Evaluation of topological features' influence on a node classification task using different GNN architectures

Nikola Bulat Alessandro Resta
{nikola.bulat, alessandro.resta}@studenti.unipd.it

Abstract

In network analysis, accurate classification of graph nodes is an essential component, with applications ranging from social networks to biological systems. We propose a novel study that aims to understand how different topological features impact the predictive performance of GNN and simple linear models when integrated with the original node features of two benchmark datasets. Features extracted from the graph topology go from simple clustering coefficient, betweenness centrality and PageRank, to more complex strategies like Node2Vec [7]. Furthermore, we conduct an in-depth analysis of different integration techniques, and describe how and in which proportion our models react to them for the different features. Extensive experimental results show how choices in the features, integration method, and dataset are crucial for getting the most out of our models' performance. The code is available at this link.

1. Introduction

The usage of topological features to improve the performance of main algorithms employed in node classification tasks is not new in the field of Graph Machine Learning and Network Science. Neural Network-based architectures like GCN [10] and GAT [18] are designed to work with unstructured data and take into account the network's topology, by iteratively building an embedding for each node that aggregates information from neighboring nodes. Even though this approach works quite well in practice, different types of informative data about the graph structure can be computed by external algorithms. The addition of these topology-related data has been proven to be useful in enhancing the power of GNNs and other types of models, see [3], [5] and [12]. In this project, we first try to understand how these topological features can be extracted and combined with original node data, in order to be used by two GNN models and three simple linear models. Then, as our main goal, we evaluate the influence that these features have on the performance of each of the tested models. Eventually, we summarize our findings into a table and discuss them quantitatively and qualitatively. Our experiments show that the most influential features are the positional ones, which allow both linear and GNN models to boost test accuracy by a significant margin when the appropriate feature combinations technique is applied.

2. Related Work

In literature it is common to find two main types of topological features that we can extract from a graph, each performing best on specific domains of application, as explained in [4]. Different ways of computing these features also exist, even though the most common include centrality measures and random walk-based approaches [7] [15]. The incorporation techniques of those features to our existing node features also vary, one being concatenation, as successfully done in [3], [5], and [12]. As well as using an MLP layer to create an embedding of node and topological features, as shown in [12]. In this paper, we exploit some of the approaches already used in previous works, plus some more, to reach the best performance in our domain of interest.

3. Architectures

In order to test the influence that topological features have on the node classification task, we decided to use two GNN architectures, GCN [10] and GAT [18], and three simple linear models, Logistic Regression (LR), Support Vector Machine (SVM) and Decision Tree (DT). The linear models were implemented using the API provided by Scikit-learn [14] and cuML [16] (for GPU acceleration). Regarding the GNNs, we recreated the same structure and hyperparameters as in their original papers, by implementing them from scratch using the framework PyTorch Geometric [6]. Since these architectures are ubiquitous in the literature, we find them the most suitable for testing the influence of additional features.

3.1. Linear Models

The aforementioned linear models were picked based on their popularity and simplicity, to test how the topological features can influence the prediction in much simpler models compared to GNNs. We used the default hyperparameters present in the library Scikit-learn for the DT model, and the ones in cuML [16] for the LR and SVM models. The cuML library was particularly helpful in speeding up the training process, thanks to its GPU support.

3.2. GCN

GCN [10] is a specialized type of Neural Network (NN) that extends the idea of convolution (used in CNN [11]) on irregular graph structures. The authors motivate the choice of the convolutional architecture via a localized first-order approximation of spectral graph convolutions [10]. Given its widespread adoption and success within the field, we have chosen to integrate it into our project.

3.3. GAT

GAT [18] is another widely used type of NN designed to work on irregular data structures. Unlike traditional convolutional approaches, GAT leverages an attention mechanism that assigns different attention scores to neighboring nodes during the aggregation step. Learning the neighborhood weights from data leads to improved expressivity of the model.

4. Dataset

In our experiments, we utilized two datasets: ArXiv (ogbn-arxiv) and Cora [17]. ArXiv is our main dataset of interest, but we also included Cora for practical reasons. In particular, Cora is a straightforward and widely used benchmark dataset that comes from the same domain as ArXiv. We employed Cora to assess our methods and gather preliminary results. Both ArXiv and Cora networks were converted to undirected graphs (using NetworkX [8]) before further processing, allowing for higher starting accuracy values.

4.1. ArXiv

In our experiments, we conducted analyses on the ogbn-arxiv dataset, which is part of the Open Graph Benchmark [9] available at this link. This dataset is a directed graph composed of 169 343 vertices and 1 166 243 edges, that represents the citation network between all Computer Science arXiv papers indexed by MAG [19]. Each node is an arXiv paper and has a 128-dimensional feature vector that is obtained by averaging the embeddings of words in its title and abstract. The embeddings of individual words are computed by running the skip-gram model [13] over the MAG corpus. Each directed edge indicates that one paper cites another one. The task is to assign one of the 40 subject areas (classes) to each of these CS papers, e.g. AI, OS, etc. The dataset split is based on the publication date of the papers. Specifically, the authors propose to train on papers published until 2017, validate on those published in 2018, and test on those published since 2019.

4.2. Cora

The Cora [17] dataset consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary that consists of 1433 unique words.

5. Features

As comprehensively discussed in [4], two primary categories of artificial node features can be derived from the topology of a graph: structural node features and positional node features. In our study, we decided to include both types to take into account a wider spectrum of information that GNNs can use to improve accuracy on node classification. Subsequent sections describe in more detail on the nature of these two artificial feature types.

5.1. Structural

Structural node features [4] help GNNs capture structural information of each node in the graph, such as degree information and neighborhood connection patterns. It follows that nodes that share relatively similar local neighborhood structures will most likely be close to each other in the GNN's embedding space. In this project, we incorporated a comprehensive set of structural features, including: clustering coefficient, betweenness centrality, degree centrality, eigenvector centrality, PageRank, node clique number, louvain communities [2], average neighbor degree, and core number. The choice of these features is due to their large adoption in previous works (see [5][12] and [1]) and their availability in the Python library NetworkX [8], which was used for their computation.

5.2. Positional

Positional node features [4] help GNNs capture node distance information regarding their relative positions in the graph. In citation networks, for instance, two authors that cite each other and also cite, or get cited, by similar other authors should be positionally close in the graph, meaning that they probably share the same research interests. To this extent, positional features will help GNNs put such positionally-close nodes close in the embedding space, fostering the same node classification output. There are

various approaches that implement this idea, like random walk, eigen decomposition of the adjacency matrix, one-hot encoding of nodes, Deep Walk [15] etc. In this project we adopted the algorithmic framework Node2Vec [7], which is a generalization of common random walk-based approaches. It adds more flexibility in the neighborhood notion of a node and neighborhood explorations, allowing the learning of richer representations. We implemented this framework using PyTorch Geometric [6], and fine-tuned the hyperparameters *number of epochs* and *embedding size* for the two datasets. Given their nature, and the fact that we are using datasets that describe citation graphs, we expect positional features to positively, and significantly, contribute to the performance of GNNs and linear models.

6. Methods

Diverse sets of features can be combined in different ways ranging from simple concatenation to more sophisticated frameworks. In this section, we describe the methods that we implemented and tested.

6.1. Concatenation

The simplest method is to concatenate the additional features to the original features of the dataset. This approach is inspired by the related papers [3], [5], [12]. Their authors tackle different problems and add additional steps, but the common denominator was building the final feature vector by concatenating the external and topological node features. Scaling techniques have an important role in this context as they represent different ways of combining distinct sets of features. It is interesting to test whether forcing the external and topological features to have the same scale leads to improvements in terms of prediction accuracy. For this reason, we have tested the concatenation method with standard normalization, min-max normalization, and sum-to-one normalization (only used for Cora node features).

6.2. MLP Preprocessing

Another approach to feature combination involves processing their concatenation with a fully-connected layer, followed by a non-linear activation function. This methodology, introduced in [12], facilitates the creation of a novel representation of the features altogether, which has been proven to raise the performance of GNNs. As in the concatenation method, we employed standard and min-max normalization techniques before feeding the features to the MLP layer.

6.3. Ensemble

Finally, we have developed and used a particular version of the stacking ensemble method to combine the features. The first step is to train the same type of base model on the training set, using distinct sets of features (original, structural and positional). Then, we train a meta-model on the feature vector composed of the predictions of all the base models on the validation set, concatenated with the features they have been trained on. Additionally, we tried to apply min-max and standard normalization to the meta-model features only, to seek improvements. In the context of our experiments, this method can be seen as a more sophisticated way of combining the external and topological features. Training the base models on distinct types of features should allow them to classify based on different patterns in the data. Their advantages are leveraged by training a meta-model on top of them.

7. Experimental Set-Up

We structured our experiments to fulfill the objective of evaluating the impact of topological features on the node classification task. Throughout all the experiments we opted for the fundamental architectures outlined in 3. These simple and common architectures allow us to have the best comparison between models trained on different features. GNNs underwent training for 200 epochs, with model selection based on the validation set, while the linear models were trained until convergence. We explored all feature combinations, namely: original, positional, structural, original-positional, original-structural, structural-positional and original-structural-positional. The baselines are the models trained only on the original features.

We conducted the independent experiments for all of our models trained on the aforementioned combinations of features, using all of the methods (with different normalization techniques) described in 6 to put the features together. Moreover, within the ensemble method, there are more different options regarding the choice of the meta-model. For simplicity, we opted for linear models (SVM, DT, and LR), however the approach can easily be extended to more expressive meta-models. Given the extensive number of combinations tested, we will focus on reporting the results that are significant in terms of performance and/or interpretation.

Each experiment, excluding the linear models (which will be elaborated shortly), was independently conducted 10 times, and the results were averaged. Across these experiments, we monitored parameters such as accuracy, running time, loss, and variance on both datasets independently. For the linear models, we executed the experiments only once due to practical reasons. Firstly, the ArXiv dataset's size requires a considerable time for the linear model training, or substantial GPU memory if we use the cuML [16] library

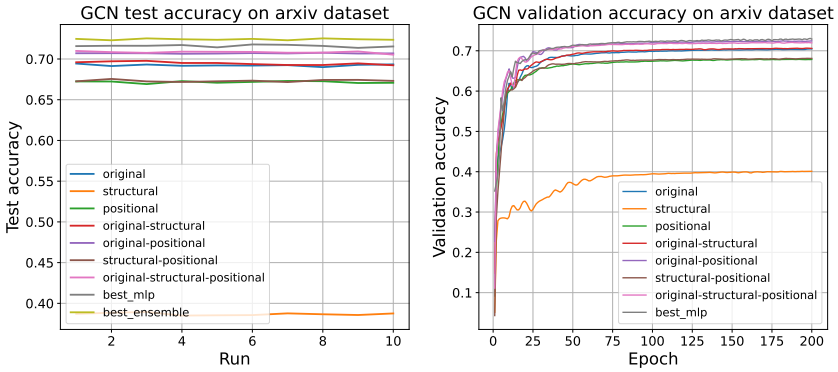

Figure 1. The image on the left shows test accuracy over 10 runs of execution of the best-performing GCN models (See table 1) trained on ArXiv dataset. We can clearly see that the top accuracy is reached by the ensemble method, followed by the MLP preprocessing approach and original-structural-positional model (which is similar to the original-positional model since structural features produce a very low accuracy alone). The image on the right shows the validation accuracy over 200 epochs for the GCNs with and without MLP preprocessing.

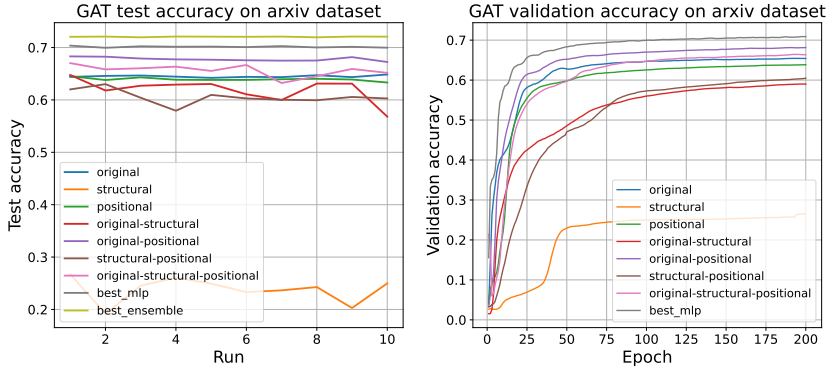

Figure 2. Like Figure 1, we report the same plot for the best-performing GAT models (refer to Table 1) on the ArXiv dataset. The image on the left shows that the top 3 models are, in order, ensemble, MLP preprocessing, and original-positional; notice how they are also the top 3 for GCN architecture. The image on the right shows the validation accuracy over the 200 epochs.

to take advantage of tensor computation. Secondly, most of the linear models we employed (Logistic Regression and SVM) produced a very low variance in the results. We used the Google Colab free version as environment for our experiments, mostly for its powerful GPU support.

8. Results

For the ArXiv dataset, a graphical visualization of the GNN results is shown in Figures 1 and 2. On the left-hand side of both figures we have the test accuracy of the best tested models (refer to section 6) over 10 runs of repeated execution, to account for variance. For both GCN and GAT the top 3 best-performing solutions are the ensemble, MLP, and original-positional concatenation. Structural features are very down in the graph compared to positional features, and that's not surprising, since in section 5 we highlighted how positional features work and how effective they can be on citation networks, like ArXiv. On the right-hand side of both 1 and 2, we also reported the validation accuracy over 200 epochs of GCN and GAT models with and without the usage of the MLP preprocessing layer, and again results match our expectations.

Regarding Cora (see figure 3 and 4), since it is our secondary dataset used for quick benchmarking and hypothesis testing, we report just the test accuracy over 10 runs of execution of the best models. Compared to ArXiv, there is more variance in the accuracy values of the models, hence it is not easy to locate the top ones from this graph alone. However, the ensemble model is clearly above all for both GCN and GAT architectures.

In Table 1 we report the results of our experiments for the best-performing instances (in the table just methods) of the approaches in 6, for both ArXiv and Cora datasets. We also

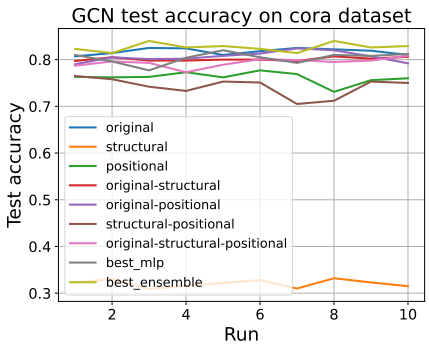

Figure 3. This figure shows the test accuracy of the best-performing GCN models (described in section 6) on the Cora dataset, over 10 runs of execution. This time, the curves appear more twisted than ArXiv, but ensemble and original-positional models performs again better than the others.

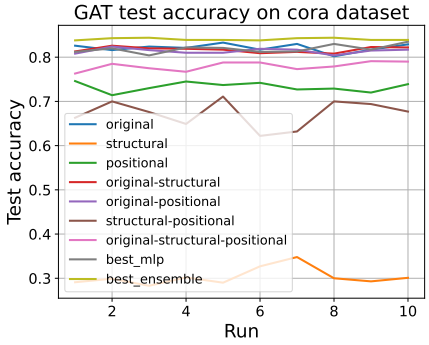

Figure 4. Similarly to 3, we report the test accuracy of the best-performing GAT models on the Cora dataset. Even in this case, most models overlap a lot, especially above 0.80, except the ensemble, which is clearly separated and better than the others.

include the baselines (first five rows) and models trained just on positional features (following three rows) to simply show their interesting and strong impact on the accuracy. By instance of a method here we refer to the choices being made after fixing that method. In particular, for concatenation these choices are: features to combine and normalization function; for MLP preprocessing are: features to combine, normalization function, and number of MLP hidden channels; and for the ensemble are: meta-model, base models, and normalization function. The method names in the table follow the structure:

- concatenation: architecture name (e.g. GCN, SVM), feature concatenations (e.g. OrigPos for the concatenation of original and positional features), and normalization function (e.g. Std for standard normalization of the features);
- MLP preprocessing: architecture name, the prefix "Pre", feature concatenations, normalization function and number of MLP hidden channels (e.g. 160);
- ensemble: meta-model (e.g. LR for Logistic Regression as meta-model), the word "Ensemble", base models (e.g. SVM, GCN), and normalization function of the meta-model.

Since it happened that some methods were top-performing on ArXiv but not on Cora, to make the comparison fair, in such cases we report the accuracy of the top-performing instances on Cora for the Cora dataset values. These accuracy values are highlighted by the symbol (*) in the table. For example, the 10th row in Table 1 has the value of 80.0% on Cora, which was achieved by a different, and better, normalization function than the one displayed on the left, but still, using GCN and the concatenation method. Besides the accuracy values on the test data, the table also includes the execution times in seconds and the standard deviation for the methods with a non-deterministic output value.

Method	**ArXiv**	**Cora**
DTOrig (baseline)	24.7	49.1
SVMOrigStd (baseline)	56.5	55.6
LROrigStd (baseline)	52.4	57.4
GCNOrigStd (baseline)	69.2 $\pm$ 0.001 (24s)	81.8 $\pm$ 0.005 (1.6s)
GATOrigStd (baseline)	64.5 $\pm$ 0.002 (39s)	83.0 $\pm$ 0.006 (2.3s)
SVMPos	68.5	72.3
GCNPosStd	67.1 $\pm$ 0.001 (23s)	72.6 $\pm$ 0.01 (1.4s)
GATPos	64.2 $\pm$ 0.002 (38s)	73.6 $\pm$ 0.01 (1.9s)
SVMOrigPos	71.0	72.3
GCNOrigPosStd	70.6 $\pm$ 0.001 (24s)	80.0* $\pm$ 0.006 (1.6s)
GATOrigPosStd	67.8 $\pm$ 0.003 (38s)	81.4* $\pm$ 0.009 (2.1s)
GCNPreOrigPos160	71.6 $\pm$ 0.001 (27s)	74.8 $\pm$ 0.007 (1.6s)
GATPreOrigPos160	70.1 $\pm$ 0.001 (42s)	74.8 $\pm$ 0.02 (2s)
LREnsembleSVM	70.6	79.1
LREnsembleGCN	**72.0** $\pm$ 0.001 (49s)	82.6 $\pm$ 0.008 (15s)
SVMEnsembleGATStd	**72.0** $\pm$ 0.001 (98s)	**83.6*** $\pm$ 0.005 (4.7s)

Table 1. Average test accuracy in % of the best-performing models on ArXiv and Cora datasets, along with standard deviation and training execution times in seconds. Values followed by (*) refer to a different method's instance, that led to the top accuracy. Ensemble's base models are always two, and trained on original and positional features, separately.

9. Discussion

The results show how the choice of features can strongly impact the models' performance. From all of the above plots, it is clear that structural features perform poorly on their own. Moreover, they either have a very small positive influence when combined with other features or they even bring the accuracy down. On the other hand, positional features perform well both on their own and in combination with the original features. In this section, we analyze this

influence referring to the best-performing models reported in Table 1.

The most extreme case of positional features influence can be seen in linear models. In particular, the accuracy of the best-performing linear baseline SVMOrigStd on the ArXiv dataset is 56.5, while the same model without standardization and with positional features (SVMPos) achieves the accuracy of 68.5. For the Cora dataset, we observe the similar behavior. However, the ensemble method (that here uses 2 base models trained on original and positional features respectively) is effective only on the Cora dataset, while the simple concatenation of positional features works better on ArXiv. In both cases, we have that the best method of including positional features in the linear model has similar performance to GNNs. On Cora, LREnsembleSVM is only 2.7% worse than GCNOrigStd (GCN baseline), while on ArXiv it is better than both GNN baselines and only 1% worse than LREnsembleGCN and SVMEnsembleGATStd (best Ensembles methods for GCN and GAT).

The first thing to note about GNNs is that on the ArXiv dataset, somewhat suprisingly, GAT baseline (64.5%) performs significantly worse than GCN baseline (69.2%). Our accuracy on Cora matches the basic architectures in the original papers, and we have used the same structure and hyperparameters on ArXiv. It is possible that further fine-tuning of GAT could achieve competitive performance with respect to GCN. As our goal is to test the influence of topological features on basic architectures, we decided that keeping the original structure allows for a fair comparison.

GNNs trained only on positional features (GCNPosStd and GATPos 1) are slightly worse than baselines on ArXiv, while the difference is more prominent on the Cora dataset (9.2% and 9.4% for GCN and GAT respectively). In any case, they do appear as potentially useful features that can positively influence the models' accuracy. Indeed, on ArXiv, the concatenation of positional features with original features plus standardization improves both GNN baselines, especially GAT (see GCNOrigPosStd and GATOrigPosStd). Conversely, on Cora, we observe that concatenating positional features makes the performance worse with respect to baselines. The same goes for MLP Preprocessing (see GCNPreOrigPos160 and GATPreOrigPos160) that is effective on the ArXiv dataset, but downgrades the performance on Cora. However, to check if the improvement on ArXiv is not only due to an additional layer rather than the more sophisticated combination of features, we have to compare it with a model that has an additional MLP layer trained just on original features. From our experimentation, that we decided not to report, such models achieve the accuracy of 70.53% and 67.9% for GCN and GAT respectively. Given that the GCNPreOrigPos160 and GATPreOrigPos160 achieve 71.6% and 70.1%, the conjecture is that MLP Preprocessing indeed finds a better way to combine original and positional features.

Finally, we inspect our top-performing models, the ensembles. On the ArXiv dataset, ensemble is very effective with GAT (SVMEnsembleGATStd) where it shows relevant improvement compared to all the previous models (72.0% accuracy reached). At the same time, the GCN ensemble (LREnsembleGCN) is noticeably better than the baseline (72.0% vs. 69.2% accuracy), and slightly better then MLP Preprocessing (72.0% vs. 71.6%). It's very interesting to note that the ensemble actually improves the baseline performance on Cora for both GNNs. This improvement is minor, but we have to take into account that all the previous attempts to include the positional features downgraded the model. These results suggest that more sophisticated ways of combining the features, like the ensemble in this context, do show potential.

10. Conclusion

Our results agree with [4] on the superiority of positional features with respect to structural in the context of citation networks. In all of our experiments structural features show no improvements while the positional features positively influence the accuracy in most cases. However, in order to improve the accuracy in combination with external features, it is important that models trained on positional features perform similarly to baselines. If that is not the case, as with GNNs on Cora, finding the proper way of including both types of features becomes a challenge.

The usefulness of positional features is, as expected, the most prominent on linear models. Equipping linear models with topological information, that they do not usually have, gives them a major performance improvement. In fact, simple linear models with positional features almost reach GNNs accuracy. On the other hand, GNNs positively respond to the addition of positional features, but the quality of the features and the inclusion method become crucial.

Lastly, our conjecture is that methods of combining different types of features are well worth exploring. In fact, a very simple ensemble (only two base models) improves the accuracy even when the models trained on positional features do not perform well on their own. However, the improvements are not major, and additional exploration (and tuning) of feature integration methods and positional features algorithms represent the main path of future work.

References

[1] Roy Abel, Idan Benami, and Yoram Louzoun. Topological based classification using graph convolutional networks. *CoRR*, abs/1911.06892, 2019.

[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, Oct. 2008.

[3] Xin Chen, Miao Liu, Yue Peng, and Benyun Shi. Can higher-order structural features improve the performance of graph neural networks for graph classification? In *2022 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pages 788–795, 2022.

[4] Hejie Cui, Zijie Lu, Pan Li, and Carl Yang. On positional and structural node features for graph neural networks on non-attributed graphs, 2022.

[5] Mojtaba Dezvarei, Kevin Tomsovic, Jinyuan Stella Sun, and Seddik M. Djouadi. Graph neural network framework for security assessment informed by topological measures, 2023.

[6] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[8] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[9] 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, 2021.

[10] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Asmaa M. Mahmoud, Abeer S. Desuky, Heba F. Eid, and Hoda A. Ali. Node classification with graph neural network based centrality measures and feature selection. *International Journal of Electrical and Computer Engineering (IJECE)*, 2023.

[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.

[16] Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *arXiv preprint arXiv:2002.04803*, 2020.

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

[18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[19] Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft Academic Graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 02 2020.