	#Features	#Classes
Cora	Node	1	2,708	5,429	1,433	7
CiteSeer	Node	1	3,327	4,732	3,703	6
PubMed	Node	1	19,717	44,338	500	3
Reddit	Node	1	232,965	11,606,919	602	41
Amazon Computers	Node	1	13,381	245,778	767	10
Amazon Photo	Node	1	7,487	119,043	745	8
Coauthor CS	Node	1	18,333	81,894	6,805	15
Coauthor Physics	Node	1	34,493	247,962	8,415	5
ogbn-products	Node	1	2,449,029	61,859,140	100	47
ogbn-proteins	Node	1	132,534	39,561,252	8(edge)	112
ogbn-arxiv	Node	1	169,343	1,166,243	128	40
ogbn-papers100M	Node	1	111,059,956	1,615,685,872	128	172
Mutag	Graph	188	17.9	19.8	7	2
PTC	Graph	344	14.3	14.7	18	2
ENZYMES	Graph	600	32.6	62.1	3	6
PROTEINS	Graph	1,113	39.1	72.8	3	2
NCI1	Graph	4,110	29.8	32.3	37	2
COLLAB	Graph	5,000	74.5	2,457.8	-	3
IMDB-B	Graph	1,000	19.8	96.5	-	2
IMDB-M	Graph	1,500	13.0	65.9	-	3
REDDIT-B	Graph	2,000	429.6	497.8	-	2
REDDIT-MULTI5K	Graph	5,000	508.5	594.9	-	5
REDDIT-MULTI12K	Graph	11,929	391.4	456.9	-	11
ogbg-molhiv	Graph	41,127	25.5	27.5	9, 3(edge)	2
ogbg-molpcba	Graph	437,929	26.0	28.1	9, 3(edge)	128
ogbg-ppa	Graph	158,100	243.4	2,266.1	7(edge)	37

Table B.2: Supported generators in the auto feature engineering module.

Name	Description
`graphlet`	Local graphlet numbers
`eigen`	EigenGNN features.
`pagerank`	PageRank scores.
`PYGLocalDegreeProfile`	Local Degree Profile features
`PYGNormalizeFeatures`	Row-normalize all node features
`PYGOneHotDegree`	One-hot encoding of node degrees.
`onehot`	One-hot encoding of node IDs

Table B.3: Supported selectors in the auto feature engineering module.

Name	Description
`SeFilterConstant`	Delete constant features and node features that appear only once
`gbdt`	Select the top-k important node features ranked by Gradient Boosting Decision Tree