[RE] Adaptive Universal Generalized PageRank Graph Neural Network

Leon Schlecht 10014720
Kevin Schumann 10014733
Paul Heinemeyer 10016184

Reproducibility Summary

Scope of Reproducibility

The proposed GNN architecture Generalized PageRank (GPR) claims to significantly outperform existing GNN architectures in performance on homophilic and heterophilic synthetic and benchmark datatsets. GPR does so by adaptively learning the weights and therefore optimizing node feature and topological information extraction.

Methodology

The re-implementation of the model was based on the original code of the authors, and only the syntax was modified slightly. We have kept the training methods and dataset processing unchanged. Function methods that were not called during run-time, like the *resetParameters* method, were removed. Therefore the main resource was the author's paper as well as the corresponding repository on Github [3] The experiments were conducted on a GTX 1070ti, a Ryzen 7 3700x, and 16 GB of RAM. A total of 15 days were available for the reproduction of this very paper, including understanding, implementing and running the implementation. For this reason, no hyperparameter search was performed, as this would have exceeded the given time frame. However, we made use of the listed hyperparameter of the GPRGNN model.

Results

Unlike in the paper, GPR-GNN does not outperform all the test methods on real-world benchmark datasets. This is the case even though we made use of the mentioned GPR-GNN hyperparameters for each dataset and did not tune the other models. For homophily datasets, our GPR-GNN performed up to 5.26% worse in accuracy than in the paper. On heterophily datasets, the same can be said but up to 11.69%. When outperformed it was done so by 3-8% in accuracy.

Similar patterns were observed on the cSBM datasets with $\phi \in \{-1, -0.75, ..., 1\}$ for a dense split as in the original paper. However, this did not apply to a sparse split on the data. In this case, GPR-GNN performed significantly worse for $\phi \leq 0.5$ than in the paper.

We believe that a hyperparameter search for each model and dataset is necessary to fully reproduce the results.

What was easy

The authors provided several scripts to generate cSBM data, running the experiments and code for most of the models. Only slight modifications and extensions were necessary to reproduce and simplify the experiments. Mostly improvements on logging were made.

What was difficult

Essential information about the architecture and the hyperparameters of models that were not implemented by the authors, were spread in the paper, making them hard to find.

Initially, a hyperparameter search was planned. However, an initial run showed that this was not feasible for the given time frame on our hardware.

Furthermore, GraphSAGE could not be reproduced as the sampling ratios were not mentioned in the paper.

Communication with original authors

Since the amount of time for the reproduction was quite low there was no contact with the original authors. Other than that, the most important aspects about the reproducability of their GPR-GNN model were self-explanatory, thanks to a small basic installation guide on their GitHub[3].

1 Introduction

In the paper, [4] the authors proposed GPR-GNN, a Graph Neural Network (GNN) architecture that can handle both homophilic and heterophilic graphs well. The authors claim to significantly outperform state-of-the-art architectures with GPR-GNN on both homophilic and heterophilic benchmark datasets, as well as synthetically generated datasets using contextual Stochastic Block Model (cSBM). By using Generalized PageRank (GPR), the authors address two major problems of GNNs. First of all, most modern architectures only perform well on homophilic graphs, since they aggregate the features of the k-hop neighbors. And second of all, existing GNNs approaches tend to be prone to oversmoothing - the process where feature information quickly gets rendered useless, since the network quickly aggregates feature information from the whole graph. In order to circumvent this, the existing approaches usually require the underlying architectures to be shallow. GPR-GNN on the other hand does not need the network to be shallow. GPR-GNN learns the node label pattern of disparate classes by optimizing the GPR weights and thereby preventing feature oversmoothing. Their architecture adaptively learns the GPR weights in order to optimize information extraction of node features and neighborhood information, irrelevant of whether the datasets are heterophilic or homophilic. The learned weights automatically adjust to the graph structure, in order to perform well on datasets that are usually difficult to handle for other GNN models.

In the following chapters, we present our gained insights from replicating GPR-GNN and investigate the reproduced results.

2 Scope of reproducibility

To verify the main claims presented in the paper we focus on the following target questions:

- How well does GPR-GNN perform compared to already existing GNN models on benchmark and synthetic datasets?
- How much do our reproduced results differ from the authors' results?
- How well does GPR-GNN perform on heterophilic compared to homophilic datasets?

3 Methodology

In order to reproduce the provided results, we made use of the publicly accessible source code [3] of the paper. As not every model was implemented in said source code, we added these rather simple models (MLPNet, SGCNet) to the source code. Furthermore, we made use of the torch_geometric.nn.Sequential container to define GNN models. This was done for every model. Function methods that were not called during run-time, like the *resetParameters* method, were removed. Methods that were responsible for training, generating, or processing datasets were kept mostly as is. Only the functionality to write results to a csv, which later helped data evaluation, was added.

The experiments were conducted on a GTX 1070ti, a Ryzen 7 3700x, and 16 GB of RAM. All the experiments were conducted like in the paper, except that no hyperparameter search was done beforehand. This is due to the fact that a test run showed that performing a hyperparameter search would have greatly exceeded the given time frame of 15 days. As the paper lists GPR-GNN hyperparameters for each real-world dataset, we used those in these settings.

GraphSage and GeomGCN were not tested. No sample ratios were mentioned in the paper for Graph SAGE. GeomGCN was not tried to reproduce, because it would have required essentially the same effort to reproduce and setup like the other things - and this was neither the purpose nor the main goal of this reproducibility report.

3.1 Model descriptions

We have tested the following models: GRPGNN, APPNP, MLP, SGC, GCN, GAT, JKNet, and GCN-Cheby. The number of trainable parameters can be seen in Table 1. SGC had the least (4002) number of trainable parameters while GAT (1058495) had the most. All models were trained from the ground up, no pre-training was used.

3.2 Datasets

Evaluation of the mentioned models was done on the same real-world datasets as mentioned in the paper. For the homophilic datasets Cora, Citeseer, PubMed, Computers, and Photo sparse splitting (2.5% training, 2.5% validation) was used. Dense splitting (60% train, 20% validation) was done for the heterophilic datasets Chamelon, Squirrel, Actor, Texas and Cornell. Properties and statistics of the mentioned datasets can be seen in Table 2.

Model	#parameters
GPRGNN	128205
APPNP [5]	128194
MLP	128194
SGC [8]	4002
GCN[2]	128194
GAT [7]	1058496
JKNet [9]	45187
GCN-Cheby [1]	128162

Table 1: List of the evaluated models with their trainable parameters respectively.

Dataset	Cora	Citeseer	PubMed	Computers	Photo	Chameleon	Squirrel	Actor	Texas	Cornell
Classes	7	6	5	10	8	5	5	5	5	5
Features	1433	3703	500	767	745	2325	2089	932	1703	1703
Nodes	2708	3327	19717	13752	7650	2277	5201	7600	183	183
Edges	5278	4552	44324	245861	119081	31371	198353	26659	279	277
H(G)	0.656	0.578	0.644	0.272	0.459	0.024	0.055	0.008	0.016	0.137

Table 2: Datasets statistics and properties of the used real-world datasets

Synthetic graphs were generated using a contextual Stochastic Block Model (cSBM). All graphs have 5000 nodes, 2000 features, an average node degree of 5, and an ϵ of 3.25. The factor ϕ was varied from -1 to 1, in steps of 0.25 resulting in 10 synthetic datasets. ϕ controls the amount of information within the graph topology. $\phi = 0$ means that information is only encoded in the nodes itself. For $|\phi| = 1$ only the graph structure is informative. $\phi = 1$ will result in a strongly homophilic graph, while $\phi = -1$ will result in a strongly heterophilic one [4]. For more information on how cSBM datasets work, see the original paper[4]

3.3 Hyperparameters

The original paper performed a hyperparameter search over the following parameters: GPR-GNN used random walk path lengths of K = 10 and a 2-layer MLP with 64 units. Alpha values for $\{0.1, 0.2, 0.5, 0.9\}$ For APPNP the search for optimal alpha was done in $\{0.1, 0.2, 0.5, 0.9\}$. For all the other nets, according to the authors, they optimized the other hyperparameters and ended up using learning rates $\{0.002, 0.01, 0.05\}$ and weight decay values $\{0.0, 0.0005\}$.

Due to a time constraint, we could not conduct the hyperparameter search ourselves. However, we made use of the provided GPR-GNN parameters for each real-world dataset. These can be seen in the file Reproduce_GPRGNN.sh of the original repository [3]. Otherwise, the default values of the models have been used.

The default values for each parameter are shown in Table 3

Parameter	Default value
K	10
Head	8
Alpha	0.1
Output heads	1
Weight decay	0.005
Early stopping	200
Learning rate	0.002
Number of epochs	1000
Dropout probability	0.5

Table 3: Default (hyper)parameters for every model.

3.4 Experimental setup and code

To reproduce the result of the paper, two experiments, 1 on the synthetic and 1 on the real-world datasets, were conducted.

All models listed Table 1 were first evaluated against every cSBM dataset. We generated all 10 synthetic datasets by modifying the create_cSBM_dataset.sh file in the original repository [3] for every ϕ value. Afterward, each model was run against each dataset twice (1 dense and 1 sparse split) making use of the original training routine. This was done 100 times. For the second experiment, each model was tested 100 trials against the real-world datasets using the data splits as insubsection 3.2.

Our repository [6] provides helper scripts (generate_all_datasets.sh, reproduce_fig2.sh, reproduce_no_grid_100_exp.sh) for both the generation of the datasets and running every model against them.

During both experiments, we logged the accuracy, standard deviation on the test split over all trials for each model, dataset, and split type The standard deviation, the number of trials, and mean accuracy were used to calculate the margin of error for each combination.

3.5 Computational requirements

Both experiments were run on the same machine utilizing a GTX 1070ti, a Ryzen 7 3700x, and 16 GB of RAM. Roughly 4 days were needed to conduct both experiments. As said in subsection 3.3 we tried to perform a hyperparameter search. After 1 day of additional training, we concluded that this was simply not feasible on our hardware.

GCN-Cheby needed roughly 400 seconds for each iteration and therefore was the slowest network we have evaluated by far. On the contrary, MLP and SGC were only needing 3 to 5 seconds.

4 Results

In the original paper, GPR-GNN performed better on all tested real-world datasets compared to the other evaluated models except Citeseer. Furthermore, GPR-GNN was only beaten by GCN-Cheby on the synthetic cSBM dataset ($\phi = -0.25$) using the spare split setting. Generally, the paper showed that GPR-GNN generalized to heterophilic graphs and succeeds in acting as a universal learner.

In our evaluation, GPR-GNN still seems to be a good universal learner on real-world datasets. However, it did not outperform every other state-of-the-art method. For instance, GPR-GNN was beaten by 7.87% in accuracy by an MLP model. Nevertheless, it was able to beat every other method by 1.84% on the chameleon dataset. The results of each model on each dataset can be seen in Table 4.

GPR-GNN has shown to be along with the top 3 performers on the cSBM datasets utilizing a dense split. The same can not be said for the sparse setting, where this novel method starts off as the lowest performer on the strong heterophilic dataset ($\phi = -1$), while ascending in rank the more homophilic the cSBM dataset is. It ends up as the strongest contender for $\phi = 1$. More info is displayed in Table 6 and Table 7.

4.1 Results reproducing original paper

Table 4 shows the mean accuracy over 100 trials of each model on each evaluated dataset. Note that the rows for SAGE and GeomGCN are left empty as we did not evaluate those methods due to the reasons provided in section 3. In the original paper, GPR-GNN outperforms every other tested method. Although it is a top contender for each dataset, GPR-GNN was outperformed in 5 out of 10 cases in our experiment.

For an easier comparison between our and the original results, the difference in accuracy is shown in Table 5. Except for SGC on the Cora and Citeseer dataset, all models performed worse in our evaluation. Excluding GPR-GNN these differences can be explained by the fact that we did not search for hyperparameters for each model on each dataset. A tuned model is expected to perform better and therefore minimize the difference between their and our results. APPNP scored worse on the texas and Cornell dataset by a large amount. How much hyperparameters influence the performance and hence the difference in score, is unknown.

As per subsection 3.4, we evaluated all the methods and GPR-GNN using cSBM datasets with $\phi \in \{-1, -0.75, ..., 1\}$. The summarized results can be found Figure 1. When comparing our results of the dense setting to the original ones, clear similarities can be found. Similar to the papers results GPR-GNN outperforms every other baselines for $\phi < 0.0$, except for $\phi = -0.25$. When the graph information is low ($\phi \in 0, -0.25$) GNNs did perform worse than a simple

MLP, which can also be seen in the original figure. However, GPR-GNN is one of the more robust methods for this setting. APPNP performs the worst on strong heterophilic graphs.

Generally speaking, we observed similar results compared to the original results. Once again, the difference can be explained by not performing a hyperparameter search for each model and dataset combination.

In the sparse settings, our results differ from the obtained result in the paper. GNN methods perform way worse (up to 40% in mean accuracy) for heterophilic datasets. For $\phi = -0.75$ most model peak (except MLP, APPNP) at a local maximum, which can also be observed in the original graph. One could argue that both figures have similar shapes. This time the peak at $\phi = -0.75$ just happens on lower local maxima. Whether a hyperparameter search influences the models' performance on a heterophilic dataset by such a large amount is to be determined.

The detailed results for both the dense and sparse settings can be found Table 6 and Table 7.

model	cora	citeseer	pubmed	computers	photo	chameleon	film	squirrel	texas	cornell
GPRGNN	75.59 $\pm$ 0.23	62.37 $\pm$ 0.31	83.01 $\pm$ 0.07	**81.32** $\pm$ **0.63**	**89.81** $\pm$ **0.21**	**60.57** $\pm$ **0.52**	31.84 $\pm$ 0.29	**41.37** $\pm$ **0.37**	**84.43** $\pm$ **1.07**	79.67 $\pm$ 1.27
APPNP	**79.37** $\pm$ **0.32**	**68.33** $\pm$ **0.34**	83.76 $\pm$ 0.09	67.45 $\pm$ 0.51	84.47 $\pm$ 0.44	39.23 $\pm$ 0.74	33.58 $\pm$ 0.24	21.65 $\pm$ 0.22	13.93 $\pm$ 1.79	20.82 $\pm$ 4.32
MLP	49.49 $\pm$ 0.5	51.52 $\pm$ 0.58	80.3 $\pm$ 0.13	66.98 $\pm$ 0.33	75.66 $\pm$ 0.38	48.12 $\pm$ 0.4	**39.18** $\pm$ **0.16**	29.56 $\pm$ 0.35	82.95 $\pm$ 0.77	**87.54** $\pm$ **0.72**
SGC	74.58 $\pm$ 0.35	68.02 $\pm$ 0.22	77.81 $\pm$ 0.24	62.23 $\pm$ 0.58	65.05 $\pm$ 0.76	42.34 $\pm$ 0.67	29.97 $\pm$ 0.16	30.84 $\pm$ 0.43	61.15 $\pm$ 1.29	41.8 $\pm$ 0.98
GCN	75.02 $\pm$ 0.23	67.31 $\pm$ 0.37	**83.83** $\pm$ **0.09**	69.89 $\pm$ 0.62	81.16 $\pm$ 0.65	58.73 $\pm$ 0.36	30.56 $\pm$ 0.18	34.97 $\pm$ 0.3	66.89 $\pm$ 1.11	57.54 $\pm$ 1.34
GAT	77.3 $\pm$ 0.3	67.16 $\pm$ 0.48	83.18 $\pm$ 0.09	66.4 $\pm$ 1.22	80.99 $\pm$ 0.69	55.43 $\pm$ 0.43	32.56 $\pm$ 0.42	25.52 $\pm$ 0.32	73.77 $\pm$ 1.08	74.26 $\pm$ 0.92
SAGE	-	-	-	-	-	-	-	-	-	-
JKNet	72.69 $\pm$ 0.7	60.01 $\pm$ 0.89	82.71 $\pm$ 0.11	50.31 $\pm$ 2.09	73.94 $\pm$ 2.09	51.53 $\pm$ 0.42	27.62 $\pm$ 0.27	37.82 $\pm$ 0.26	66.23 $\pm$ 3.63	45.57 $\pm$ 3.24
ChebNet	70.66 $\pm$ 0.46	64.28 $\pm$ 0.39	83.75 $\pm$ 0.06	79.05 $\pm$ 0.3	87.35 $\pm$ 0.24	57.79 $\pm$ 0.47	37.7 $\pm$ 0.19	36.18 $\pm$ 0.3	81.31 $\pm$ 0.73	76.89 $\pm$ 1.2
GeomGCN	-	-	-	-	-	-	-	-	-	-

Table 4: Results on real-world dataset: Shows the mean accuracy (%) and 95% confidence interval. Best results for a given dataset are shown in bold.

model	cora	citeseer	pubmed	computers	photo	chameleon	film	squirrel	texas	cornell
GPRGNN	-3.92	-5.26	-2.06	-1.58	-2.12	-6.91	-7.49	-8.56	-8.49	-11.69
APPNP	-0.04	-0.26	-1.26	-14.54	-6.64	-12.68	-5.28	-13.12	-77.25	-70.98
MLP	-0.85	-1.36	-0.27	-3.5	-3.03	1.4	0.6	-1.72	-9.31	-3.82
SGC	3.77	9.04	-4.28	-14.04	-18.75	-20.68	0.58	-12.3	5.97	-6.00
GCN	-0.19	0.01	-0.44	-12.63	-9.38	-2.23	-0.03	-10.69	-8.27	-9.18
GAT	0.6	-0.04	-0.1	-15.55	-9.1	-8.47	-3.42	-17.2	-5.1	-1.74
SAGE	-	-	-	-	-	-	-	-	-	-
JKNet	-0.53	-0.84	-0.2	-27.49	-13.76	-11.39	-5.79	-6.9	-9.3	-21.16
GCN-Cheby	-0.73	-1.39	-0.08	-3.36	-2.74	-2.17	-0.32	-4.49	-4.69	-8.44
GeomGCN	-	-	-	-	-	-	-	-	-	-

Table 5: Difference between the mean accuracy of our and the original results. Negative indicates denote that our model performed worse than the original.

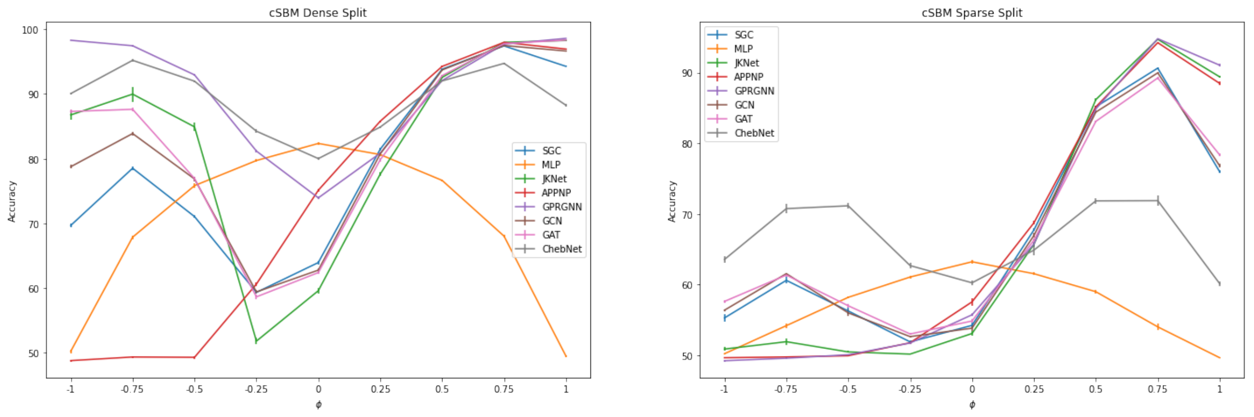

Figure 1: Reproduced accuracy scores on dense and sparse cSBM datasets for each model and different ϕ

model	$\phi(-1)$	$\phi(-0.75)$	$\phi(-0.5)$	$\phi(-0.25)$	$\phi(0)$	$\phi(0.25)$	$\phi(0.5)$	$\phi(0.75)$	$\phi(1)$
GPRGNN	**98.29 $\pm$ 0.06**	**97.44 $\pm$ 0.09**	**92.94 $\pm$ 0.16**	81.17 $\pm$ 0.21	73.94 $\pm$ 0.17	80.85 $\pm$ 0.14	92.07 $\pm$ 0.17	97.69 $\pm$ 0.1	**98.58 $\pm$ 0.06**
APPNP	48.8 $\pm$ 0.12	49.36 $\pm$ 0.12	49.31 $\pm$ 0.2	60.63 $\pm$ 0.32	75.12 $\pm$ 0.27	**85.78 $\pm$ 0.16**	**94.25 $\pm$ 0.1**	**97.97 $\pm$ 0.09**	96.93 $\pm$ 0.09
MLP	50.23 $\pm$ 0.28	67.87 $\pm$ 0.28	75.83 $\pm$ 0.32	79.71 $\pm$ 0.23	**82.35 $\pm$ 0.2**	80.63 $\pm$ 0.25	76.64 $\pm$ 0.2	68.03 $\pm$ 0.28	49.5 $\pm$ 0.16
SGC	69.68 $\pm$ 0.29	78.52 $\pm$ 0.34	71.05 $\pm$ 0.21	59.34 $\pm$ 0.25	63.95 $\pm$ 0.27	81.46 $\pm$ 0.24	93.82 $\pm$ 0.18	97.42 $\pm$ 0.14	94.26 $\pm$ 0.08
GCN	78.75 $\pm$ 0.27	83.88 $\pm$ 0.26	76.86 $\pm$ 0.34	59.41 $\pm$ 0.27	62.78 $\pm$ 0.29	80.73 $\pm$ 0.22	93.7 $\pm$ 0.13	97.48 $\pm$ 0.09	96.63 $\pm$ 0.13
GAT	87.29 $\pm$ 0.25	87.62 $\pm$ 0.33	76.93 $\pm$ 0.25	58.65 $\pm$ 0.36	62.4 $\pm$ 0.21	79.83 $\pm$ 0.26	92.83 $\pm$ 0.18	97.78 $\pm$ 0.05	98.31 $\pm$ 0.08
JKNet	86.75 $\pm$ 0.76	89.97 $\pm$ 1.14	84.91 $\pm$ 0.6	51.81 $\pm$ 0.52	59.6 $\pm$ 0.4	77.61 $\pm$ 0.3	92.56 $\pm$ 0.12	**97.97 $\pm$ 0.06**	98.26 $\pm$ 0.07
ChebNet	90.05 $\pm$ 0.2	95.19 $\pm$ 0.15	91.97 $\pm$ 0.17	**84.24 $\pm$ 0.26**	80.01 $\pm$ 0.27	84.88 $\pm$ 0.17	92.01 $\pm$ 0.14	94.72 $\pm$ 0.15	88.25 $\pm$ 0.26

Table 6: Results for cSBM, dense splitting. Bold values indicate the best-obtained results for the tested models.

model	$\phi(-1)$	$\phi(-0.75)$	$\phi(-0.5)$	$\phi(-0.25)$	$\phi(0)$	$\phi(0.25)$	$\phi(0.5)$	$\phi(0.75)$	$\phi(1)$
GPRGNN	49.18 $\pm$ 0.09	49.54 $\pm$ 0.09	50.03 $\pm$ 0.08	51.69 $\pm$ 0.15	55.69 $\pm$ 0.16	65.71 $\pm$ 0.26	84.85 $\pm$ 0.19	**94.8 $\pm$ 0.1**	**91.05 $\pm$ 0.17**
APPNP	49.61 $\pm$ 0.07	49.72 $\pm$ 0.06	49.89 $\pm$ 0.08	51.73 $\pm$ 0.23	57.52 $\pm$ 0.52	**68.72 $\pm$ 0.29**	85.14 $\pm$ 0.14	94.24 $\pm$ 0.09	88.47 $\pm$ 0.24
MLP	50.17 $\pm$ 0.11	54.15 $\pm$ 0.36	58.16 $\pm$ 0.19	61.04 $\pm$ 0.21	**63.2 $\pm$ 0.21**	61.51 $\pm$ 0.19	58.96 $\pm$ 0.26	53.98 $\pm$ 0.46	49.61 $\pm$ 0.11
SGC	55.24 $\pm$ 0.54	60.57 $\pm$ 0.44	56.23 $\pm$ 0.49	51.85 $\pm$ 0.2	54.19 $\pm$ 0.36	67.72 $\pm$ 0.18	85.15 $\pm$ 0.15	90.63 $\pm$ 0.13	75.96 $\pm$ 0.24
GCN	56.34 $\pm$ 0.15	61.52 $\pm$ 0.1	55.96 $\pm$ 0.43	52.6 $\pm$ 0.19	53.79 $\pm$ 0.41	66.93 $\pm$ 0.33	84.4 $\pm$ 0.21	89.96 $\pm$ 0.14	76.8 $\pm$ 0.27
GAT	57.58 $\pm$ 0.17	61.32 $\pm$ 0.21	56.97 $\pm$ 0.2	52.97 $\pm$ 0.13	54.81 $\pm$ 0.26	66.26 $\pm$ 0.18	83.09 $\pm$ 0.15	89.24 $\pm$ 0.21	78.34 $\pm$ 0.18
JKNet	50.83 $\pm$ 0.27	51.88 $\pm$ 0.52	50.43 $\pm$ 0.12	50.12 $\pm$ 0.09	53.04 $\pm$ 0.32	65.67 $\pm$ 0.31	**86.17 $\pm$ 0.14**	94.72 $\pm$ 0.07	89.4 $\pm$ 0.11
ChebNet	**63.52 $\pm$ 0.43**	**70.73 $\pm$ 0.62**	**71.13 $\pm$ 0.37**	**62.66 $\pm$ 0.36**	60.22 $\pm$ 0.39	64.85 $\pm$ 0.7	71.83 $\pm$ 0.43	71.86 $\pm$ 0.69	60.11 $\pm$ 0.35

Table 7: Results for cSBM, sparse splitting. Bold values indicate the best-obtained results for the tested models.

5 Discussion

We have evaluated baselines models against the novel GPR-GNN method on both real-world and synthetic data. Both the dense and sparse settings are covered by our evaluation. We did not perform hyperparameter search on every dataset and baseline combination, due to the time constraint of the report. However, this was explicitly mentioned by the researchers in order to fully reproduce their results.

Our results show clear similarities to the ones in the paper that was to be reproduced. While they are not the same (especially for cSBM dataset while doing sparse splits), we assume that performing a hyperparameter search as clearly written by the authors would have gotten us closer to the results of the paper as we argued in subsection 4.1. To test this assumption we propose doing another reproduction. This time tuning of the hyperparameters shall be performed. If a similar time constraint is given, the number of experiments, in this case 100, or the hyperparameter space could be reduced.

In general, GPR-GNN seems to be a strong universal, more robust learner on graphs as can be on real-world datasets (see Table 1).

5.1 What was easy

The authors aided the reproducibility by providing the source code [3] for most of their models, generation of cSBM datasets, and their training routine. While the source code is commented in a sparse manner, it is still easy to follow given apriori knowledge with PyTorch and PyTorch Geometric.

Furthermore, a shell script was provided that reproduced the result of the GPRGNN model on real-world datasets. This script contains the already tuned hyperparameters for each dataset which was determined by grid search beforehand. Furthermore, it was easy to adapt the said script to other models.

While the source code of the grid search and hyperparameters of the other models were not published, such program code can be easily implemented especially as the search space of each tuned parameter was mentioned in the paper.

Extending the training script to log the models' results to a csv was also easily addable. This, later on, helped to evaluate each model. Moreover, we were able to export each dataframe to a latex table directly.

5.2 What was difficult

The implementation of some models were quite difficult, because some essential information about the architecture and the hyperparameters were either spread in the paper, or not mentioned at all.

Initially, a hyperparameter search was planned, but an initial run showed that this was not feasible for the given time frame on our hardware. Even trying to split the calculation between the machine and colab (free version) did not reduce the run time by a significant amount. Colab also performed significantly worse than the mentioned machine.

Furthermore, GraphSAGE could not be reproduced as the sampling ratios were not mentioned in the paper.

5.3 Communication with original authors

As already mentioned in the summary - there was no additional contact with the original authors. Setting up the most essential parts necessary for reproducing was not difficult at all and therefore no additional contact with the authors was needed. It might have been helpful to contact the authors to gain specific information about how they used GeomGCN and what sampling ratios they had used for GraphSAGE, as well as their hyperparameters for their models. However, due to the limited time available for the project and the general purpose of reproducing the results for the GPR-GNN model, we decided against contacting the authors.

References

[1] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph neural networks with convolutional arma filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2021.

[2] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.

[3] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Original github repository.

[4] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network, 2021.

[5] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank, 2019.

[6] P.H K.S, L.S. Reproduction github repository, 2021.

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

[8] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr. au2, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks, 2019.

[9] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018.