Which Graph Properties Affect GNN Performance for a Given Downstream Task?

Pavel Procházka[1,*], Michal Mareš[1,2] and Marek Dědič[1,3]

[1]Cisco Systems, Inc., Karlovo náměstí 10, Prague, Czech Republic
[2]Czech Technical University in Prague, Technická 2, Prague, Czech Republic
[3]Czech Technical University in Prague, Trojanova 13, Prague, Czech Republic

Abstract
Machine learning algorithms on graphs, in particular graph neural networks, became a popular framework for solving various tasks on graphs, attracting significant interest in the research community in recent years. As presented, however, these algorithms usually assume that the input graph is fixed and well-defined and do not consider the problem of constructing the graph for a given practical task. This work proposes a methodical way of linking graph properties with the performance of a GNN solving a given task on such graph via a surrogate regression model that is trained to predict the performance of the GNN from the properties of the graph dataset. Furthermore, the GNN model hyper-parameters are optionally added as additional features of the surrogate model and it is shown that this technique can be used to solve the practical problem of hyper-parameter tuning. We experimentally evaluate the importance of graph properties as features of the surrogate model with regards to the node classification task for several common graph datasets and discuss how these results can be used for graph composition tailored to the given task. Finally, our experiments indicate a significant gain in the proposed hyper-parameter tuning method compared to the reference grid-search method.

Keywords
Graph neural networks & Model performance prediction & Hyper-parameter tuning & Node classification

1. Introduction

Across a wide variety of applications and domains, graphs emerge as a ubiquitous way of organizing data. Consequently, machine learning on graphs has, in recent years, seen an explosion in popularity, breadth and depth of both research and applications. At the same time, the underlying graph topology has, until recent works [1, 2], received much less attention. Specifically, the organization of data points into nodes and edges of a graph is usually assumed to be given, unambiguous, and well-defined, especially in works utilizing common, publicly available graph datasets that have a pre-defined topology. While in the research environment this may be considered beneficial in simplifying the comparison of various graph-based methods, in many practical applications, the mapping from data to graphs is a non-trivial and open problem. An example of such an application domain is computer network security, where a graph representation of network telemetry may contain entities of various types (users, servers, IP addresses), the edges may represent either a physical connection between two entities or a more general similarity or distance measure, both the nodes and edges may have associated with them additional features, the full dataset may be prohibitively large to efficiently process and some data points may not be available across all entities.

In our work, we investigate the problem of creating a graph representation of data such that graph neural networks (GNNs) may be used to effectively and efficiently solve a given task on the data (e.g., classifying agents in a computer network as infected by malware or clean). Given the complex and multi-faceted nature of this problem, we limit the reported research to the sub-problem of measuring and predicting the suitability of a given graph representation to a given task.

1.1. Problem formulation

To solve the problem of predicting the suitability of a given graph dataset for a particular task, we first represent the graph dataset by its *properties* (Section 2.2) that are aggregate values representing the whole graph dataset instead of individual nodes or edges. A GNN model [3] is trained to solve the task, and its performance is measured using several metrics. The main aim of this work is to extract information about the usefulness of individual graph properties from a *meta-model* that is trained on the graph dataset properties to predict the GNN performance represented by the *performance metrics*. In the reported research, we propose and evaluate such a meta-model on publicly available datasets, how-

ITAT'23: Information technologies – Applications and Theory, September 22–26, 2023, Tatranské Matliare, Slovakia
*Corresponding author
paprocha@cisco.com (P. Procházka); mimares@cisco.com (M. Mareš); marek@dedic.eu (M. Dědič)
https://dedic.eu (M. Dědič)
0000-0003-1021-8428 (M. Dědič)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

ever, it forms a useful and general tool for evaluating the suitability of graph datasets to tasks on them, which is a basic prerequisite to solving the more general problem of constructing an advantageous graph representation.

1.2. Related work

Machine learning model performance prediction is commonly used to avoid the expensive evaluation of the original model on the test set [4]. However, the problem of trust in these meta-models limits their applicability in real-world scenarios. To address this problem, the authors in [5] propose attaching prediction uncertainty to the meta-models and suggest a method for evaluating this uncertainty. In [6], the authors observe that state-of-the-art shift detection metrics (referred to as graph properties in our paper) do not generalize well across datasets, and they propose incorporating error predictors. In this paper, we address both the trust and generalization problems. The novelty of this paper lies in our use of the meta-model: firstly, for interpreting the graph properties that drive the model's performance, and secondly, for hyper-parameter tuning. To the best of our knowledge, there is no existing use of the meta-model for these purposes in the current state of the art.

Graph theory encompasses various numeric graph properties, ranging from basic ones such as the number of nodes, to more sophisticated metrics like graph curvature [1]. In this paper, we select a subset of these metrics, listed in Table 1, as features for the meta-model.

Graph Neural Networks (GNNs) [3] achieve superior performance on graph datasets. However, this performance often comes at the cost of high computational resources required for training. Additionally, the large configuration space of these models necessitates non-trivial resources for fine-tuning. Our research aims to reduce the required computational resources in two ways. Firstly, we attempt to construct a graph dataset from the source data with favorable properties for GNN execution. Secondly, the proposed hyper-parameter search aims to reduce computational resources during fine-tuning.

Shapley Additive Explanations (SHAP) [7] is a framework for explaining predictions of any model based on coalition game theory concepts introduced in [8]. An additional benefit of this framework is its ability to see whether low or high values of the input variables contribute to low/high predictions of the model. In this paper, we adopt the SHAP framework for model explanation.

1.3. Contribution

- We propose a method to identify important graph dataset properties using the meta-model.
- We present a hyper-parameter tuning method based on the meta-model.
- We experimentally verify the generalization capability of the meta-model.
- We evaluate the importance of graph properties and their impact on GNN performance.
- We experimentally validate the hyper-parameter tuning approach with very promising results.

2. Graph representation for GNN performance prediction

2.1. Notation and definitions

Consider an undirected graph $G = (V, E, \mathbf{X})$ with nodes V, edges $E \subseteq V^2$, and real-valued node features $\mathbf{X} \in \mathbb{R}^{n \times d}$, where $n = |V|$. In this work, we limit the definition of a graph task to be one of *transductive node classification*, however, the method as defined is general and can be applied to other tasks such as *inductive node classification* or *link prediction*. In the transductive setting, a task on graph G can be viewed as an assignment of *node labels* Y (belonging to one of N_C classes) to the graph. Using a model $\mathcal{M}$, a prediction $\hat{Y} = \mathcal{M}(G)$ is obtained for the task, and compared to the ground truth using a *performance metric* $\mu(\hat{Y}, Y)$.

2.2. Graph representation

Our goal is to find a set of graph dataset properties $\mathcal{P}_i$ such that those properties would only keep global-level information about the graph G and at the same time provide as much information as possible about the performance μ obtainable on G.

We offer a range of graph dataset properties (see Table 1). We categorize these properties into three types of information. Specifically, these properties can convey information regarding: 1) node attributes, 2) graph structure, 3) specified task, or any combination thereof (awareness in Table 1).

Apart from basic graph properties and well-established metrics on graphs, we consider some additional graph properties for better description. In order to define these additional non-standard properties formally, we denote $V^c \subseteq V$ the set of nodes belonging to the class c and $|V^c|$ its size. The mean attribute vector over the class c is then given as $\bar{x}_c = \frac{1}{|V^c|} \sum_{i \in V^c} x_i$, where x_i denotes the attribute vector of the corresponding node i. Finally, we define a mean squared distance between attributes in class c_1 and mean of attributes in c_2 (attribute similarity) as

$$C_A(c_1, c_2) = \frac{1}{|V^{c_1}|} \sum_{i \in V^{c_1}} (x_i - \bar{x}_{c_2})^2. \tag{1}$$

This asymmetric quantity is used to express similarity between attributes based on the task (see Table 1).

Graph Dataset Property	Awareness			Description / Definition
	Task	Structure	Attributes	
Node count	No	No	No	Number of nodes – dataset size.
Class ratio	Yes	No	No	Ratio between the number of positive and negative nodes.
Number of components	No	Yes	No	Number of connected components of the graph.
Average node degree	No	Yes	No	Average node degree in the graph.
Global assortativity	No	Yes	No	Measure of the tendency of nodes to connect with other similar nodes, rather than dissimilar nodes [9].
Attribute similarity	No	Yes	Yes	Average cosine similarity of attributes across all edges in the graph.
Attribute homophily	Yes	Yes	No	Measure of how clustered together are nodes with similar attributes [10].
Edge homophily	Yes	Yes	No	Fraction of edges connecting nodes of the same class [11].
Node homophily	Yes	Yes	No	Fraction of node neighbours having the same class as the node in question, averaged over all nodes [12].
Class homophily	Yes	Yes	No	A modification of node homophily that is invariant to the number of classes [13].
Ratio of positive nodes of degree > 1	Yes	Yes	No	Ratio of positive nodes with degree greater than one.
Fraction of positive nodes of degree > 2	Yes	Yes	No	The fraction of positive nodes with degree greater than two, out of those with degree greater than one.
Average positive node degree	Yes	Yes	No	Average node degree in the sub-graph restricted to nodes from V^1.
Relative presence of positive edges	Yes	Yes	No	Number of edges connecting positive nodes, divided by the number of edges that would be present in a theoretical clique constructed of all positive nodes.
Positive attribute similarity	Yes	No	Yes	$C_A(1,1)$ – see Equation (1).
Positive to negative attribute similarity	Yes	No	Yes	$C_A(1,0)/C_A(1,1)$ – see Equation (1).
Negative to positive attribute similarity	Yes	No	Yes	$C_A(0,1)/C_A(1,1)$ – see Equation (1).

Table 1
Graph dataset properties considered for the meta-model training.

2.3. GNN performance prediction

Based on the graph dataset properties, we consider a meta-model $\mathcal{M}_{\text{meta}}$, which makes a prediction $\hat{\mu}$ of the true performance μ based on the properties $\{\mathcal{P}_i\}$.

2.4. Multiple binary classification

To train and evaluate the meta-model, a sufficient dataset is needed. Given that the individual points in this dataset themselves correspond to graph-task pairs and models trained on them, obtaining such a dataset for the meta-model is computationally expensive. To aid with its creation, only binary classification tasks were considered, where for datasets with more than 2 classes, multiple tasks were constructed by taking one class as positive and other classes as negative, for each class in the original dataset. This procedure has its motivation in applications, where e.g. in the domain of computer security, a classifier distinguishing each particular kind of malware is a useful addition to a general malware classifier.

2.5. Measuring graph property usefulness

We train a regression meta-model on a dataset consisting of graph properties (features of the meta-model) and the corresponding GNN performance metric (label for the meta-model). If the regression model generalizes well, *we consider the graph properties that are important for the meta-model prediction to also be important for the GNN performance* on a graph with the given properties.

By evaluating the meta-model's performance on the test set and determining the important features of the meta-model (e.g., using SHAP), we propose applying the meta-model explanation to determine the impact of individual graph properties on the GNN performance. The validity of this claim is assessed through the meta-model's performance on the test set.

2.6. Hyper-parameter optimization

In order to apply the meta-model to hyper-parameter optimization, the hyper-parameter values are added to

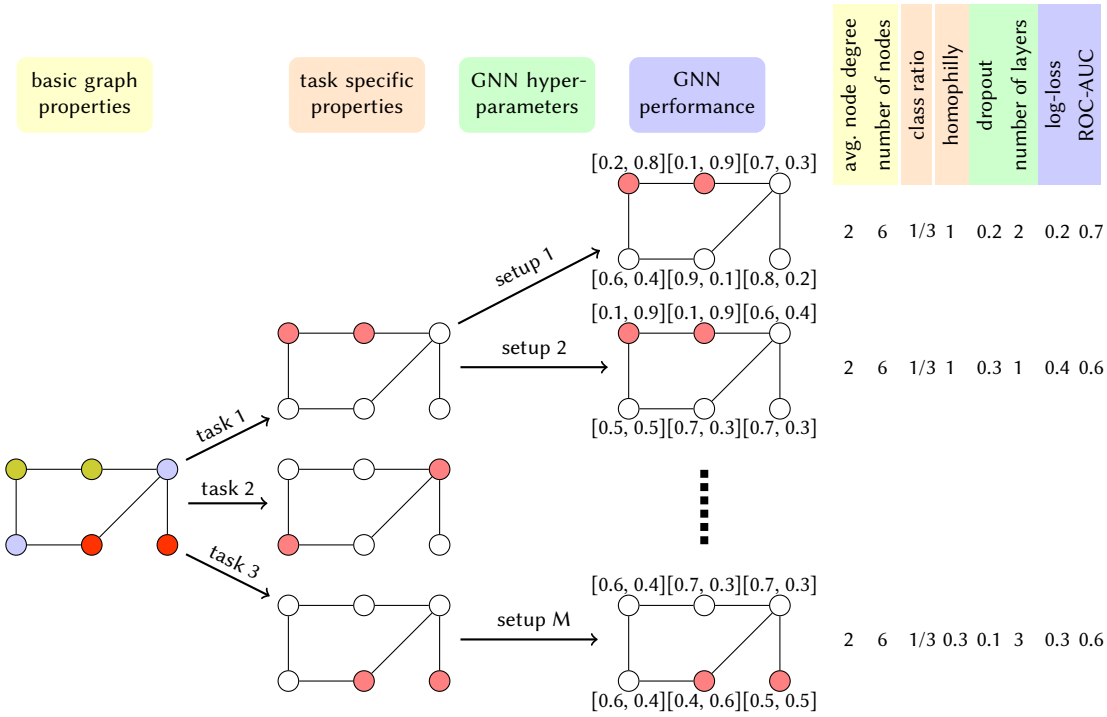

Figure 1: A schema of how the proposed dataset for the meta-model is obtained. Starting from well-established graph datasets suited for multiple-class node classification – Table 2 (left), we create a graph (task) for each label, so that each label induces a positive class for binary node classification. We apply a GNN described in Equation (2) with hyper-parameter space described in Table 3. The output of the GNN is then used for performance metric evaluation. Each datapoint is the described by general graph properties (yellow), task specific graph properties (orange) (see Table 1 for the considered properties), hyper-parameters of the GNN (green) and performance of the GNN (blue), which is then used as label.

the inputs of the meta model, making it

$$\hat{\mu} = \mathcal{M}_{\text{meta}}\left(\{\mathcal{P}_i\} \cup \{H_i\}\right)$$

where H_i are the hyper-parameters of a particular model $\mathcal{M}$ whose performance is being predicted.

Since the meta-model is trained solely on aggregated graph dataset properties and hyper-parameters, its training process is considerably simpler compared to training the full GNN on the original graph. Therefore, the suggested procedure is to find a hyper-parameter setup that maximizes the predicted meta-model performance for a given set of graph dataset properties. We experimentally evaluate this proposed method in Section 3.3.

3. Experimental evaluation

3.1. Experiment description

We consider graph datasets (Table 2) each inducing N_C binary classification tasks (see Section 2.4). We follow the design space [20] to run the GNN for each dataset.

Dataset	#Classes	#Nodes	#Edges
ArXiv [14]	40	169 343	2 315 598
Flickr [15]	7	89 250	899 756
Computers [16]	10	13 752	491 722
Pubmed [17]	3	19 717	88 648
DBLP [18]	4	17 716	105 734
Squirrel [19]	5	5 201	396 846
Cora [17]	70	19 793	126 842

Table 2
Graph datasets considered in the experimental section. Each graph induces a binary classification problem for each class.

The GNN is described as

$$\mathbf{m}^{(k)}_{\mathcal{N}_s(u)} = \text{AGGREGATE}^{(k)}\left(\{\mathbf{h}^{(k)}_v, \forall v \in \mathcal{N}_s(u)\}\right) \quad (2)$$

$$\mathbf{h}^{(k+1)}_u = \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}\left(\mathbf{h}^{(k)}_u, \mathbf{m}^{(k)}_{\mathcal{N}_s(u)}\right)\right) \quad (3)$$

with $\mathbf{h}^{(0)}_u = X_u$ being the feature vector corresponding to the node $u \in V$. The parameters of the design space are described in Table 3 and $\mathcal{N}_s(u)$ denotes the 1-hop

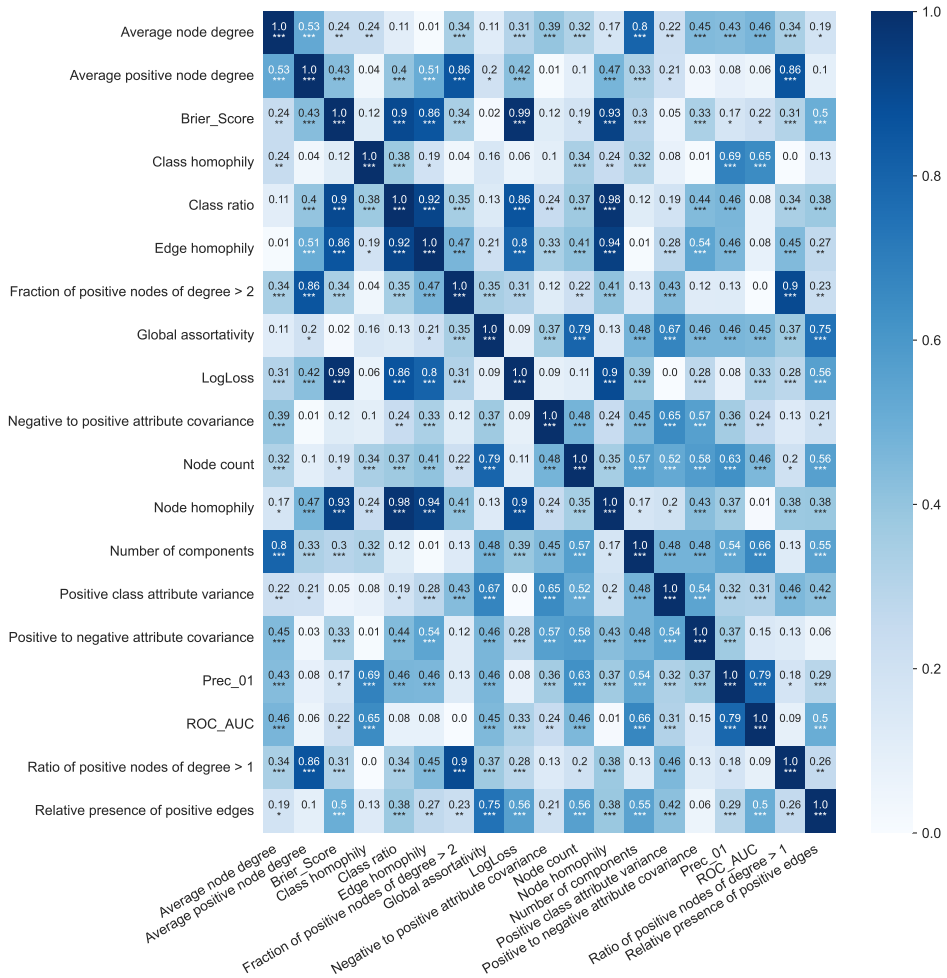

Figure 2: Spearman correlation coefficient of graph properties with the performance metrics over all datasets with associated p-value represented as $*$ if $p \leq 0.05$, $**$ if $p \leq 0.01$ and $***$ if $p \leq 0.001$.

Hyperparameter	Values
Layers	1, 2, 3
Hidden channels	16, 32, 64
Dropout probability	0, 0.3, 0.6
Activation function	ReLU, PReLU
Aggregation function	mean, max

Table 3
Design space [20] considered for the GraphSAGE architecture.

neighborhood of the node $u \in V$ in the graph G, k stands for order of GNN-layer and W^k is the trainable weight vector describing the k-th layer of the GNN.

The size of the considered design space consists of 108 unique hyper-parameter configurations and therefore $N_C \times 108$ datapoints for each dataset, that is 15 012 datapoints in total.

The GNN, along with its hyper-parameter setup, is applied to each point in the dataset for predictions in each node. Using these predictions and the true labels, we evaluate the final performance metrics on the test nodes. Specifically, we consider the following metrics:

- Log Loss.
- Brier Score – Mean squared distance between GNN prediction output and true label.
- ROC-AUC – Area under the ROC Curve.
- Prec0.1 – Precision at top 10% positive nodes based on the predicted score.
- F1 score.

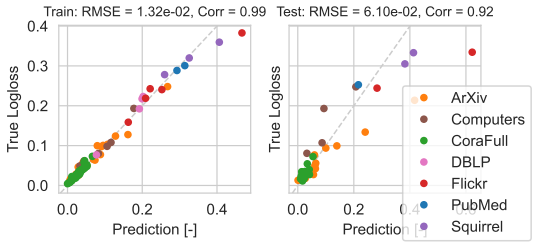

Figure 3: Performance of the meta-model predicting Log Loss of the GNN model on the original graph. The model is able to generalize reasonably for unseen tasks on the given datasets.

The tuple (basic graph properties, target class specific properties, hyper-parameters, performance measured on the test nodes) constitutes a datapoint in the final dataset (see Figure 1).

Our meta-model is a random forest regression model with 100 trees with mean squared optimisation criterion with at most 30% of features considered at each split.

3.2. GNN performance prediction based on graph properties

Given a dataset described in Section 3.1, we train a regression meta-model predicting the performance metric using the graph properties. In this experiment, we consider a selection of the best performance over all available hyper-parameter settings for each graph and target class, where only graph properties are used for training.

The dataset was split randomly to a training and testing subsets of sizes 93, respectively 46 data-points. The meta-model was optimised to minimize MSE between the model prediction and true performance of GNN.

We evaluated Spearman correlation of the considered graph properties with the target metrics within the dataset (see Figure 2). The results show very high correlation between the log loss and the Brier Score and also high correlation between the ROC AUC and precision at 10% of positive nodes. Additionally, graph properties correlate better with log loss and Brier score, indicating better performance in predicting these metrics (which is later confirmed in the experiments).

The results of our meta-model predicting the true log loss of the GNN are shown in Figure 3. We can see quite decent performance on the testing set. Although we do not use this performance directly for any task, it provides us with the important information that the meta-model does not just memorize the training set and indeed uses the graph properties to model the true performance of the GNN. Based on this generalization ability, we claim that graph properties are driving the decision of both the meta-model as well as the underlying GNN (Section 2.5).

Based on SHAP explanation of the meta-model, we evaluated how individual graph properties affect the final GNN performance (see results in Figures 4 and 5 for Log Loss and ROC AUC). We can observe that the most important graph properties differ for each task.

As expected, a higher homophily (all of its variants) contributes to better performance. Interestingly, a higher class ratio leads to better performance in ROC-AUC prediction but worsens the performance in log-loss prediction. Although these observations are very interesting and essentially answer our research question, we should also consider the limitations of these results. Firstly, their validity is conditioned by the validity of our hypothesis assuming that the explanation of the meta-model holds for the task itself. Secondly, as the graph properties are not independent of each other (see Figure 2), impact of one particular property can be reflected in importance of multiple correlated properties. We leave deeper investigation of these limitations and their impact for future work.

3.3. Hyper-parameter optimization

In this experiment, we use the dataset generation method described in Figure 1 for each graph from Table 2. We split randomly test and train sets with a ratio r so that the training set size is given by round(rN_D), where N_D is the dataset size. We provide 100 realizations of this split for each ratio r. In each realization, we train the meta-model on the training set and calculate predictions on the test set. We consider the performance based on the following hyper-parameter selection procedures:

- Reference (random search): We select the best performance on the training set plus one sample from the test set to ensure fair comparison.
- Ours: We find the hyper-parameter setup achieving the best performance prediction on the test set, evaluate the true corresponding performance, and select the best performance on the training set along with the evaluated one.
- Optimum: We select the best performance from both the test and training sets.
- Ours - Cross-datasets: Similar to "ours" method, but we consider all graph datasets except the evaluated one for training.

The mean of the resulting performance over realisations is reported in Figure 6. In addition to the aforementioned hyper-parameter selection procedures, we consider one more reference (best hyper-parameter) by selecting a single hyper-parameter setup with the best average performance over all binary tasks for each dataset, ensuring that the model does more than just learn the best setup for a specific dataset.

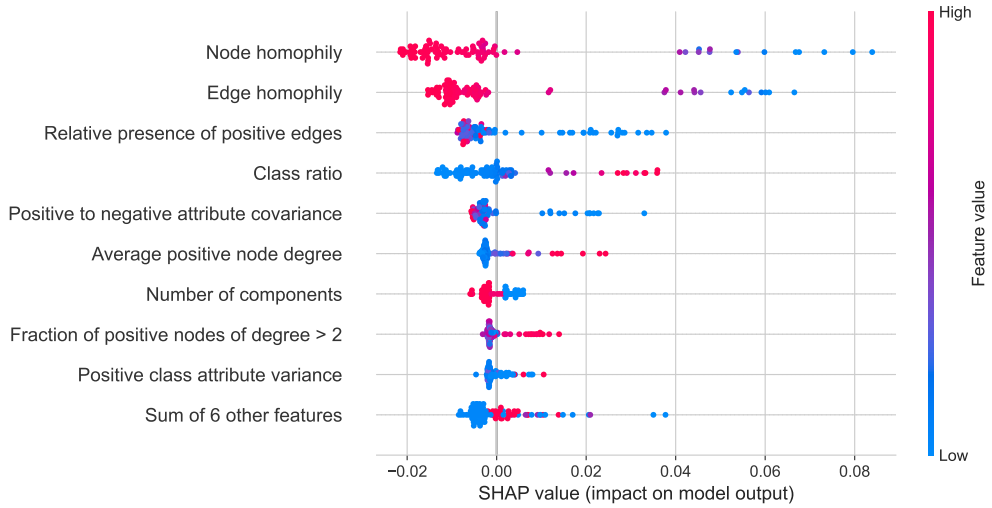

Figure 4: SHAP explanation for log loss prediction identifying node and edge homophily as the most important graph properties for the log-loss prediction. Therefore, we claim these graph properties to be the most important graph properties, when training GNN on the graph with log-loss objective. In addition, the SHAP explanation shows that the higher homophily the lower log-loss and therefore the better performance. Recall that the GNN aims to minimize log-loss.

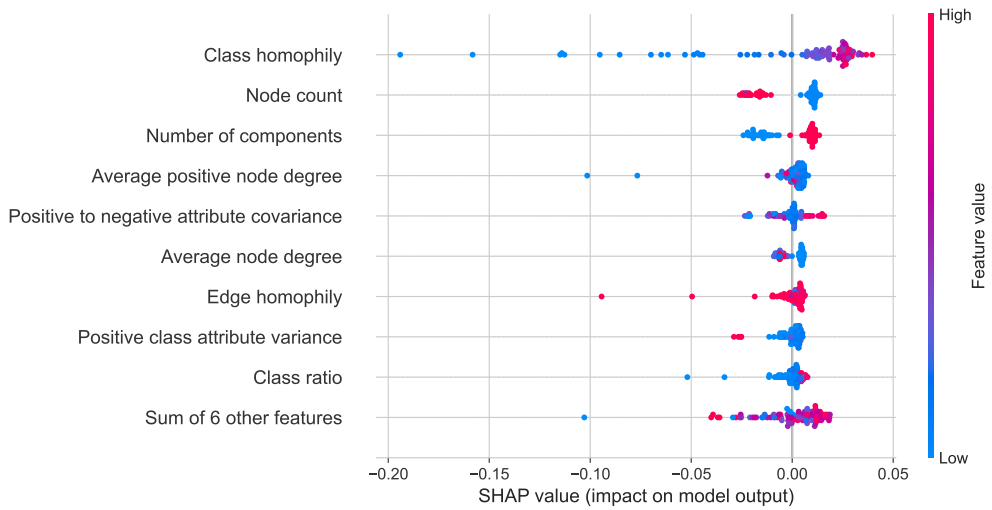

Figure 5: SHAP explanation for ROC-AUC prediction.

As we can see, the suggested method ("ours") outperforms the reference in almost all cases, resulting in a significant difference, for example, in the Cora dataset. However, the most interesting result is achieved by "ours cross-datasets" method. This method is evidently able to learn the optimal parameter setup from the graph properties since it achieves nearly optimal performance across all datasets. The comparison to the best hyper-parameter reference method ensures that the meta-model did not simply learn the global solution for all datasets.

4. Conclusion

We propose a systematic approach to linking graph properties with corresponding GNN performance using a simple meta-model. This meta-model is trained to predict the true performance based on the graph properties. We experimentally validated the generalization capability of

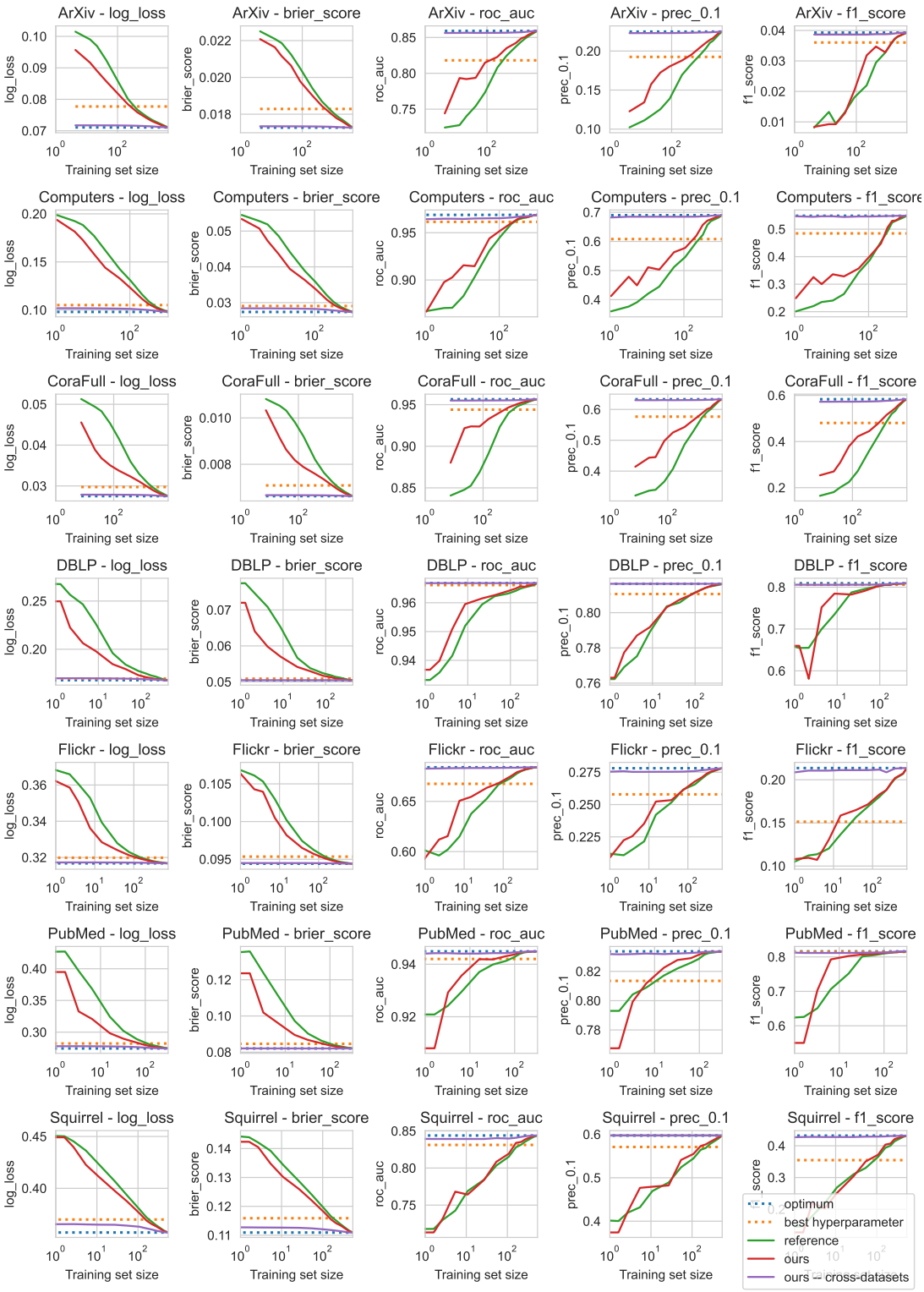

Figure 6: Comparison of reference random hyper-parameter search with the proposed solutions for considered datasets. While log-loss and Brier score (two columns from left) are to be minimised, the remaining performance metrics are to be maximised. We stress that the violet curve is very close to the (optimal) blue dotted line for almost all datasets and target metrics. It means that the proposed way of hyper-parameter tuning is able to reveal almost optimal hyper-parameter setup based only on the actual graph properties (not the whole graph itself!).

this meta-model on common datasets in the graph research community. By interpreting the meta-model's explanations, we identified graph properties that influence the meta-model's behavior and claim that this interpretation also applies to the impact on GNN performance. We evaluated these properties and found that they align with our expectations.

The meta-model predictions were also utilized to solve the hyper-parameter optimization problem. Leveraging the fact that the meta-model is computationally cheaper compared to the GNN, we demonstrated that relying on the meta-model's predictions can lead to superior performance compared to the reference random search method. Specifically, when the meta-model incorporates knowledge from other graph datasets, we achieved almost optimal performance even without seeing any data points from the target dataset. This indicates that the model is capable of learning solely from the graph properties.

The proposed hyper-parameter search method can potentially be extended beyond graph datasets, where we train the meta-model on suitable properties of the given dataset. However, in this paper, we only scratched the surface of this topic, which warrants further research and an in-depth survey of available works on hyper-parameter optimization. In the context of this paper, we view it as a validation of the concept of learning a meta-model based on graph properties. Nonetheless, the presented results offer a practical approach to solving the hyper-parameter search problem for graph datasets.

References

[1] J. Topping, et al., Understanding over-squashing and bottlenecks on graphs via curvature, in: The Tenth International Conference on Learning Representations, 2021.

[2] P. Veličković, Geometric Deep Learning - Grids, Groups, Graphs, Geodesics, and Gauges, 2021.

[3] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Advances in neural information processing systems 30 (2017).

[4] S. B. Guerra, R. B. Prudêncio, T. B. Ludermir, Predicting the performance of learning algorithms using support vector machines as meta-regressors, in: Artificial Neural Networks-ICANN 2008: 18th International Conference, Prague, Czech Republic, September 3-6, 2008, Proceedings, Part I 18, Springer, 2008, pp. 523–532.

[5] B. Elder, et al., Learning Prediction Intervals for Model Performance, Proceedings of the AAAI Conference on Artificial Intelligence 35 (2021) 7305–7313. Number: 8.

[6] S. Maggio, V. Bouvier, L. Dreyfus-Schmidt, Performance Prediction Under Dataset Shift, in: 2022 26th International Conference on Pattern Recognition (ICPR), 2022, pp. 2466–2474. ISSN: 2831-7475.

[7] S. M. Lundberg, S.-I. Lee, A Unified Approach to Interpreting Model Predictions, in: Advances in Neural Information Processing Systems, volume 30, Curran Associates, Inc., 2017.

[8] L. S. Shapley, Notes on the N-Person Game — II: The Value of an N-Person Game, Technical Report, RAND Corporation, 1951.

[9] M. E. J. Newman, Mixing patterns in networks, Physical Review E 67 (2003) 026126. Publisher: American Physical Society.

[10] L. Yang, et al., Diverse Message Passing for Attribute with Heterophily, in: Advances in Neural Information Processing Systems, volume 34, Curran Associates, Inc., 2021, pp. 4751–4763.

[11] J. Zhu, et al., Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs, in: Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., online, 2020, pp. 7793–7804.

[12] H. Pei, et al., Geom-GCN: Geometric Graph Convolutional Networks, 2020. ArXiv:2002.05287 [cs, stat].

[13] D. Lim, et al., Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods, in: Advances in Neural Information Processing Systems, volume 34, Curran Associates, Inc., online, 2021, pp. 20887–20902.

[14] W. Hu, et al., Open Graph Benchmark: Datasets for Machine Learning on Graphs, 2021. ArXiv:2005.00687 [cs, stat].

[15] H. Zeng, et al., GraphSAINT: Graph Sampling Based Inductive Learning Method, in: International Conference on Learning Representations, 2019.

[16] O. Shchur, et al., Pitfalls of Graph Neural Network Evaluation, 2019. ArXiv:1811.05868 [cs, stat].

[17] Z. Yang, W. Cohen, R. Salakhudinov, Revisiting Semi-Supervised Learning with Graph Embeddings, in: Proceedings of The 33rd International Conference on Machine Learning, PMLR, New York, NY, USA, 2016, pp. 40–48.

[18] A. Bojchevski, S. Günnemann, Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking, in: 6th International Conference on Learning Representations, 2018.

[19] B. Rozemberczki, C. Allen, R. Sarkar, Multi-Scale attributed node embedding, Journal of Complex Networks 9 (2021) cnab014.

[20] J. You, Z. Ying, J. Leskovec, Design Space for Graph Neural Networks, in: Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 17009–17021.