

TECHNICAL UNIVERSITY BERLIN

Electrical Engineering and Computer Science

Master Thesis

Understanding Explainability Methods for Graph Neural Networks

by

Ramona Bendias

Supervisors:

Dr. Marina M.-C. Höhne

Prof. Dr. Klaus-Robert Müller

October 13, 2022



Declaration of Authorship

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, October 13, 2022

.....
Unterschrift

Zusammenfassung

Da Graph Neuronale Netze (GNN) zunehmend in sensiblen Bereichen eingesetzt werden, sind in den letzten Jahren zahlreiche graphspezifische Methoden vorgeschlagen worden, die bei der Interpretation der Modelle helfen sollen. Darüber hinaus ist anzumerken, dass ebenfalls allgemeine Machine Learning Erklärungsansätze für die Verwendung auf GNNs ausgeweitet werden können. Um den Einsatz allgemeiner Erklärungstechniken zu erleichtern, haben wir eine Schnittstelle entwickelt, die den Gebrauch von Captum-Erklärungsmethoden auf PyG GNNs ohne weiteren Implementierungsaufwand ermöglicht. Um sowohl allgemeine als auch graphspezifische Erklärungstechniken zuverlässig einsetzen zu können, müssen sie ausgiebig getestet werden. Bislang gibt es jedoch keine standardisierten Verfahren hierfür. Daher entwickeln wir in dieser Arbeit Benchmarks, bei denen wir Datensätze durch mathematische Regeln erstellen. Anhand der Regeln leiten wir Ground Truth-Erklärungen ab. Wir untersuchen verschiedene Erklärungsmethoden für GNNs in Bezug auf die Grapheingaben: Kanten, Knoten und Knotenmerkmale. Die Verfahren GNNExplainer, PGExplainer, Guided Backpropagation, GradientShap, Lime, Integrated Gradients (IG) und Saliency werden in dieser Arbeit evaluiert. Wir wenden diese Methoden auf die Graphenmodelle SumGNN, GCN, PNA und GraphSAGE an. Zunächst stellen wir fest, dass die GNN-spezifischen Algorithmen GNNExplainer und PGExplainer nicht in der Lage sind, Erklärungen ähnlich zu den Ground Truth Werten zu liefern. Im Gegensatz dazu erkennen IG und GradientShap zuverlässig wichtige und unwichtige Kanten, Knoten und Knotenmerkmale, insbesondere beim PNA-Modell, und übertreffen damit alle anderen Erklärungsmethoden.

Abstract

As Graph Neural Networks (GNNs) are increasingly used in critical applications, numerous methods for interpreting these models have been presented in recent years. Furthermore, traditional machine learning explanation approaches can also be extended for use on GNNs. To facilitate the deployment of generic explanation techniques, we design an interface that enables the application of Captum explanation methods to PyG GNNs without requiring additional code changes. Both generic and graph-specific explanation strategies must be thoroughly validated in order to be used reliably. To this point, however, there are no standardized procedures for that purpose available. Therefore, we create benchmarks with datasets that have been generated according to mathematical rules. Using these rules, we derive ground truth explanations for edges, entire nodes, and node features. We investigate the explanation methods GNNExplainer, PGExplainer, Guided Backpropagation, GradientShap, Lime, Integrated Gradients (IG), and Saliency on their performance for GNNs. These methods are applied to the graph models SumGNN, GCN, PNA, and GraphSAGE. First, we find that the GNN-specific algorithms GNNExplainer and PGExplainer cannot provide explanations corresponding to ground truth values. In contrast, IG and GradientShap detect important and unimportant edges, nodes, and node features, especially for the PNA model, and outperform all other explanation methods.

Acknowledgment

I am deeply indebted to Dr. Marina M.C. Höhne for allowing me to write my master's thesis at her institute. Without her support and dedication at every step of the process, I would never have been able to complete this thesis. My thanks go to her for her invaluable patience and feedback. I am also deeply grateful to Prof. Dr. Klaus-Robert Müller, who kindly agreed to review my work. Special appreciation is also due to Dr. Matthias Fey, who supervised me. I thank him for his focused and precise guidance, innovative and creative suggestions, and for sharing his extensive knowledge and profound insights. In addition, I am grateful to Kristof Schütt for the discussions about my work. I would like to express my deep appreciation for his collaboration. During my master's studies, my supervisors at Workist, Dr. Fabian Brosig and Alexander Müller have always supported and encouraged me to apply deep neural networks in practice. I thank them especially for introducing me to the subject of explainability in Graph Neural Networks.

Contents

1	Introduction	1
1.1	Research Questions and Contributions	2
1.2	Scope of this Thesis	3
1.3	Thesis Overview	4
2	Background	5
2.1	Neural Networks	5
2.2	Graph Neural Networks	8
2.2.1	GCN	9
2.2.2	SumGNN	10
2.2.3	GraphSAGE	10
2.2.4	GIN	11
2.2.5	PNA	11
2.3	Explainable Artificial Intelligence	12
3	Related Work	15
3.1	Explaining Graph Neural Networks	15
3.1.1	Generic Methods	15
3.1.2	Graph-Specific Methods	20
3.2	Validating Explanation Methods for Graph Neural Networks	22
4	Developing Interface Between PyG and Captum	27
5	Experiments	31
5.1	General Experiment Setup	31
5.2	Infection Benchmark: Identify Important Path	34
5.2.1	Dataset and Model	34
5.2.2	Results	36
5.3	Titanic Benchmark: Identify Important Node and Noise	38
5.3.1	Dataset and Model	39
5.3.2	Results	40
5.4	Neighborhood Benchmark: Identify Important Edges and Noise	44
5.4.1	Dataset and Model	44
5.4.2	Results	46

5.5	Summation Benchmark: Identify Important Node Features and Noise	49
5.5.1	Dataset and Model	49
5.5.2	Results	51
5.6	Runtime Comparison	55
6	Conclusion	57
	Bibliography	59
A	Appendices	65
A.1	Infection Benchmark	65
A.2	Titanic Benchmark	65
A.3	Neighborhood Benchmark	67
A.4	Summation Benchmark	67

Chapter 1

Introduction

Graph Neural Networks (GNN) are a type of neural networks designed to analyze graph-structured data. The research on GNNs dates back to 1997/98 (Sperduti & Starita, 1997; Frasconi et al., 1998); however, the first significant advancement occurred roughly ten years later (Scarselli et al., 2008; Micheli, 2009). Since then the use of GNNs has increased greatly, accelerated by advances in machine learning (ML), especially deep learning (DL), and by the growing availability of data and computing power.

Traditional convolutional DL techniques have shown great success on various types of data, such as images or text. However, many real-world applications are best modeled using complex structures like graphs or manifolds. GNNs perform exceptionally well on issues of this nature. They are a DL technique that operates on non-Euclidean data to solve problems in various fields, including social networks, recommendation systems, and drug discovery. They can provide a simple approach for tackling tasks at the node, edge, and graph level.

A major requirement in real-world applications of Artificial Intelligence (AI) models, such as GNNs, is that the model’s output should be understandable and trustworthy to the developer (Montavon et al., 2018). However, similar to DL approaches, GNNs suffer from the black-box problem and lack transparency in their decision-making (Burrell, 2016). The algorithm’s complexity is too high for humans to comprehend in a reasonable amount of time. While the interpretability of GNN models may not only aid in detecting prediction errors, it may also boost user trust in the forecasting process (Ribeiro et al., 2016).

To improve the understanding of the models, numerous techniques have been developed for DL models (Samek et al., 2017; Baehrens et al., 2010). The suggested techniques fall under the category of Explainable Artificial Intelligence (XAI). Applying established explanation approaches to GNNs is not trivial due to the graph topology’s irregularity. Therefore, new explanation techniques were created specifically for GNNs, such as GNNExplainer (Ying et al., 2019), PGExplainer (Luo et al., 2020), GNN-LRP (Schnake et al., 2020), and GraphLime (Huang et al., 2022). On the other hand, a number of research projects have modified established explanation methods to be compatible with GNNs (Baldassarre & Azizpour, 2019; Pope et al., 2019). A further distinction can be

made between methods that only evaluate the relevance of node features, those that only evaluate the relevance of edges, and those that evaluate the relevance of both edges and node features to the model output.

Depending on the data and the task at hand, different approaches of explainability methods for GNNs may be more reliable. Therefore, it is necessary to assess and compare various methodologies and to identify each approach’s advantages and disadvantages. This enables the detection of potential weaknesses and directs future research. Due to their intuitiveness, explanatory visualizations are frequently used to assess explanations for image and text data (Samek et al., 2016). However, this approach is not as intuitive for graph data as for other data types. Hence, suitable assessment criteria are needed to evaluate the GNN XAI methodologies.

Due to the novelty of this topic, not much research has been done in this area. Li et al. (2022), Agarwal et al. (2022), and Faber et al. (2021) addressed the evaluation issue of GNN XAI methods using faithfulness matrices. They assessed how precisely the model’s logic is described by the explanation using real-world and toy datasets. Similarly, this work aims to increase the knowledge about the applicability of explanation methods to GNNs, performing fundamental tests to evaluate how well these techniques distinguish between important and less important information in the graph. We build benchmark datasets with specific characteristics, such as datasets where just the nodes, the graph structure, or a limited number of node features include information that is relevant for the prediction. Hence, the GNN XAI methods should pick up these specific attributes in order to be considered satisfactory. In addition, we develop a Captum (Kokhlikyan et al., 2020) integration for PyG (Fey & Lenssen, 2019), where Captum is an open-source library for model interpretability and PyG is a library for graph machine learning, to facilitate the simple use of generic explanation methods on GNNs. Overall, we test explanation approaches for GNNs on a number of benchmark datasets and various GNN architectures. In the following section, we present the research questions and our contribution. We also delimit our study and outline the remaining chapters of the thesis.

1.1 Research Questions and Contributions

There is a lack of research on assessing explainability approaches for GNNs. For this reason, our work focuses on evaluating the usability and accuracy of different XAI methods for graph data.

We aim to understand how the methods work on graphs and answer the following questions:

1. Are generic explanation methods applicable to graph problems?
2. How well does the importance generated by each explanation method match our ground truth understanding of four benchmark datasets?
3. What are the advantages and disadvantages of each explanation method?

4. Can we find a useful combination of graph model and explanation method for a variety of tasks?

To achieve this, we first suggest a strategy for extending generic explanation approaches to graph-structured data. This is to facilitate the applicability of these approaches to GNN architectures. Besides, we evaluate the effectiveness of several generic and GNN-specific XAI methods on four datasets. More precisely, we compare our prior knowledge of the feature importance scores of those datasets to the importance scores derived by the XAI algorithms. The datasets include a benchmark dataset created by Faber et al. (2021) and three datasets we propose with the characteristics listed below:

1. Titanic Benchmark: Most of the information is contained within the **nodes**, whereas the **graph structure** contains noise.
2. Neighborhood Benchmark: Most of the information is contained within the **graph structure**, whereas the **nodes** contain noise.
3. Summation Benchmark: Most of the information is contained within a **small number of node features**, whereas the **remaining features** contain noise.

Our benchmark tests and implementations can also be used to evaluate newly proposed methods in the future. Moreover, this study will help researchers apply generic XAI methods to graph machine learning problems. Besides, answering the research questions will contribute to the body of knowledge related to the evaluation of graph explanation methods. Since the number of available methods is large, this study is intended to guide developers in industry and research in selecting the optimal explanation method.

1.2 Scope of this Thesis

This thesis studies a selection of explanation methods, GNN architectures, and datasets. We restrict the primary evaluation to comparing the ground truth understanding of various datasets and the output of XAI algorithms. It is important to note that ground truth information is often unavailable in real-world scenarios. Therefore, other evaluation methods must be added, e.g., to provide debugging options for real-world use cases. We decided to compare the two GNN-specific methods, GNNExplainer (Ying et al., 2019) and PGExplainer (Luo et al., 2020), and additionally test the methods Saliency (Simonyan et al., 2013), Integrated Gradients (Sundararajan et al., 2017), Guidedbackpropagation (Springenberg et al., 2014), GradientShap (Lundberg & Lee, 2017), and Lime (Ribeiro et al., 2016). All these methods were accessible either through Captum (Kokhlikyan et al., 2020) or PyG (Fey & Lenssen, 2019) at the beginning of our study. Some of these methods can not only provide absolute values for importance but also distinguish between positive and negative attributions. In this thesis, we only consider absolute values. In addition, we consider just four toy datasets. These datasets are relatively small and, thus, only indicate how well selected GNNs and XAI methods would perform on large real-world datasets. The following section provides an overview of the remaining thesis.

1.3 Thesis Overview

In Chapter 1, we outline the motivation for our research. We also present the research questions and discuss the limitations of this thesis.

Chapter 2 offers the necessary background information for the experiments that follow. We explain both neural networks and GNNs. All network architectures utilized in this thesis are described here. A brief introduction to the field of Explainable Artificial Intelligence follows. Furthermore, we answer the following questions: Why do we need explainability? What approaches are available, and where are they used?

Chapter 3 discusses related work, describes the explanation methods used in this thesis, and presents the current state of evaluation of these methods.

Chapter 4, illustrates the interface we have developed for Captum and PyG. We explain the technical implementation and how, from now on, generic XAI methods implemented in Captum can be applied to many GNN architectures provided via PyG.

Chapter 5 contains the experiments. We take five generic methods made applicable to GNNs through the previous chapter and two specific GNN XAI methods and test them on four datasets and four different GNN architectures. We organized the chapter according to the datasets. The experimental setup is described for each dataset, and then the results are listed. Each section focuses on whether the methods on the GNN architectures detect certain important or irrelevant information. In this chapter, we also compare the methods in terms of their runtime performance.

Finally, Chapter 6 presents a summary and a conclusion to this thesis. Future research directions are also identified.

Chapter 2

Background

The following chapter presents the basics of Artificial Neural Networks, Graph Neural Networks, and Explainable Artificial Intelligence techniques. Those fundamentals are required to comprehend this thesis’s problem formulation and proposed solutions.

2.1 Neural Networks

We begin this chapter by covering the topic of neural networks. We answer the following questions: How does a simple neural network work, and how is it optimized? We also introduce the concept of Convolutional Neural Networks, which serve as the foundation for Graph Neural Networks. For a detailed explanation of the subjects in this section, see Nielsen (2015), LeCun et al. (2015), and Goodfellow et al. (2016).

Inspired by the human brain, a neural network processes data to solve problems such as classification or regression. It is a type of machine learning class called deep learning (DL) that consists of a large number of interconnected processing neurons and weighted connections. Typically, the neurons are arranged in layers. Figure 2.1 illustrates the computational graph of a neural network, also called Multilayer perceptron (MLP), where the input layer constitutes the input data, passes the information through weighted connections to the hidden layers, and lastly to the output layer, which represents the model’s prediction. MLPs have several hidden layers. Mathematically, the output $h_j^{(k)}$ of a neuron j in layer k with inputs from the previous layer $h_i^{(k-1)}$ can be written as

$$h_j^{(k)} = \sigma \left(\sum_i w_{ji}^{(k)} h_i^{(k-1)} + b_j \right), \quad (2.1)$$

where the bias b_j represents a threshold, $w_{ij}^{(k)}$ constitutes the weight for the connection from neuron i to j in layer k , and σ denotes a nonlinear activation function such as

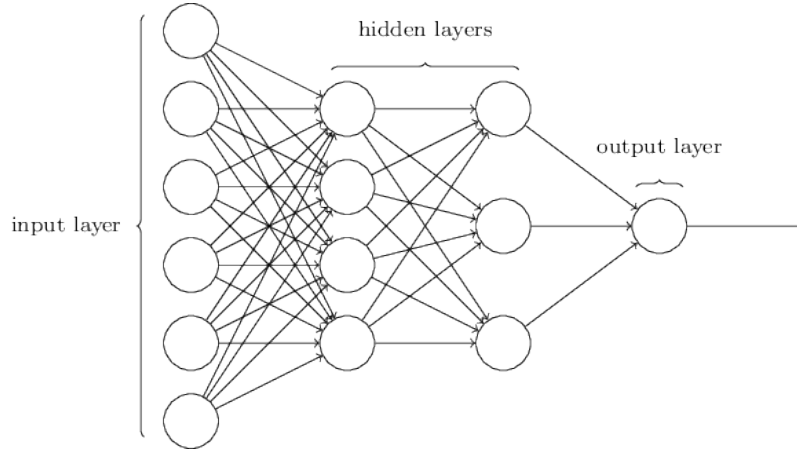


Figure 2.1: Neural network with one input layer, two hidden layers, and one output layer. From Nielsen (2015).

ReLU, formally described as

$$\sigma(z) = \max(0, z). \quad (2.2)$$

In the first layer, \mathbf{h}^0 corresponds to the input data \mathbf{x} . Furthermore, in matrix notation the output or so-called activation $\mathbf{h}^{(k)}$ of all neurons in layer k is described by the equation

$$\mathbf{h}^{(k)} = \sigma(W^{(k)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}). \quad (2.3)$$

The weight matrix $W^{(k)}$ for layer k contains all weights connecting neurons from layer $k - 1$ to layer k . In the beginning, the model parameters, such as weights and biases, are, e.g., randomly initialized. However, the remarkable power of ANNs lies in their ability to adjust those parameters through a process called learning or training. This procedure includes a loss function that tells us "how well" the model is making predictions based on the current set of neural network parameters with respect to a known target or label \mathbf{y}^m . Accordingly, the loss function L of all learnable parameters Θ of the neural network can generally be described as

$$L(\Theta, \mathbf{x}^m, \mathbf{y}^m) = \frac{1}{M} \sum_{m=1}^M J(f(\Theta, \mathbf{x}^m), \mathbf{y}^m), \quad (2.4)$$

where $f(\Theta, \mathbf{x}^m)$ is the prediction function for sample m based on the model parameters Θ and the input vector \mathbf{x}^m . J is used to evaluate how well the neural network performs per instance. M is the total number of examples. The method J is chosen depending on the task at hand. When dealing with a dataset with multiple classes, we employ the cross entropy as a loss function in our experiments, that is mathematically described as

$$L(\Theta, \mathbf{x}^m, \mathbf{y}^m) = \frac{1}{M} \sum_{m=1}^M -(\mathbf{y}^m \ln f(\Theta, \mathbf{x}^m) + (1 - \mathbf{y}^m) \ln (1 - f(\Theta, \mathbf{x}^m))), \quad (2.5)$$

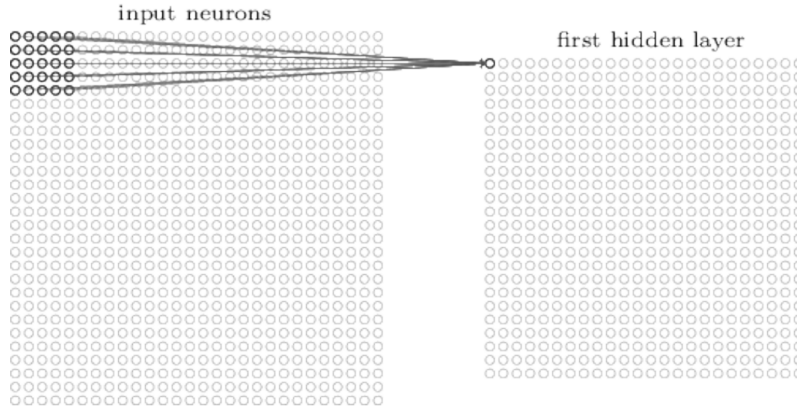


Figure 2.2: Local receptive field (left) of a neuron belonging to the first hidden layer (right). From Nielsen (2015).

and the mean absolute error when dealing with regression problems

$$L(\Theta, \mathbf{x}^m, \mathbf{y}^m) = \frac{1}{M} \sum_{m=1}^M |f(\Theta, \mathbf{x}^m) - \mathbf{y}^m|. \quad (2.6)$$

The goal is to minimize these loss functions during the training process. Stochastic gradient descent (SDG) or adaptive learning rate optimization (Adam) are optimization methods used in this regard.

After the training is completed, a so-called test set, which is not used during the learning process, is employed for an unbiased evaluation of the model. In addition to the model parameters mentioned so far, hyperparameters may also be fitted for specific architectures (see the following section). Therefore, an additional validation set is used to select the model with optimal hyper-parameters. Evaluating the trained model can be done using the mentioned loss functions. However, alternative evaluation metrics exist that are more intuitive to humans, such as accuracy, which we will use in the following experiment, among others.

Although the neural network architecture described so far is well suited for various problems, the computational and storage complexity is high as soon as one deals with high-dimensional data such as images. This is due to the networks' fully connected nature. Each neuron in one layer is connected to all neurons in the preceding layer, cf. Figure 2.1. A different type of model, the Convolutional Neural Network (CNN), overcomes this problem.

CNNs use local connections, shared weights, and pooling to exploit local information. As shown in Figure 2.2, instead of connecting a neuron to every neuron in the preceding layer, it is only connected to a small local region. This region is referred to as a neuron's receptive field and is, e.g., 5x5 pixels in size. Along a layer, the same weights and biases are applied to each receptive field. The mapping from one layer to the next layer is also called a feature map. Typically, multiple of these feature maps are utilized in one convolutional layer, cf. Figure 2.3. A convolutional layer is usually followed by a

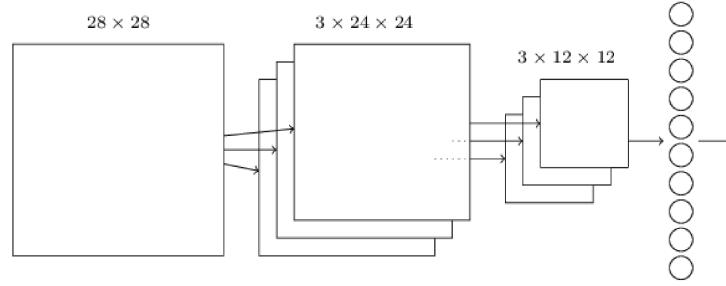


Figure 2.3: A basic Convolutional Neural Network applied on a 2D input image, containing two hidden convolutional layers with three feature maps, followed by a pooling layer and a fully connected layer. From Nielsen (2015).

pooling layer. A pooling layer condenses the feature map. Thus, fewer parameters are required. Finally, as with basic neural networks, a fully connected layer is utilized to generate the output, see Figure 2.3. All neurons of the previous layer’s feature maps are connected to the neurons of the final layer.

CNNs have a much smaller memory footprint than fully connected networks. CNNs are robust and can recognize features from the local neighborhood that are shared with the entire dataset. Similar models are needed for graph input data. The difference here, however, is the non-Euclidean nature of the data, which is why CNNs cannot simply be used with graphs. Therefore Graph Neural Networks have been developed, which are explained in more detail in the next section.

2.2 Graph Neural Networks

Data with graph structure is ubiquitous in the real world. The graph data type can capture the interactions (edges) between individual entities (nodes) and is widely applied to encode data from irregular or non-Euclidean domains. Examples of graphs include social networks, biological networks, and the World Wide Web, on which numerous tasks can be performed naturally. These tasks include link prediction, node classification, and graph classification. However, such tasks require effective node embeddings that incorporate both the properties of the nodes and the graph’s structure. Graph neural networks (GNNs) have been developed specifically for learning such representations. GNNs have gained increasing attention, especially in recent years, and can learn both node and entire graph embeddings. This paper focuses only on the first variant, learning node representations. In the following section, we provide a mathematical foundation for graphs. Then, we introduce a general framework for GNNs and finally explain the exact network concepts that are also used in the experiments of this thesis.

To describe a graph formally, we use $G = (V, E)$, where $V = v_1, v_2, \dots, v_N$ is a set of $N = |V|$ nodes and $E \subseteq V \times X$ describes a set of $P = |E|$ edges between nodes. The edges can be directed or undirected. Directed edges exist if there is a directed dependency between nodes. Each node has a set of features \mathbf{x}_v , where X is the full

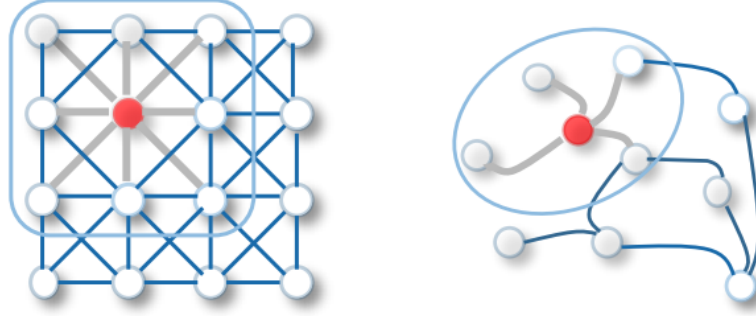


Figure 2.4: **Comparison between a traditional convolutional layer (left) and a graph convolutional layer (right).** From Wu et al. (2020).

feature matrix of the graph. The neighborhood of node v is defined as $N(v)$. This set contains all nodes reachable by one hop from node v .

While there are numerous types of GNNs, the most effective and flexible is the neighborhood aggregation or message passing approach (Ye & Ji, 2021). In this type of GNN, the representation of node features is determined by collecting information from neighboring nodes and then incrementally updating the node itself. The general framework proposed by Gilmer et al. (2017) is therefore described with two functions:

$$\mathbf{a}_v^{(k)} = \text{Aggregate}^{(k)}(\{\mathbf{h}_u^{(k)}, \forall u \in N(v)\}), \quad (2.7)$$

$$\mathbf{h}_v^{(k)} = \text{Combine}^{(k)}(\{\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)}\}), \quad (2.8)$$

here, the representations of the initial nodes correspond to $\mathbf{h}^0 = \mathbf{x}$. K layers are stacked on top of each other so that the final node embedding is composed of the features belonging to all k -hop neighbors. At the end, similar to CNNs, a fully connected neural network is used for the downstream task. Overall, the method is an extension of traditional convolution operators. The same principles as weight sharing or pooling are applied. Images can also be considered as a particular case of graphs, where instead of disordered, unfixed nodes, pixels are regularly connected to their neighboring pixels, cf. Figure 2.4.

While the research field is constantly expanding, there are already numerous architectures for graph data. These include the Graph Convolutional Network (GCN), SumGNN, GraphSAGE, the Graph Isomorphism Network (GIN), and the approach of Principle Neighborhood Aggregation (PNA). In the following sections, we will discuss these strategies in detail.

2.2.1 GCN

We begin with the Graph Convolutional Network (GCN) introduced by Kipf & Welling (2016). It is the most common layer model due to its ease in use and effectiveness in a wide range of tasks (Tang & Liao, 2022). In this technique, the representation of the

nodes of each layer is updated with

$$\mathbf{h}_v^{(k)} = \sigma \left(W^{(k)} \sum_{u \in N(v) \cup v} \frac{\mathbf{h}_u^{(k-1)}}{\sqrt{|N(u)||N(v)|}} \right), \quad (2.9)$$

where σ is the activation function, as introduced previously, and $W^{(k)}$ comprises the learnable weight parameters for layer k . The function can be divided into the *Aggregate* function, which calculates the weighted average of the neighboring nodes, and the *Combine* function, which computes the sum of the aggregated adjacent nodes with the node itself, which is likewise normalized by its node degree. In conclusion, GCN adds up all node embeddings while limiting the impact of nodes with a high degree.

2.2.2 SumGNN

Secondly, we introduce the simple architecture SumGNN, first mentioned by Morris et al. (2019) and Hamilton et al. (2017). Unlike the GCN described earlier, the nodes are not weighted in this approach. Also, two weight matrices are used per iteration instead of only one. Mathematically, the node representation in each layer is described as

$$\mathbf{h}_v^{(k)} = \sigma \left(W_1^{(k)} \mathbf{h}_v^{(k-1)} + \sum_{u \in N(v)} W_2^{(k)} \mathbf{h}_u^{(k-1)} \right). \quad (2.10)$$

The *Aggregate* function adds up the node embeddings of the neighboring nodes, and the *Combine* function also calculates the sum over the aggregated neighbors and the node itself.

2.2.3 GraphSAGE

Another extension of the original GCN is the Graph SAmple and AggreGatE technique (GraphSAGE), introduced by (Hamilton et al., 2017). The method introduces a general aggregation function for which the paper provides several options. Thus, the node representation for each layer can be described in the terms

$$\mathbf{a}_v^{(k)} = \text{Aggregate}^{(k)} (\{\mathbf{h}_u^{(k)}, \forall u \in N(v)\}), \quad (2.11)$$

$$h_v^{(k)} = \sigma (W^{(k)} [\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)}]). \quad (2.12)$$

Depending on the use case, the aggregation function can be chosen. Common aggregation functions include mean or sum, as described in the previous methods. Alternatively, LSTM or the pooling aggregator were introduced by the paper. The max pooling aggregator is defined as

$$\text{Aggregate}^{(k)} = \text{MAX} \left(\left\{ \sigma \left(W^{(k)} \cdot \mathbf{h}_u^{(k-1)} \right), \forall u \in N(v) \right\} \right). \quad (2.13)$$

The combine function in GraphSAGE is a linear combination of the aggregated neighborhood representation and the node itself rather than a sum.

2.2.4 GIN

The Graph Isomorphism Network (GIN) was proposed by Xu et al. (2018). The architecture aims to maximize the expressive power of GNNs. Accordingly, different graph structures should be mapped to different node representations. Therefore, the following update scheme for node representations is formalized to

$$\mathbf{h}_v^{(k)} = \text{MLP}^{(k)} \left((1 + \epsilon^{(k)}) \mathbf{h}_v^{(k-1)} + \sum_{u \in N(v)} \mathbf{h}_u^{(k-1)} \right). \quad (2.14)$$

First, the sum of the neighboring nodes is calculated. Furthermore, the node itself is weighted with a learnable parameter ϵ . Using one MLP per layer also ensures that the representation capacity of the model is maximized.

2.2.5 PNA

Finally, we present the Principle Neighborhood Aggregation Network (PNA), introduced by Corso et al. (2020). While all methods previously discussed use a single aggregator, the PNA architecture employs multiple aggregators. The paper by Xu et al. (2018) found that different single aggregation functions cannot distinguish between node neighborhood embeddings of different graph structures. In addition to using an MLP, as in GIN, a combination of degree scalars and aggregation operators are utilized here. The generation of the node representation of a layer is described with

$$\mathbf{h}_v^{(k)} = \text{MLP}_1^{(k)} \left(\mathbf{h}_v^{(k-1)}, \bigoplus_{u \in N(v)} \text{MLP}_2^{(k)} \left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)} \right) \right). \quad (2.15)$$

This uses a scalar-operator combination which is defined mathematically by

$$\bigoplus_{u \in N(v)} = \begin{bmatrix} I \\ S(d, \alpha = 1) \\ s(d, \alpha = -1) \end{bmatrix} \otimes \begin{bmatrix} \text{mean} \\ \text{std} \\ \text{max} \\ \text{min} \end{bmatrix}, \quad (2.16)$$

and the composition of the degree scalar resembles with

$$S(d, \alpha) = \left(\frac{\log(d+1)}{\delta} \right)^\alpha, \quad (2.17)$$

where d is the degree of node v . The parameter α is negative for attenuation, positive for amplification, and zero for no scaling. δ is a regularization parameter that both correlates with the average degree d_{Train} and the total number of nodes in the training dataset V_{Train} , mathematically defined as

$$\delta = \frac{1}{d_{Train}} \sum_{i \in V_{Train}} \log(d_i + 1). \quad (2.18)$$

The techniques mentioned in this section can be used for many prediction applications on graphs. However, their inner workings remain concealed, as with other DL systems. Thus, in the next chapter, we focus on the concepts available to explain the underlying decision-making process of AI models.

2.3 Explainable Artificial Intelligence

The use of neural networks of various types, such as MLP, CNN, or GNN, is incredibly effective in a number of applications, such as translations (Bahdanau et al., 2014) or in industries such as automotive (Bojarski et al., 2016) or healthcare (Thomas et al., 2019). They offer the advantage of adaptive learning over other algorithms and can perform tasks on data that is either very simple or even quite complicated. However, one of the disadvantages of DL approaches is that they are black-box models (Molnar, 2020). In other words, the actual process of how the models reach a decision cannot be understood by looking at its parameters. This is a major limitation of DL (Bach et al., 2015; Baehrens et al., 2010). The lack of transparency associated with, for instance, using AI models in high-stake decisions drives the demand for explainability. Hence, in this section, we broadly introduce the research field of Explainable Artificial Intelligence (XAI).

In this area, a variety of approaches have been developed to improve the understanding of the prediction process underlying the models while maintaining their performance. Doshi-Velez & Kim (2017) identified the goals that explanation methods and consequent interpretability of models can achieve as follows. First, *improving the fairness* of a model's predictions can be addressed, whereby interpretability can be used to uncover discriminatory predictions for specific groups. In addition, *ensuring privacy* is mentioned by disclosing the use of sensitive data. Furthermore, *ensuring the reliability and robustness* of the model is listed, which is achieved by investigating with explanation methods whether minor changes in the inputs affect the model's prediction. Likewise, *ensuring causality* is mentioned, whereas, with interpretability, we investigate if there is a relationship between technical explainability and human understanding. Finally, *improving trust* in the model is noted; this is important as people are more inclined to use a system that explains its results.

Due to the wide range of interpretability challenges and network architectures, several XAI methods have been developed that differ in scope, i.e., in the part of the prediction process, they are designed to explain (Molnar, 2020; Murdoch et al., 2019; Das & Rad, 2020).

Global XAI methods aim to explain the models' overarching decision strategy (Molnar, 2020). Specifically, they seek to identify concepts to which certain neurons respond most strongly (Bau et al., 2020). They also analyze how specific attributes affect predictions on average. This approach is used, for example, to uncover potential biases in models or to provide new scientific insights (Murdoch et al., 2019). However, due to the data's high dimensionality, achieving a global explanation in practice is challenging (Molnar, 2020).

Local XAI methods, on the other hand, aim to explain individual predictions (Molnar, 2020). The goal of gaining an understanding of a specific instance is not only easier to achieve in practice but also has the potential to be more accurate, as we may be able to find linear dependencies for a single instance (Molnar, 2020). Local explainability techniques typically assign relevance scores to each input feature that correlate positively or negatively with the classification outcome of the model.

Following, we will only discuss local explainability approaches. These, in turn, can be divided into different categories concerning their methodology. In the remainder of this report, we utilize gradient-based, perturbation-based, and surrogate models.

Gradient-based explanation methods incorporate the backward pass in a neural network, originally used for training, to understand the impact and relevance of an input feature towards the output class (Ancona et al., 2019). Prominent examples include Integrated Gradients (Sundararajan et al., 2017) and Saliency (Simonyan et al., 2013).

Perturbation-based explanation methods investigate how small input variations affect the prediction to explain individual feature attributions of the model towards an output class (Samek & Müller, 2019). Occlusion (Zeiler & Fergus, 2014) is a well-known method here.

Surrogate-based explanation methods approximate the predictions of high nonlinear AI models by an interpretable linear model or a decision tree (Samek & Müller, 2019). Lime (Ribeiro et al., 2016) can be mentioned here as an example.

In the following chapter, we will examine individual methodologies that have been tried on graphs.

Chapter 3

Related Work

This chapter addresses the functionality of individual local explanation methods and their application to GNNs. In detail, we provide an overview of different techniques for explaining GNNs, starting with generic strategies, followed by a description of their extension to graphs, and concluding with methods explicitly developed for GNNs.

3.1 Explaining Graph Neural Networks

As outlined in Chapter 2, local explainability involves assigning relevance and attribution values for predicting a single instance class. So far, most local explanation methods have been designed for grid-like inputs such as images or text. These methods are called generic explanation methods in this thesis. However, given that graphs are unsorted, their inputs also include topological information in the form of edges. Hence, both edges and node features together contribute to the final prediction of a graph machine learning model. This should be detectable by an explanatory framework. Consequently, either existing generic explanation methods need to be expanded to identify more than one type of information, or GNN-specific explanation approaches that directly incorporate graph structural information are needed. This chapter provides an overview of different techniques for explaining GNNs, starting with generic strategies and concluding with methods explicitly developed for GNNs. All explanation methods covered in this section can be found in Figure 3.1.

3.1.1 Generic Methods

Various generic interpretability approaches for DL models have been proposed in recent years. These include Saliency, Integrated Gradients, Guided Backpropagation, GradientShap, and Lime, which we describe in this section. All of these methods have been made available in a unified framework via the open-source tool Captum (Kokhlikyan et al., 2020). It is built on PyTorch and provides researchers and developers with a comprehensive set of implementations of explanation methods. We first introduce the above techniques in general and then explain their extension to graphs.

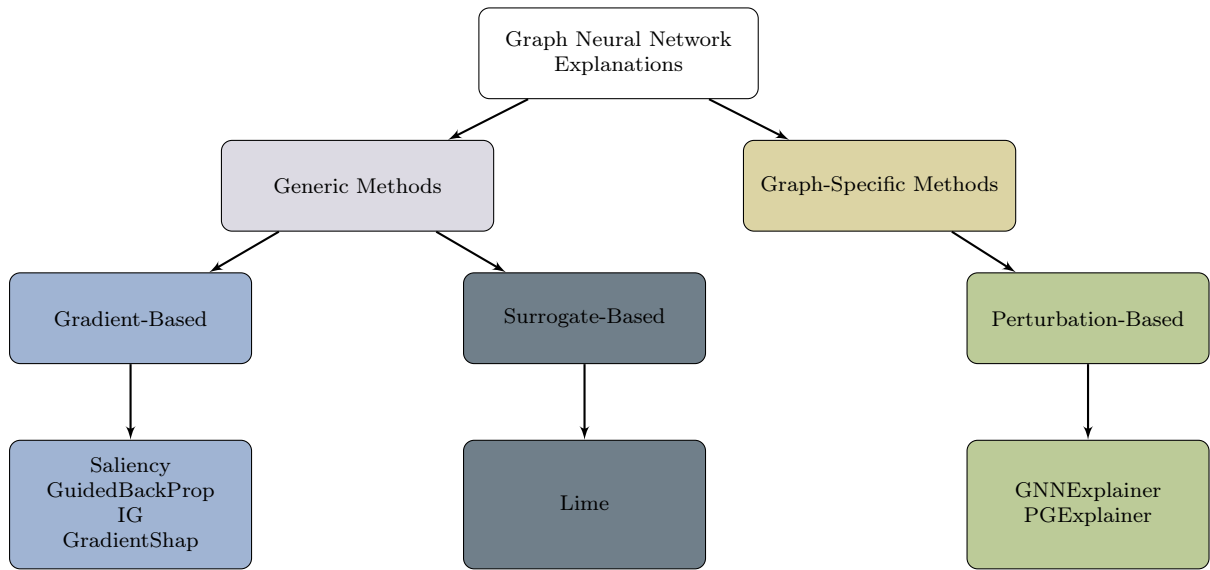


Figure 3.1: An overview of the methods reviewed in this thesis. We categorize existing GNN explanation approaches into two types: generic methods and GNN-specific methods. Lime (Ribeiro et al., 2016) is a surrogate-based technique, and Saliency (Simonyan et al., 2013), GuidedBackProp (Springenberg et al., 2014), IG (Sundararajan et al., 2017), and GradientShap (Lundberg & Lee, 2017) are gradient-based techniques that fall under the umbrella of generic methods. Two instances of perturbation-based graph-specific methods that we examine are GNNExplainer (Ying et al., 2019) and PGExplainer (Luo et al., 2020).

Saliency

Saliency, introduced by Simonyan et al. (2013), can be considered a baseline approach for computing input attributions. It produces local explanations for a differentiable model utilizing the gradient with respect to an input feature x_i : $\nabla_{x_i} f$. The gradient, intuitively, describes the amount of variation that a small change in a feature causes on the associated prediction. This approach has the disadvantage of having a saturation problem (Shrikumar et al., 2017). When the model output varies only minimally in response to any input change, the gradients can rarely capture the contributions of inputs.

Guided Backpropagation

Guided Backpropagation (GuidedBackProp), introduced by Springenberg et al. (2014), is founded on the concept of Saliency. While it also calculates the gradients for the output concerning the input, it additionally introduces a guidance process. This process restricts the flow of gradients to those that have positive values. Values for negative gradients are set to zero. This is to reduce the amount of noise in the attributions.

Integrated Gradients

Integrated Gradients(IG), introduced by Sundararajan et al. (2017), is another gradient-based methodology. Instead of calculating gradients only on the current value of the input and thus considering only local changes around the input, IG integrates all changes from a chosen baseline to the current value. This solves the saturation problem of Saliency. Formally, calculating the average of all gradients along a straight line between a baseline x'_i and the input value x_i can be expressed by the integral

$$IG_i(x) = (x_i - x'_i) \int_{\alpha=0}^1 \frac{\delta f(x' + \alpha(x - x'))}{\delta x} d\alpha. \quad (3.1)$$

In practice, the integral is approximated by a finite number of samples n using the Riemann principle, following

$$IG_i(x) = (x_i - x'_i) \sum_{k=1}^n \frac{\delta f(x' + \frac{k}{n}(x - x'))}{\delta x_i} \frac{1}{n}. \quad (3.2)$$

This approach, in turn, can lead to an approximation error. The fundamental benefit of the approach is that it satisfies two axioms: sensitivity (every input information that modifies the classification result in any manner has a relevance score other than 0) and invariance (for two different neural networks that give the same prediction for the same inputs, their attribution values are identical).

GradientShap

GradientShap, introduced by Lundberg & Lee (2017), is an explanation method based on Shapley values. Originating from game theory, Shapley values describe how the winning sum of a game should be distributed among all participants with respect to their individual contributions. A player's Shapley value is defined as

$$\phi_i = \frac{1}{N!} \sum_R [f(P_i^R \cup \{i\}) - f(P_i^R)], \quad (3.3)$$

where N is the total number of players, P_i contains all permutations (subsets) of players participating excluding player i and $f(P_i^R \cup \{i\})$ corresponds to the game outcome with permutation P_i^R of players including player i . In summary, this corresponds to the average participation of player i in all different combination scenarios. GradientShap attempts to approximate the Shapley values and thus distributes the initial value of the model among the individual features involved according to their participation. This is achieved by drawing random samples along a distribution of baselines and adding noise to them. Also, a random point is chosen along the path between the baseline and the input, and then the gradient is calculated with respect to the input. This method can be considered an extension of IG, where the gradient is calculated using different baselines rather than just one baseline.

GradientShap satisfies both local precision and consistency. Local precision is defined as the sum of feature attributions equaling the prediction output of the model. Consistency implies that even if we update a model such that one feature has a more significant impact on the model, the importance attributed to that feature would never decrease. One of the drawbacks of GradientShap can be its computational complexity. (Holzinger et al., 2022)

Lime

Lime (Local Interpretable Model Agnostic Explanations), introduced by Ribeiro et al. (2016), provides explanations by fitting a local surrogate model whose predictions can be interpreted by humans. Technically, Lime generates samples in a neighborhood N_x around an input x , computes the output for this input using the DL model to be explained, and then approximates the model in the generated local neighborhood using a simple linear function. Thus, minimizing the function

$$\operatorname{argmin}_{t \in T} L(f, t, \pi_x) + \Omega(t), \quad (3.4)$$

where T represents the set of possible explanation models. In addition, $\Omega(t)$ quantifies the complexity of those models, which should be as low as possible. A loss function L is selected to measure the extent to which a surrogate model t deviates from the true predictions for all inputs within a given neighborhood. The neighborhood size is set by π_x .

Lasso, which calculates sparse coefficients, is an example of a surrogate model class. Mathematically, it is a linear model with an extra regularization component that can be minimized with

$$\min_w \frac{1}{2N} \|Xw - y\|_2^2 + \alpha \|w\|_1, \quad (3.5)$$

where N is the total number of samples found in the local neighborhood, X is the feature matrix containing all of those samples, and y is the original model's output vector for one class. $\|\cdot\|_1$ and $\|\cdot\|_2$ are the l1 and l2 normalizations, respectively, and w contains the model parameters. We also employ the Lasso model in our experiments.

There are currently numerous successful deployments of Lime in various application sectors, demonstrating its widespread adoption. However, one disadvantage of Lime is that its explanation is heavily dependent on the fit of the surrogate model, which can require dense sampling and high computational costs. Sampling always introduces uncertainty, which can result in non-deterministic behavior and numerous interpretations of the same input sample. (Holzinger et al., 2022)

Extension for GNNs

All generic approaches, including those presented in this chapter, have been implemented and tested primarily on data that is not graph-structured. Therefore, several research papers have been published to clarify whether these strategies also work for GNNs. The following section explains how they extended the original implementations.

Pope et al. (2019) created a graph-analog version of various gradient-based approaches, including Saliency. They examined the capability of computing attribution values for node features. In order to achieve this, they modified the gradient computation and computed it with regard to each input node feature.

Similarly, Baldassarre & Azizpour (2019) adopted Saliency and GuidedBackProp, as well as other gradient-based algorithms. However, they calculated the gradient not only for the node feature but also with respect to the edges.

In addition, Schlichtkrull et al. (2020) deployed IG on graphs in order to determine edge attributions. They define the scalar variable $\hat{z}_{u,v}^k$, multiply it by the message from node u to v in layer k , and interpolate $\hat{z}_{u,v}^k$ between 1 and 0. Then they determine the relative attribution $\hat{z}_{u,v}^k$, with 0 serving as the baseline. With this baseline, they presume that the states of "completely present" and "entirely absent" of edges are interpolated by "partially present." However, they also note that this strategy is not trivial, as "partially present" edges in upper layers influence the gradient flow and, consequently, the assignment to lower layers during interpolation.

Further research has been conducted using generic explanation methods in a similar fashion (Ying et al., 2019; Luo et al., 2020; Vu & Thai, 2020). However, to our knowl-

edge, GradientShap and Lime have not yet been used to investigate the relevance of graph structure.

Each of the research above has implemented its extension of the generic XAI methods for GNNs. While there are frameworks for traditional NNs or CNNs, such as Captum, that unify many implementations of XAI methods, making it easier to test different methods, there is no such framework for GNNs. Since PyG is the most widely used GNN library for PyTorch, we have included a functionality that allows Captum to be used on models generated with PyG. Thus, Captum’s complete set of explainability techniques can be applied to GNNs without further modification. This will be covered in greater depth in Chapter 4.

3.1.2 Graph-Specific Methods

In addition to generic techniques, several graph-specific approaches have been proposed to explain the predictions of GNNs. Some of these explanation methods, such as GraphLime (Huang et al., 2022), Zorro (Funke et al., 2020), and PGM-Explainer (Vu & Thai, 2020), specialize in finding the attribution of node features. Other methods such as PGExplainer (Luo et al., 2020), GraphMask (Schlichtkrull et al., 2020), and Causal Screening (Wang et al., 2020) compute attribute values for edges only. Moreover, SubgraphX (Yuan et al., 2021) explores subgraph explanations and GNN-LRP (Schnake et al., 2020) searches for relevant walks in the graph. Whereas GNNExplainer (Ying et al., 2019) can compute relevance values for edges as well as for nodes. GNNExplainer and PGExplainer are both implemented in the GNN open-source library PyG (Fey & Lenssen, 2019). Therefore, in the following, we will explain GNNExplainer and PGExplainer in more detail. We also used these methods in our experiments due to the simplicity of their application.

GNNExplainer

GNNExplainer, introduced by Ying et al. (2019), is suitable for all GNN-based models and aims at learning a mask for edges and/or node features. The optimization of an edge mask M is based on the concept of maximizing the mutual information (MI) between the prediction on the original graph G and on a new subgraph G_S formed using the mask, containing the associated features $X_S = \{x_j \mid v_j \in G_S\}$. GNNExplainer formally specifies the optimization framework as:

$$\max_{G_S} MI(Y, (G_S, X_S)) = H(Y) - H(Y \mid G = G_S, X = X_S), \quad (3.6)$$

where MI measures the change in conditional entropy $H(\cdot)$ when the graph is restricted to the explanation subgraph (G_S, X_S) . Considering that the first term is constant, we can also only minimize the conditional entropy:

$$H(Y \mid G = G_S, X = X_S) = -\mathbb{E}_{Y \mid G_S, X_S} [\log P_{\Theta}(Y \mid G = G_S, X = X_S)]. \quad (3.7)$$

In this section, we follow the terms of Ying et al. (2019) for simplicity. Thus a GNN model learns a conditional distribution $P_{\Theta}(Y|G, X)$, where Y is a random variable representing labels $\{1, \dots, C\}$, indicating the likelihood of nodes belonging to each of C classes. As the number of viable masks is proportional to the number of nodes, GNNEExplainer employs a gradient descent-based optimization minimizing the cross-entropy function:

$$\min_M - \sum_{c=1}^C \mathbb{1}[y = c] \log P_{\Theta}(Y = y | G = E \odot \sigma(M), X = X_c). \quad (3.8)$$

Moreover, two regularizations are introduced. First, a high explanatory size is penalized by summing up the mask parameters, and second, the masks are encouraged to be discrete by an element-wise entropy. The approach presented so far refers only to finding edge masks. However, the conditional entropy can also be applied to finding a node feature mask: $H(Y | G = G_S, X = X_S^F)$, where X_S^F contains all non-masked features. The optimization then follows the same procedure. A disadvantage of GNNEExplainer is that the masks are optimized for each input graph individually; hence, the explanations may lack a global view.

PGExplainer

PGExplainer, introduced by Luo et al. (2020), is based on the same concept as GNNEExplainer. However, it additionally includes a global understanding of the trained GNNs. Similarly to GNNEExplainer, the algorithm aims at maximizing MI (see Equation 3.6).

First, the GNN-based model with parameters Θ is split into two parts: GNN_{Θ_0} learns the node representations Z and GNN_{Θ_1} feeds the vector representations to the downstream tasks generating Y according to

$$Z = GNN_{\Theta_0}(G, X), Y = GNN_{\Theta_1}(Z). \quad (3.9)$$

For the creation of an edge mask \hat{G}_S PGExplainer trains an explanation network $\Omega = g_{\Psi}(G, Z)$. Given an input graph G , it obtains the embeddings Ω for each edge by concatenating the node embeddings Z with the input graph G . Those edge embeddings Ω are latent variables in edge distributions, which can be considered as an importance value. A random graph \hat{G}_S is sampled from the edge distributions Ω and then fed to the trained GNN model to receive prediction \hat{Y} . The parameters Ψ in the explanation network are optimized via cross-entropy between the original prediction Y and the updated prediction \hat{Y} , mathematically described as:

$$\min_{\Psi} - \sum_{i \in \mathcal{I}} \sum_{k=1}^K \sum_{c=1}^C P_{\Theta}(Y = c | G = G^{(i)}) \log P_{\Theta}(Y = c | G = \hat{G}_s^{(i,k)}). \quad (3.10)$$

The explanation model is trained on a set of instances \mathcal{I} . C is the number of labels, K is

the total number of sampled graphs, and $\hat{G}_S^{i,k}$ is the k -th sampled graph, for instance, i . PGExplainer only trains a model once on multiple instances; due to that, it incorporates a global view and can be significantly faster compared to GNNExplainer.

All proposed XAI strategies need to be tested experimentally to determine whether they can provide trustworthy explanations. Therefore, the following chapter provides an overview of the current state of research on the performance evaluation of the presented methods.

3.2 Validating Explanation Methods for Graph Neural Networks

Over the past few years, a number of techniques have been presented to explain GNNs. This section reviews the work in which those methods have been evaluated.

The first studies on the explainability of GNNs, which included (Pope et al., 2019) and Baldassarre & Azizpour (2019), focused primarily on the application of generic explanation methods on molecular datasets. They used visual inspection to compare explanations. In addition, (Pope et al., 2019) also utilized fidelity and sparsity for their evaluations, which are discussed below.

Subsequent publications have developed strategies tailored to the problem of explaining GNNs. The study by Yuan et al. (2021) provides a comprehensive overview of various general and graph-specific GNN explanation techniques, as well as their evaluations. They compare the methods in terms of several characteristics, including the type of explanation (local vs. global), the type of task they are best suited for (graph, node, or link-level prediction), and the type of explanation they provide (relevance scores for node features, nodes, and/or edges). They also list the most popular datasets for testing GNN explanation techniques:

- *BA-Shapes* contains a random graph generated by the Barabási-Albert model and a set of house-structured motifs randomly attached to selected nodes of the random base graph. Each node is classified according to whether it belongs to the base graph or to one of the spatial positions of the motif (top, middle, bottom). For the classification of a node belonging to the house motif, the structure of the house graph itself is considered as the ground truth importance.
- *BA-Community* is a combination of two BA-shaped graphs with eight possible node labels, four for each community in the graph. The nodes have normally distributed feature vectors, with one feature differing by the community. As a result, in order to classify a node belonging to a community’s house motif, the house motif is considered as the ground truth importance as well as the decisive node feature.
- *Tree-Cycle* contains a balanced tree and cycle motifs attached to randomly selected tree nodes. Each node is classified according to whether it belongs to the tree or the cycle. For the classification of a node belonging to the cycle motif, the structure of the cycle itself is considered the ground truth importance.

- *Tree-Grid* is similar to the Tree-Cycle dataset, but the motif is a grid rather than a cycle.
- *BA-2Motif* contains a random graph generated using the Barabási-Albert model and one motif, either a cycle or a house. Each graph is classified according to whether it contains a house or a cycle motif. The graph structure of the motif is considered the ground truth importance for classification.
- *MUTAG* contains molecular graphs labeled according to their mutagenic effect on Gram-negative bacteria. The NO_2 and NH_2 compounds are regarded as important for the graph classification. In contrast to the aforementioned datasets, this is not a synthetic dataset.

In Table 3.1, we visualize all of the datasets and attached the papers that have used these datasets to evaluate GNN explanation methods so far. Additionally to the datasets, Yuan et al. (2021) provides an overview of the metrics used to assess the GNN explanatory methods, containing the following:

- *Sparsity*, measures an explanation’s localisation. Good explanations should be sparse, capturing only the most important input features while ignoring the irrelevant ones.
- *Fidelity* quantifies the variation in the prediction while removing all node features/edges from the input data that are considered crucial by the explanation method. However, a disadvantage of this metric is that the data may not follow the training distribution after removing crucial data anymore (Hooker et al., 2019). This might be the reason for the change in model behavior as much as for the removal of essential data. For graphs, in particular, deleting a node or edge carries a significant risk of out-of-distribution inputs due to their discrete nature (Faber et al., 2020).
- *Stability* evaluates whether the explanations of a graph are identical when compared to its perturbed counterpart with infinitesimally small changes. It is assumed that the predictions for both graphs are identical.
- *Ground Truth Comparison* evaluates the correctness of an explanation based on the data set’s ground truth understanding. The rule used to create a synthetic dataset can be utilized to provide a reasonable approximation of ground truth. F1, accuracy, or ROC AUC scores can be computed for the identified edges/node features.

In addition to Yuan’s review, empirical studies have been conducted to evaluate GNN explanation methods (Sanchez-Lengeling et al., 2020; Agarwal et al., 2022). In the latter, the methods were tested on the described metrics using the aforementioned datasets, among others. Methods were also tested for their applicability to real-world datasets. Furthermore, a theoretical evaluation of GNN explanation methods was carried out (Agarwal et al., 2022).

As for the comparison with ground truth, which is an extremely powerful evaluation tool, all approaches were evaluated almost exclusively with the above datasets. Faber

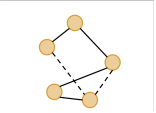
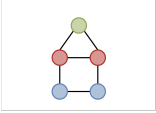
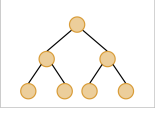
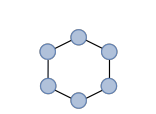
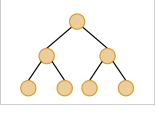
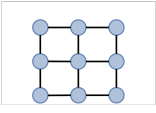
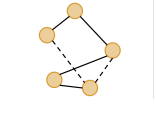
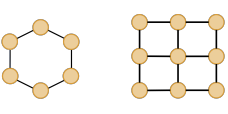
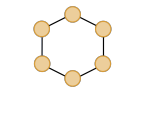

Datasets	Base	Motif	References
BA-Shapes			Ying et al. (2019) Luo et al. (2020) Sanchez-Lengeling et al. (2020) Vu & Thai (2020) Bajaj et al. (2021) Yuan et al. (2021) Lin et al. (2021) Lucic et al. (2022)
Tree-Cycles			Ying et al. (2019) Luo et al. (2020) Vu & Thai (2020) Bajaj et al. (2021) Lin et al. (2021) Lucic et al. (2022)
Tree-Grids			Ying et al. (2019) Luo et al. (2020) Sanchez-Lengeling et al. (2020) Vu & Thai (2020) Lucic et al. (2022)
BA-2Motifs			Bajaj et al. (2021) Luo et al. (2020) Wang et al. (2021) Yuan et al. (2021)
MUTAG		 NH_2 NO_2	Ying et al. (2019) Faber et al. (2020) Luo et al. (2020) Yuan et al. (2020) Bajaj et al. (2021) Lin et al. (2021) Rao et al. (2021) Wang et al. (2021) Yuan et al. (2021)

Table 3.1: Illustration of the most frequently used datasets to evaluate explanation approaches on GNNs. Included are the real-world MUTAG dataset and the synthetic BA-Shapes, Tree-Cycles, Tree-Grids, and BA-2Motifs.

et al. (2021) recently discovered several pitfalls in these synthetic datasets. One of the problems is that the ground truth understanding may differ from the features used by the model. For example, in the case of BA-Shape, the edges belonging to the house motifs are all assumed to be important and therefore required by the model to classify a node to be part of the house. However, Faber et al. (2021) show that a two-layer GNN performs similarly well compared to commonly used three-layer networks. Due to the way GNNs work, a two-layer network would not be able to capture the house structure. Therefore, there must be another discriminating attribute. Indeed, the node degree within the base graph varies compared to the node degree in the motifs. Thus, it is very likely that the features considered ground truth are not the deciding features for the model. Therefore, Faber et al. (2021) present alternative benchmark datasets that attempt to overcome the problems.

Beyond that, we can further note the following problems in the evaluation of GNN explanation methods: In the present study, explanations provided by different explanation methods were compared and evaluated only in terms of their discrete values. This means that they were classified as important and unimportant. However, no ranking of features is examined. Furthermore, to our knowledge, no study compares the importance of edges and nodes. Hence, answering the question of whether it is possible with current explanation methods to determine if node attributes are more important compared to the graph structure or vice versa. In fact, this is very important because the choice of GNN architecture may depend on whether the node structure or the graph structure is more relevant in the data. Finally, it should be noted that most of the research have evaluated the methods using only one architecture.

In our research, we, therefore, intend to first re-implement and revalidate a benchmark dataset from Faber et al. (2021). Furthermore, we introduce new benchmark datasets generated by rules that allow the investigation of continuous explanations as well as the comparison between edge-level and node-level explanations. We also perform evaluations for different GNN types and assess whether the explanation methods perform similarly on all architectures. Further explanations and results of the experiments can be found in Chapter 5.

As preparation for our experiments, we decided to simplify the usability of generic GNN explanation methods and extend the open-source tool PyG, allowing its models to be used with Captum. This is explained in the following chapter.

Chapter 4

Developing Interface Between PyG and Captum

Captum is an open-source interpretability library for PyTorch (Kokhlikyan et al., 2020). It provides a variety of attribution methods that help us build an understanding of the relevance of individual features in the dataset during the prediction process of a model. While many PyTorch models are supported, PyG models cannot be easily applied to Captum. For this reason, we implement a corresponding interface. We first introduce the basic concepts of Captum and PyG in this chapter. We also explain why applying PyG models to Captum fails. Moreover, we explain the interface and how it overcomes the challenges.

We illustrate how Captum works using an example attribution method. Algorithm 1 shows the computation of IG using Captum. For the mathematical definition of IG, see Equation 3.2. The method takes as arguments: an ML model, model inputs, a class for which the relevance values are to be calculated, a baseline, and the number of steps to approximate the integral. Optionally, additional forward arguments can be provided, which are forwarded to the model without calculating relevance values for them. Initially, the approach generates a new input on each step, which lies on the straight path between the baseline and the original input. For each new input, a model pass is performed. The output belonging to the specified class is selected, followed by the calculation of the gradients. In the last step, the new inputs' gradients are summed up.

While the presented attribution technique creates continuous new data inputs, PyG accepts only edge inputs in COO form. The columns represent edges, and the entries in each column correspond to the node indices connected by the edge. Node features are provided in the same way as conventional input data, such as in images. While node features can be modified as in Algorithm 1, the edge input cannot.

To address this, we created a method named `to_captum`. This method creates a new model, called *CaptumModel*, that receives not only node features and edges but also an edge mask. The edge mask is continuous and is applied to each layer of the model. While updating node embeddings within a layer in PyG follows the MessagePassing

Algorithm 1 Captum IG Attribution Method

Input:

model: Model for which attribution scores are being searched for,
inputs: Inputs for which IGs are calculated (can be multiple inputs),
baselines: Starting points for calculating the integral (same shape as inputs),
class: Class for which IGs are computed,
n_steps: Number of steps used to approximate the integral,
additional_forward_args: Additional inputs passed to the model for which no attributions are calculated.

Output:

attributions: IGs with respect to each input feature

Initialize *attributions* with zeros and matrix of size *inputs*.

for *step* = 0 **to** *n_steps* **do**

$step_inputs \leftarrow baseline + \frac{step}{n_steps} \cdot (inputs - baselines)$

$output \leftarrow model(step_inputs, additional_forwards_args)$

$output \leftarrow$ select the *output* associated with *class*

$grads \leftarrow$ get gradients of output with respect to each input feature

$attributions \leftarrow attributions + grads$

end for

Return *attributions*

scheme, as shown in Figure 4.1a, CaptumModel includes an additional step, as shown in Figure 4.1b.

Compared to the general GNN framework presented in Section 2.2, the MessagePassing scheme splits the generation of messages coming from neighboring nodes from the aggregation function. For instance, a message coming from a node might be its normalized node embedding, as in GCN layers. The aggregation is then the sum of all normalized node embeddings (cf. Equation 2.9). Using an edge mask, all messages are weighted prior to aggregation, as shown in Figure 4.1b.

We describe the forward function of the new CaptumModel that uses the edge masks in Algorithm 2. First, the edge mask is set for each layer that follows the MessagePassing schema. Then, prediction is performed with the original model using the updated layers passing node features and edge indices as input. In node-level tasks, since the predictions are made for all nodes simultaneously, the final step involves selecting the node output for which the explanation values are required.

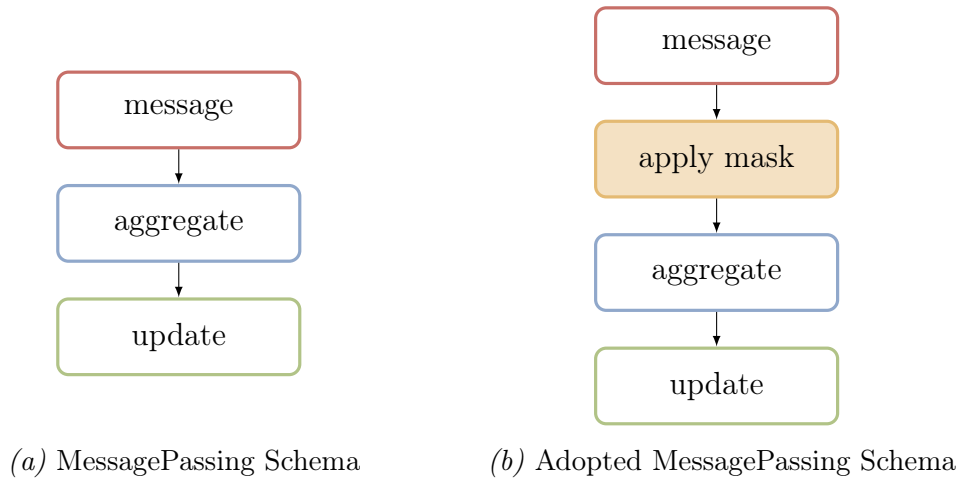


Figure 4.1: A visualization of how the node embedding update schema is extended for being used with Captum. On the left side we see the standard MessagePassing framework, including *message*, *aggregate* and *update* functions. On the right side, we add a step *apply mask* between the *message* and the *aggregate* function to weight the messages.

Overall, when applying a Captum attribution technique to a PyG model, the following steps should be taken:

1. Transform the PyG model to a CaptumModel using the method `to_captum`.
2. Initialize an edge mask that contains ones.
3. Run the attribution method, e.g., IG, on the CaptumModel.
 - (a) Pass node features and edge mask as inputs, pass edge indices as additional forward arguments.
 - (b) Obtain attribution values for each node feature and edge.

Algorithm 2 CaptumModel Forward Function

Input:

node_features: Original node features
edge_mask: Weights for each edge
edges: Original edge information in COO format.

Output:

out: *pyg_model* output, where edges are weighted based on the edge mask at model pass

```

for layer in pyg_model do
  if layer of type MessagePassing then
    Set edge_mask in layer
  end if
end for
output  $\leftarrow$  pyg_model(node_features, edge_index)
if node level task then
  output  $\leftarrow$  select output of node
end if
Return output

```

In order to use the visualization tools provided by PyG for explanations, absolute and normalized attribution values must be calculated. Furthermore, we can use the sum of the node feature relevance values for the entire node importance.

Additionally, it must be noted that Captum offers the option to compute attributions in batch mode; however, this is currently not supported via the CaptumModel.

Besides the variant described here, the CaptumModel also allows the computation of attribution values only for node features or edges. In the former case, simply no edge mask is passed to the attribution function, and in the latter case, the node features are passed as an additional forward argument instead of as input.

A selection of the generic algorithms from Captum, which can also be used for graph models in PyG using the interface developed here, are examined in the following chapter for their performance and compared with GNN-specific explanation algorithms.

Chapter 5

Experiments

In this chapter, we test several generic and GNN-specific XAI techniques on four different datasets and four different model architectures. We evaluate the performance of the approaches in comparison to ground truth explanations. Furthermore, we analyze the runtime of each technique. This chapter is divided into three main parts. In the first part, the general experimental setup is described. Then, per dataset, the specific experimental design and results are presented before finally comparing the runtime metrics.

5.1 General Experiment Setup

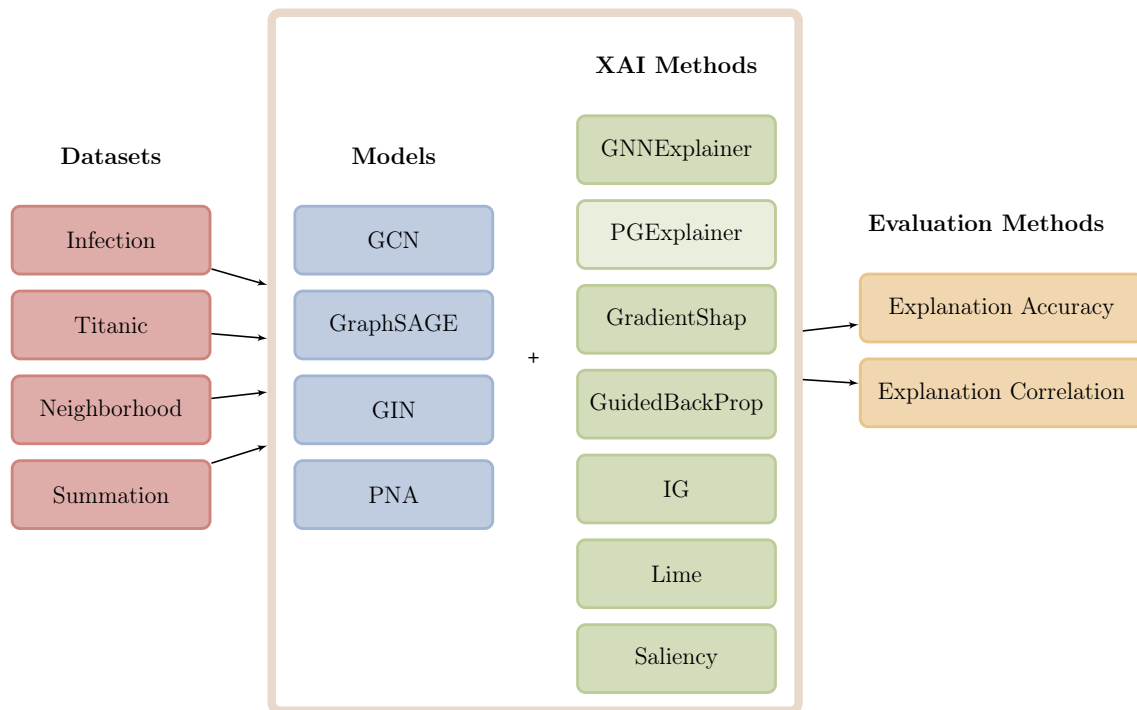
Each experiment in this chapter uses the same GNNs and set of XAI techniques. Furthermore, we use two main evaluation methodologies across experiments. These aspects are described in greater depth subsequently. Figure 5.1 provides an overview.

Datasets

For the experiments, we consider four datasets. The datasets' prediction tasks - classification or regression - are performed at node level. Each dataset is composed of numerous graphs. The nodes in each graphs are categorized into test, train, or validation nodes (60% training, 20% testing, and 20% validation). Chapters 5.2.1, 5.3.1, 5.4.1 and 5.5.1 contain a full description of the datasets.

Models

We train the datasets using the GNN architectures: GCN, GraphSAGE, GIN, and PNA. Because a baseline model exists for two of the datasets, we additionally train those datasets on a regular MLP and a SumGNN model. The number of message-passing layers and the size of the feature maps vary depending on the experiment. To optimize the model parameters, we employ a Bayesian optimizer. Each model was trained for 400 epochs, and the model with the highest validation performance during training was chosen for XAI methods evaluation.



*Figure 5.1: **Experiment Overview.*** For four distinct datasets, we train four models, and for each model, we analyze a large number of XAI techniques. Finally, we assess them based on their explanation accuracy or correlation. PGExplainer is not used in all experiments; hence, it is highlighted lighter.

XAI Methods

This study compares the following explanation methods: GNNExplainer, PGExplainer, Saliency, GuidedBackProp, Lime, IG, and GradientShap. PGExplainer is used only for edge attribution analysis, as it does not provide relevance values for node features. Some strategies provide both negative and positive attributions. However, we only consider their absolute values. Moreover, we compute the relevance of an entire node by summing up its individual feature attributions. In addition, we also normalize the relevance scores to facilitate comparison across different instances and explanation methods. The attributions are rescaled such that the sum of all importance values within one sample equals one.

Evaluation Methods

In addition to visually evaluating the explanation methods for single instances, we primarily benchmark their performance with respect to the ground truth knowledge on each dataset. For one prediction, each explanation method returns an attribution vector $\hat{R} = \{\hat{r}_1, \hat{r}_2, \dots, \hat{r}_n\}$, an attribution \hat{r}_n belongs to an input. Here, an input can be either an edge or a node feature. We compare \hat{R} to a ground truth attribution vector $R = \{r_1, r_2, \dots, r_n\}$ containing ground truth estimates for all input values with respect to a single prediction. While for some datasets, R holds only binary values, for other datasets, we have continuous importance attributes.

For the binary case, $r = 1$ represents an important input and $r = 0$ and unimportant value. We compute an explanation accuracy defined with

$$ExplanationAccuracy = \frac{1}{N} \sum_{i=0}^N \frac{1}{k_i} \sum_{j=1}^{k_i} 1(a_j^i \in GroundTruth^i), \quad (5.1)$$

where N is the number of explanation samples, $GroundTruth^i$ contains the inputs for a sample i , edges and node features, that are considered important according to R , a_j^i is the j -th input in sample i , whereby the inputs are sorted by the attribution vector \hat{R} . Here, a_1^i corresponds to the input with the highest attribution value, and k_i is the size of $GroundTruth^i$.

For the continuous case, where we not only know which edges or node features are considered important or unimportant but also know the exact ranking of the inputs in terms of importance, we compute an explanation correlation with

$$ExplanationCorrelation = \frac{cov(R, \hat{R})}{\sigma_R \sigma_{\hat{R}}}, \quad (5.2)$$

which corresponds to the Pearson correlation between the ground truth attributions R and the relevance values \hat{R} obtained from the XAI methods. Here cov is the covariance, and σ is the standard deviation.

In the following, we elaborate on the particular experiments for each dataset.

5.2 Infection Benchmark: Identify Important Path

Faber et al. (2021) proposed the Infection dataset to address the lack of suitable datasets for evaluating interpretability methods for GNNs. The dataset aims to investigate the relevance values for edges generated by various explanation methods. Specifically, it is intended to determine if the explanation methods can identify an important path within the graph. The study by Faber et al. (2021) utilizes a single GNN architecture type for all experiments. Furthermore, Faber et al. (2021) examine the explanation accuracy on a small number of approaches, such as GNNExplainer and IG. Due to the interface defined in Chapter 5, however, various explanation techniques can now be applied to GNNs without requiring additional implementation effort. Consequently, it is essential to compare the efficacy of these approaches on GNNs, similar to other methods. As a result, in this chapter, we first investigate whether the findings of Faber et al. (2021) can be replicated using the original experimental design. Second, we extend the experimental design by including additional GNN types and explanation methods.

In the following, we outline the experimental setup in detail. We describe both the dataset and the GNNs. After that, we present the model performance and compare the output of the XAI techniques using the trained models.

5.2.1 Dataset and Model

The Infection dataset is composed of 10 graphs with 1000 nodes each. Using the Erdős–Rényi model, these nodes are connected by random edges. The probability of an edge between two nodes is 0.004. Each node carries two features, one corresponding to *infected* and the other to *healthy*. Fifty nodes are marked as infected, and the rest as healthy. The objective of this dataset is to determine the distance to the nearest infected node. Label 0 implies that the node itself is infected, labels 1-4 imply that the nearest infected node is at a distance equal to the label, and label 5 implies that the nearest infected node is more than four nodes away or unreachable. A visualization of the dataset is shown in Figure 5.2.

The special aspect of this dataset is the fact that it contains unambiguous important edges. Each edge that lies along the path between a node to be classified and the nearest infected node is considered important. Removing one of these edges causes the label to be different. On the other hand, removing an edge that belongs not to the shortest path should not affect the result; hence, these edges are regarded as unimportant. Relevance values are sought for all node prediction samples in the test set for which there is a unique shortest path. The distribution of labels within the test dataset and nodes utilized for relevance analysis is shown in Figure 5.3.

In the study by Faber et al. (2021), a SumGNN model was trained with four GNN layers, each having a hidden dimension of 32. Moreover, the model had a final linear layer. This network also employed the jumping knowledge approach, meaning that the output of each layer is combined and passed to the final linear layer. Figure 5.4

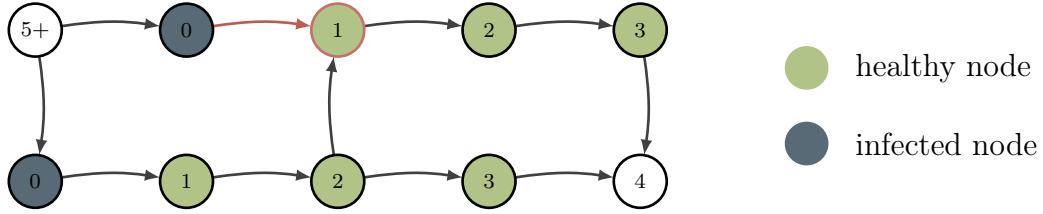


Figure 5.2: Infection Benchmark. For each node, the distance to the nearest infected node is predicted. Unreachable nodes and those with at least a distance of 5 are grouped into one class. The edges along the path from the infected node to the node itself are considered important for the classification, i.e., for the classification of the node marked in red, the edge from the infected node to the node itself is most relevant. Nodes with ambiguous shortest paths to an infected node are excluded (white) for the evaluation of explanation methods.

■ Test Nodes ■ Test Nodes for Explanation Evaluation

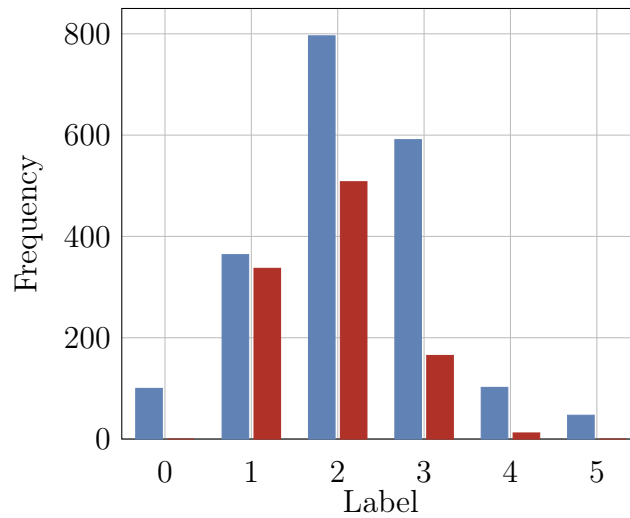


Figure 5.3: Label distribution for the entire test set and the selection of test nodes used for XAI method evaluation.

illustrates this kind of model architecture. For comparison purposes, we replicate the model proposed by Faber et al. (2021), besides training GCN, GIN, GraphSAGE, and PNA models according to the same structure. We employed a learning rate for all models of 0.005.

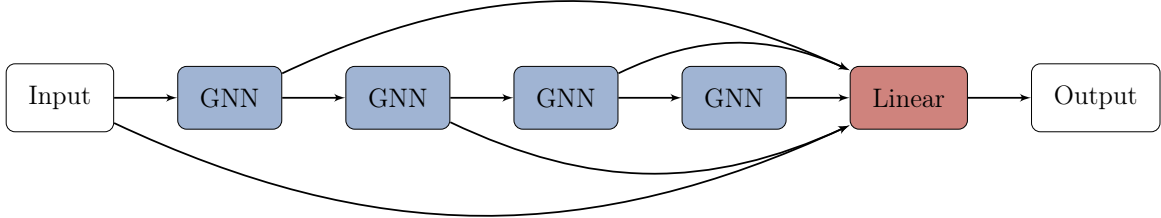


Figure 5.4: A general 4-layer GNN with a final linear layer and jumping knowledge. The GNN layers can be either SumGNN, GCN, GraphSAGE, GIN, or PNA.

The following section describes the findings on XAI methods utilizing the infection dataset.

5.2.2 Results

Table 5.2 summarizes the model performance on each architecture. Faber et al. (2021) achieved 100% accuracy with the SumGNN model. We were able to reproduce this accuracy. GCN, GraphSAGE, GIN, and PNA also achieved accuracies between 99.80% and 99.95%.

Table 5.1: Classification accuracy (in %) across different graph model architectures. All GNNs perform well on this task as they reach accuracies close to 100%.

	SUMGNN	GCN	GRAPHSAGE	GIN	PNA
TRAINING	100.00	99.80	99.90	100.00	99.95
VALIDATION	100.00	100.00	99.90	100.00	100.00
TESTING	100.00	99.80	99.90	99.90	99.95

For each of those trained models, we compute relevance scores for all edges using the XAI methods: GNNExplainer, PGExplainer, GradientShap, GuidedBackProp, IG, Lime, and Saliency. We select the edges based on the highest relevance score and compare those to the edges along the shortest path. We expect them to coincide, which we measure by explanation accuracy. The overview of the explanation accuracy is shown in Table 5.2.

We begin by comparing our results to the work of Faber et al. (2021). We achieve 99.54% explanation accuracy for IG, which is comparable to the results of Faber et al. (2021). The GNNExplainer method achieves a higher score of 58.68% when compared to the original work’s score of 32%. This could be due to different underlying versions

Table 5.2: **Explanation accuracy (in %) across different graph model architectures and explainability methods.** Explanation accuracy reported in Faber et al. (2021) are given in (\cdot). GradientShap performs best across all models.

	SUMGNN	GCN	GRAPHSAGE	GIN	PNA	MEAN +- STD
GNNEPLAINER	58.68 (32)	29.68	30.32	46.36	80.10	48.93 \pm 21.41
PGEXPLAINER	9.52	0.29	6.14	9.36	5.55	6.17 \pm 3.75
GRADIENTSHAP	99.83	96.94	89.99	98.71	99.51	97.00 \pm 4.07
GUIDEDBACKPROP	99.71	87.20	93.31	52.53	99.32	86.41 \pm 19.62
IG	99.54 (100)	94.67	90.47	99.02	99.44	96.62 \pm 3.99
LIME	54.97	81.91	46.49	77.13	99.51	67.24 \pm 15.66
SALIENCY	99.71 (74)	98.28	85.99	98.75	99.41	96.43 \pm 5.86

of the GNNExplainer implementation in PyTorch. Additionally, hyperparameters that were not specified in the original work might have been selected differently.

We proceed to compare GNN-specific methods with generic methods. Considering the performance across all models, all generic methods have higher explanation accuracy than the GNN-specific methods. While GNNExplainer detects only half of the ground truth edges with an average performance of 48.93% \pm 21.41%, PGExplainer can detect almost no important edges with an average performance of 6.17% \pm 3.75%. One possible explanation for this could be the method class: While the GNN-specific methods are both perturbation-based, all other techniques fall into surrogate- or gradient-based methods. A known problem with perturbation-based methods is the generation of samples outside the training distribution; thus, predicting those samples may not be reliable.

Overall, we see that GradientShap performs best across all models with a mean explanation accuracy of 97.0 % \pm 4.07%. Thus, it detects almost all edges that are considered important. Also, IG and Saliency achieve a similar score. Guidedbackprob and Lime detect fewer edges correctly, with an average of 86.41% \pm 19.62% and 67.24% \pm 15.66%, respectively. Although Lime is not considered one of the perturbation-based methods in this work, it uses perturbations in the locality of an explained instance to generate explanations, which could be the reason for the lower performance.

In Appendix A.1, we investigate the explanation accuracy of each approach and GNN grouped by the distance of the edge from the node.

Figure 5.5 shows the importance of each edge in a classification example using the best-performing method on SumGNN with GradientShap. GradientShap can correctly recognize the path between the infected node and the prediction node as important. While we also see that the edges belonging not to the path are assigned relatively low importance scores.

Additionally, we review the results of PGExplainer on another example; see Figure 5.6. We find that PGExplainer assigns high importance to all edges leaving an infected node, regardless of whether this infected node is the closest. This can be due to

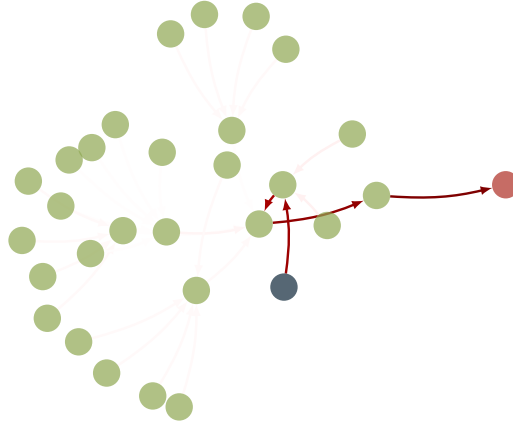


Figure 5.5: Evaluation of explanations for a single instance with GradientShap. Shown is an example subgraph for node classifications using the SumGNN model. The node to be classified is highlighted in red, while the next infected node is marked in grey. The darker the edge, the more relevant it is based on GradientShap.

the design of the methodology. PGExplainer is trained using multiple instances. We hypothesize that the model tries to find important characteristics for all examples and has difficulties identifying only those features being important for the single instance instead of for all samples.

In summary, we find that some explanation techniques do exceptionally well at identifying the most important path. GNN-specific approaches perform much worse than generic ones. In the following sections, we will examine whether comparable results can be observed for more datasets. We also like to study the relevance of node features and entire nodes.

5.3 Titanic Benchmark: Identify Important Node and Noise

While the previous experiment allowed us to examine the efficacy of various XAI approaches in locating relevant graph substructures, the input to a GNN also includes node information. Consequently, it is crucial to comprehend which algorithms reliably detect the relevance of node inputs. Moreover, graph datasets differ in the quantity of valuable information for prediction regarding graph structure and node attributes. While the graph structure contains most of the information required for prediction in some datasets, the essential attributes are contained in the nodes in other datasets. Depending on the situation, specific GNN topologies are more suitable. Therefore, we first conduct an experiment to determine the XAI techniques' capacity to distinguish between a node carrying all relevant data and a graph structure containing only noise. We developed the Titanic benchmark dataset with the mentioned characteristics.

Following is a description of the detailed experimental setup. Both the dataset and

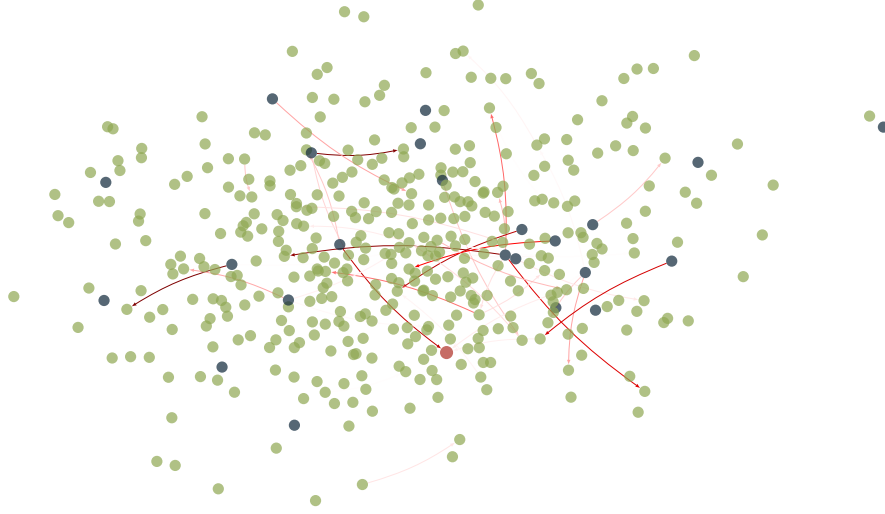


Figure 5.6: Evaluation of the explanation with PGExplainer of a single node classification instance with target 1. Shown is the subgraph used for classification. The node to be classified is highlighted in red, while all infected nodes in the subgraph are highlighted in grey. The darker the edge, the more relevant it is, according to PGExplainer.

the GNNs are thoroughly explained. Then, we present the performance of the trained models and compare the outcomes of the XAI strategies.

5.3.1 Dataset and Model

The Titanic Benchmark dataset is based on the OpenML¹ Titanic dataset. This dataset includes information on which passengers survived the Titanic shipwreck. It contains details such as age, number of siblings and spouse (Sibsp), number of parents and children (Parch), fare, gender, class booked, and boarding location for each passenger. Some data points are missing; hence, we use the average value as a substitute. The entire preprocessing is based on an implementation in Captum². On the basis of this dataset, we construct a graph. Here, each node includes information on one passenger, and that is to be categorized based on the label in the OpenML titanic dataset. To generate noise, we randomly connect the nodes with a probability of 0.004 using the Erdős-Rényi model. We generated ten distinct graphs, each with the same passengers as nodes but with a unique graph structure. Across all graphs the same nodes were assigned as test, training, or validation nodes. Due to the noise in the graph structure, the model should only rely on the information in the node itself for the prediction. The dataset is represented visually in Figure 5.7.

For the graph dataset, we employ models with four graph layers and one final linear

¹OpenML is a platform for sharing machine learning data, algorithms, and experiments (<https://www.openml.org/>).

² An example of interpretability with Captum using Titanic survival data (https://captum.ai/tutorials/Titanic_Basic_Interpret).

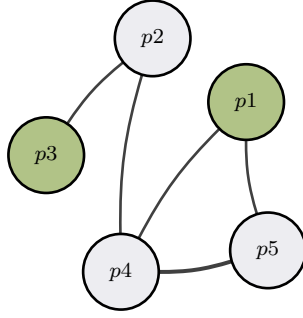


Figure 5.7: Illustration of the Titanic graph. Each passenger from the original Titanic dataset represents a node, and all nodes are randomly connected to add noise. The task is to classify the nodes into survivors (green) and non-survivors (gray) based on the node feature information.

layer. We use a learning rate of 0.025 for training. In contrast to the previous experiment, no jumping knowledge is added as input for the final linear layer. Figure 5.8 illustrates the structure of such GNN with n -layers. For comparison, we also train a non-graph-based model with three linear layers, which follows the structure in the Captum example.

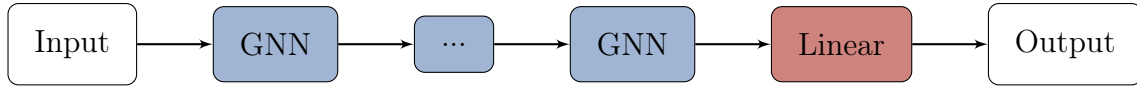


Figure 5.8: A general n -layer GNN that includes a final linear layer. The GNN layers can be either of type GCN, GraphSAGE, GIN, or PNA.

Having defined the dataset and the GNNs, we can discuss the performance of each DL model and XAI method.

5.3.2 Results

Table 5.3 summarizes the performance of the models in terms of their accuracy. Due to the dataset’s imbalanced nature, we can reach 58.62% in accuracy for test nodes by simply assigning them all to the main class. We achieve 81.61% in accuracy with a conventional MLP, which is similar to the accuracy reported by Captum. Furthermore, we attain comparable performance with GraphSAGE and PNA, achieving an accuracy of 78.48% and 79.66%, respectively. On the other hand, GCN and GIN do not cope as well with this problem and achieve an accuracy of about 69%, which is still above that of the main class baseline. The difference in performance is most likely due to the layer design. While PNA and GraphSAGE have two weight matrices per layer, one for the neighboring nodes and the other for the node over which aggregation is performed, GCN and GIN have only one weight matrix. This leads to difficulties with respect to the simultaneous exploitation of all information from the main node while ignoring the neighboring nodes. When analyzing the importance values of node features compared to edges, we expect to see differences depending on the model.

Table 5.3: Classification accuracy (in %) on the Titanic dataset. Included are the models MLP, GCN, GraphSAGE, GIN, and PNA, as well as the performance when assigning all nodes to the main class. PNA and GraphSAGE have similar accuracy to the MLP model.

	MLP	GCN	GRAPHSAGE	GIN	PNA	MAIN CLASS
TRAINING	81.83	70.83	80.17	72.40	81.37	63.28
VALIDATION	80.84	69.81	81.49	71.80	81.19	60.54
TESTING	81.61	68.85	78.47	69.58	79.66	58.62

We first analyze the distribution of attribution scores among edges and nodes, distinguishing between the node to be classified and its neighboring nodes. We anticipate that the XAI methods assign higher importance to the node being categorized than to the other nodes. We also expect the majority of edges to be deemed irrelevant. This is indicated by the proportion of relevance for an edge being close to 0. Figure 5.9 shows the distribution of relevance scores obtained by the XAI methods.

Initially, we examine the GNNExplainer. We note that there is no clear distinction between the classified node and the other nodes. This supports the first experiment’s results that the perturbation-based approach GNNExplainer may not be suitable for the interpretation analysis of GNN models.

Analyzing the remaining XAI methods on the top-scored model, which is PNA, we find a clear difference in the assignment of relevance values for the node to be classified, its neighboring nodes, and the graph structure. In particular, the distributions of relevance scores produced by Saliency, IG, and GuidedBackProp appear to be quite similar. In the majority of instances, the main node has over 40 percent of the relevance appointed. Most edges and neighboring nodes are assigned almost minimal weight. However, according to those explainers, a few noise attributes are nevertheless deemed important. We assume that although the edges in our dataset are randomly generated, they cannot be completely disregarded by GNNs. Lime also assigns high attribution values to a few edges and nodes that are close to the node of interest, although mostly only the node to be classified is considered crucial. Moreover, it is noticeable that the proportion of attribution to the main node seems to be lower in Lime than in all other techniques except GNNExplainer.

Analyzing the performance on the least accurate model, GCN, we find that Saliency, IG, and GradientShap show similar distributions. Although the relevant nodes still account for more than 25 percent of the total importance, the difference is not as high as in PNA or GraphSAGE. Moreover, edges and neighboring nodes are also assigned high attribution values. GuidedBackProp shows a slightly different distribution, where edges are assigned the most importance. The difference between the models with high accuracy compared to those with lower accuracy is consistent with our expectations. We do not find this to be the case with GNNExplainer.

We also evaluated the explanation accuracy of the node features. In detail, we compared the most important node features according to the XAI methods with those

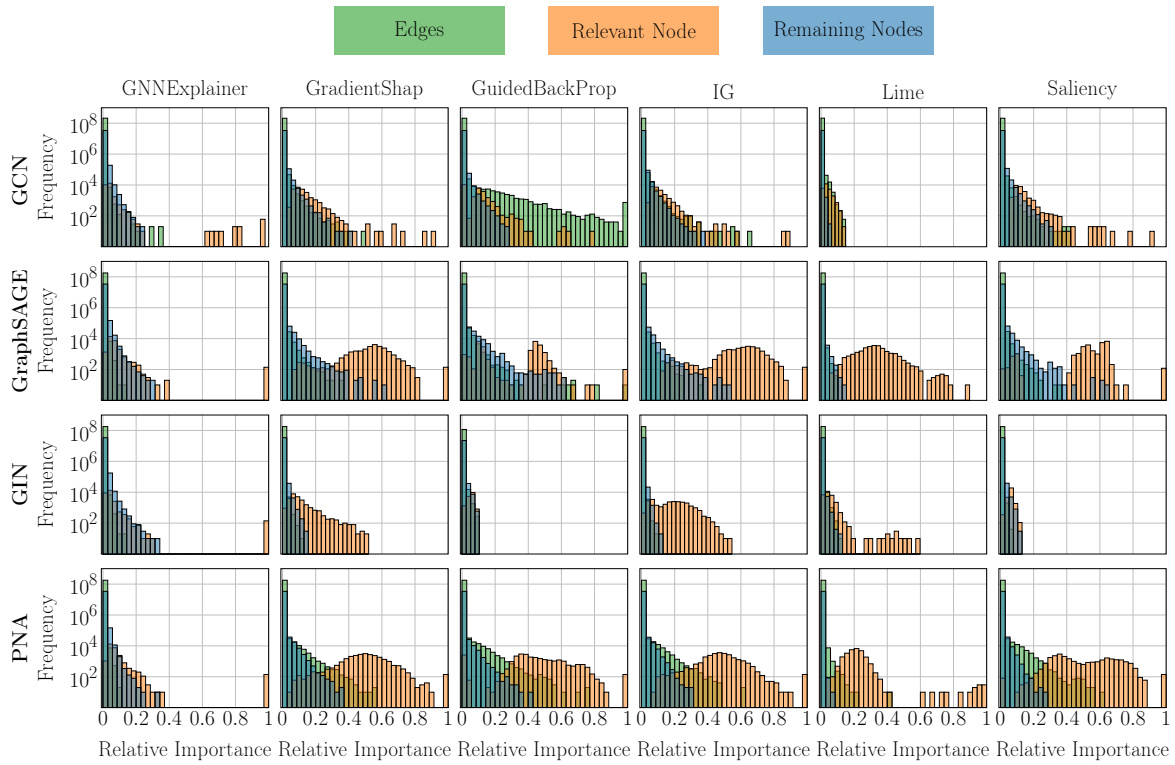


Figure 5.9: Distribution of attribution scores among the relevant nodes, i.e., the nodes to be classified, the remaining nodes, and the edges. Here, the XAI methods, GNNExplainer, GradientShap, GuidedBackProp, IG, Lime, and Saliency are compared in providing explanations for the models GCN, GraphSAGE, GIN, and PNA.

belonging to the key node. The results can be found in Appendix A.2.

Another aspect we can consider in more detail is the importance of the individual features. Using the Captum example, we have an indication of features that might be important or less important in classifying a passenger. According to Captum, essential features are age, gender (female or male), 3rd class, and fare. GNNs may use different features for classification, but we expect to find overlapping important features across the XAI methods. We see in Figure 5.10 that features similar to those described by Captum are detected by all explanation methods, e.g., female and male. We note that GradientShap, IG, and Lime consider similar features to be important. It is also noticeable that Saliency and Guidedbackprop consider the characteristic age and fare less important than other node features when compared to the other approaches. Both node features are continuous values, indicating that Saliency and Guidedbackprop are proficient at detecting the importance of binary/categorical values but have difficulties detecting the importance of continuous numerical features. We further investigate this in Subsection 5.5.1.

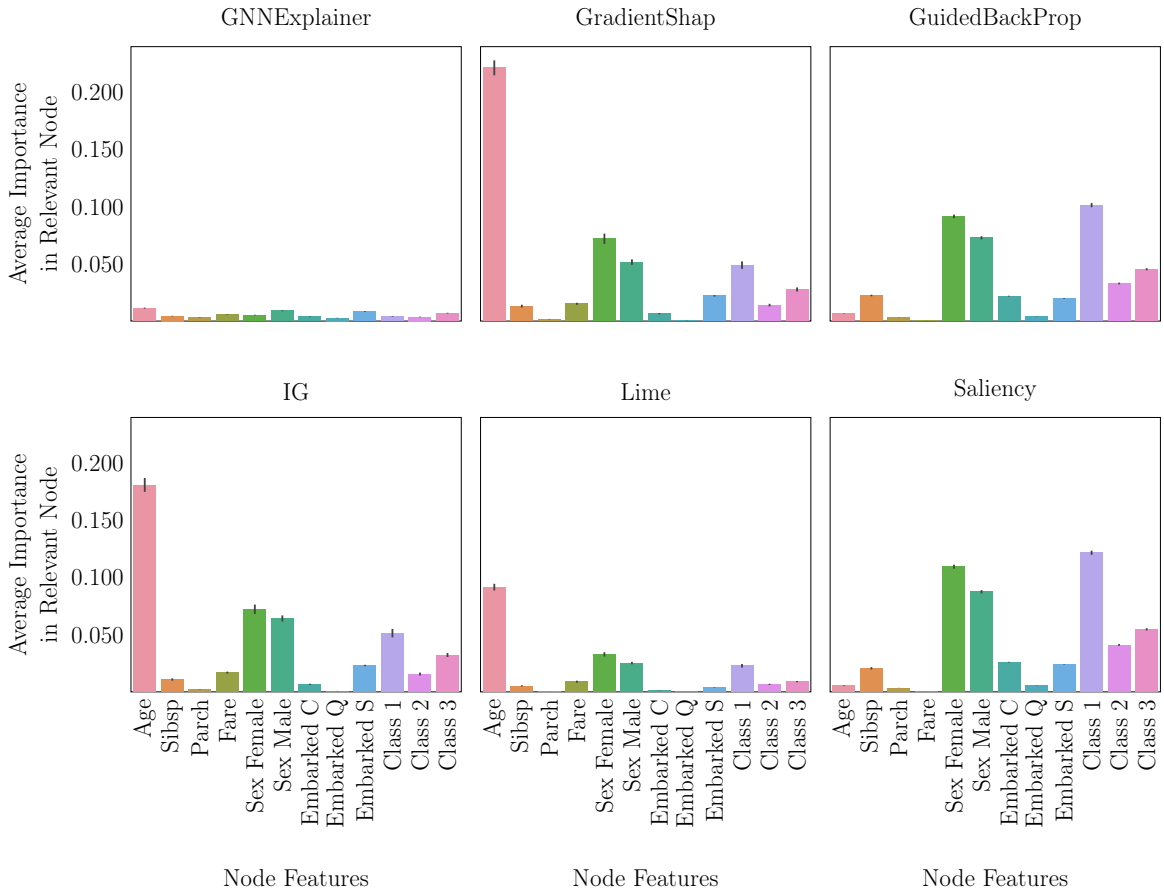


Figure 5.10: Relevance of node features belonging to the main node. We compare the allocation of relevance to the node features with GNNExplainer, GradientShap, GuidedBackProp, IG, Lime, and Saliency for the PNA model.

In summary, the Titanic Benchmark shows that generic XAI methods assign higher values to the relevant nodes than to the graph structure. We observe that this ability

changes with the model’s performance along all methods. The distinction in importance between the relevant node and noisy graph structure is not as explicit in GNNExplainer as in other methods. Moreover, we find that Saliency and GuidedBackProp assign low relevance to features with continuous values and therefore consider other features as more important compared to Lime, IG, and GradientShap.

5.4 Neighborhood Benchmark: Identify Important Edges and Noise

Using the Infection and Titanic benchmarks, we investigate and contrast various XAI algorithms in terms of their ability to identify important edges and noise in edges relative to essential node information. However, these tests are limited in the aspect that we exclusively evaluate the performance of binary importance; that is, we categorize an edge or node as either important or unimportant. In contrast, in real-world datasets, ground truth relevance values might be instead continuous. It is challenging to establish such continuous ground truth attribution values for many types of datasets. For graph datasets, however, we can develop those when using a rule to establish a prediction task. In addition, to ensure that a method can detect important graph structures and noisy nodes, we perform an experiment similar to the Titanic benchmark with inverted importance distribution: The nodes contain only noise, whereas the graph structure contains information crucial to the prediction. We generate a dataset called Neighborhood following these criteria and compare the distribution of node and edge relevance. Moreover, rather than quantifying the explanation accuracy, we measure the correlation between the attributions of all edges provided by the XAI algorithms and the ground understanding of the synthetic dataset in order to enhance the search for continuous trustworthy explanations.

The complete experimental setup is described below. Both the Neighborhood dataset and the GNNs utilized are discussed. Moreover, we present the models’ performance and compare the results for each XAI technique.

5.4.1 Dataset and Model

The Neighborhood benchmark dataset includes ten graphs with 1000 nodes each. Unlike previous datasets, these Neighborhood graphs consist of random trees. Here, the maximum node degree is ten, and the maximum depth is four. The node features are random numbers. Furthermore, the task is a node regression task in which the target of each node equals the number of nodes in its four-hop Neighborhood. Figure 5.11 illustrates a sample in the neighborhood dataset. Besides Figure 5.12 presents the target distribution. For obtaining the target, just the graph’s topology is relevant, not the node features. We can calculate the true relevance of each edge given the particular graph structure by determining how much the result changes if an edge is removed. We assign this difference in the result as an edge’s importance value; thus, the greater the difference, the more essential the edge.

GNN-wise, we use models that differ in their number of layers and dimensions. The

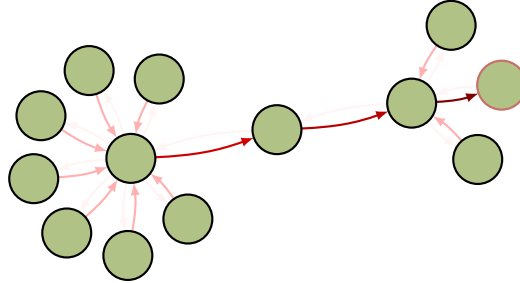


Figure 5.11: Neighborhood Benchmark. For each node, the number of neighbors within four hops is predicted. Each edge’s importance is calculated by the change in the actual target that would result from removing that edge from the graph. The relevance when predicting the target for the node marked in red is highlighted in red. The true label of this node is 13. The edge is considered more important the darker it is.

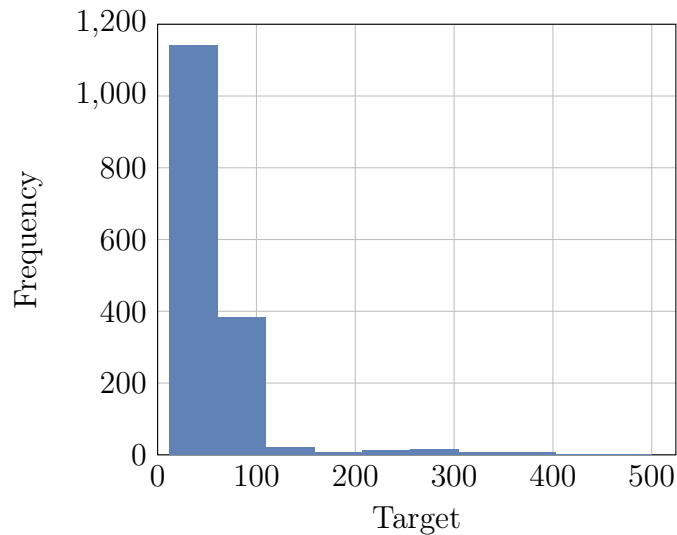


Figure 5.12: Target distribution for all nodes in the test set on the Neighborhood benchmark. While the majority of nodes have a target between 0 and 100; there are also a few nodes with targets up to 500.

GCN used has six layers with a hidden dimension of eight, GIN has five layers with a hidden dimension of 64, GraphSAGE has six layers with a hidden dimension of four, and PNA has four layers with a hidden dimension of 16. Furthermore, all models feature a final linear layer. The structure of the GNNs is similar to the models used in the Titanic benchmark, shown in Figure 5.8. We use a learning rate of 0.05 for all models.

Following the introduction of the dataset and the GNNs, we evaluate the models' performance and the relevance score generated by various XAI methods on those models.

5.4.2 Results

Table 5.4 provides a summary of the model performances in terms of their mean absolute error (MAE). As a simple baseline, the average target is assigned as a prediction to each node, called the Mean model. On the test set, we achieve an MAE of 24.28 for this baseline. The score indicates that the average variance between the correct and mean target equals 24.28. The more accurate the model, the lower the MAE score. Using the PNA model, we attain an MAE of 0.54. Whereas with GIN, we register an MAE of 0.94, GraphSAGE performs similarly with an MAE of 1.09. GCN has the lowest performance values, as indicated by its high MAE of 13.24.

Table 5.4: MAE on the Neighborhood dataset. Included are the models GCN, GraphSAGE, GIN, and PNA, as well as the performance when the mean of all targets is assigned as prediction to all nodes, called Mean Model.

	GCN	GRAPHSAGE	GIN	PNA	MEAN MODEL
TRAINING	13.43	1.10	0.86	0.49	24.29
VALIDATION	13.93	1.51	1.06	0.48	26.17
TESTING	13.24	1.09	0.94	0.54	24.28

First, the XAI approaches are analyzed on the explanation correlation. An explanation correlation value of 1 implies a linear relationship between the attribution generated by the XAI methods and the computed ground truth. For the best-performing model PNA, we expect a value close to 1, as it captures the neighborhood number correctly and, therefore, needs to consider the edges according to our ground truth understanding. Due to the variation in model performance, we also assume a difference in the explanation correlation on other models.

The findings are summarized in Table 5.5. Using the Neighborhood dataset, we find that GNNExplainer again delivers inferior results compared to generic techniques. In addition, we discover that the explanation correlations measured for GCN are low compared to those of other models. This is consistent with the model's performance. On the other side, we see a high correlation for GraphSAGE, GIN, and PNA, with Saliency attaining the most remarkable correlation of 0.9563 on PNA. Thus, a linear relationship exists between the relevance values obtained from the dataset's rules and the values provided by the XAI methods.

Table 5.5: Explanation correlation for the Neighborhood dataset across different graph model architectures and explanation methods. With Saliency, the highest correlation can be reported.

	GCN	GIN	GRAPHSAGE	PNA
GNNEPLAINER	0.0740	0.1382	0.0906	0.1533
GRADIENTSHAP	0.3831	0.7096	0.8550	0.9284
GUIDEDBACKPROP	0.4377	0.8922	0.8650	0.9076
IG	0.4603	0.7925	0.8991	0.9536
LIME	0.2733	0.7144	0.6367	0.8169
SALIENCY	0.3229	0.9542	0.9418	0.9563

We also examine the explanation correlation when plotted against the distance between the edges and the prediction node, which can be found in Appendix A.3.

We evaluated the XAI algorithms’ ability to correctly detect relevant edges, as assessed by the correlation to known ground truth. However, we comprehend not only the importance of edges but also the importance of nodes. Since we know they contain simply noise, we anticipate that they will be less informative for the prediction than edges.

Figure 5.13 demonstrates the attribution distribution among edges and nodes. We begin by assessing GNNExplainer’s attributions. First, we see that no input is allocated more than 10% of all importance. This is lower when compared to the maximum attribution value of the other approaches. This shows that GNNExplainer’s attributions are not as expressive as those provided by the other techniques. Furthermore, there is no apparent differentiation between the relevance of nodes and edges, which is consistent with the preceding sections’ findings. Secondly, we consider the methods attributions for the model with the best performance, PNA. GNNExplainer is not included in this comparison. It is apparent that a subset of edges is given more importance in comparison to the nodes in all XAI methods. This shows that these approaches can detect instances where the graph structure is considered more relevant than the node features. The model with the lowest performance accuracy is GCN.

In summary, we found with the Neighborhood dataset that generic explanation approaches not only find the most relevant edges with high accuracy but also attribute high correlations with the ground truth when the model has a high-performance score. The GNNExplainer, on the other hand, lacks this capability. Furthermore, with the exception of the GNNExplainer, we demonstrate that the XAI technique can detect noise in nodes by assigning them lower attributes than individual edges in the graph structure.

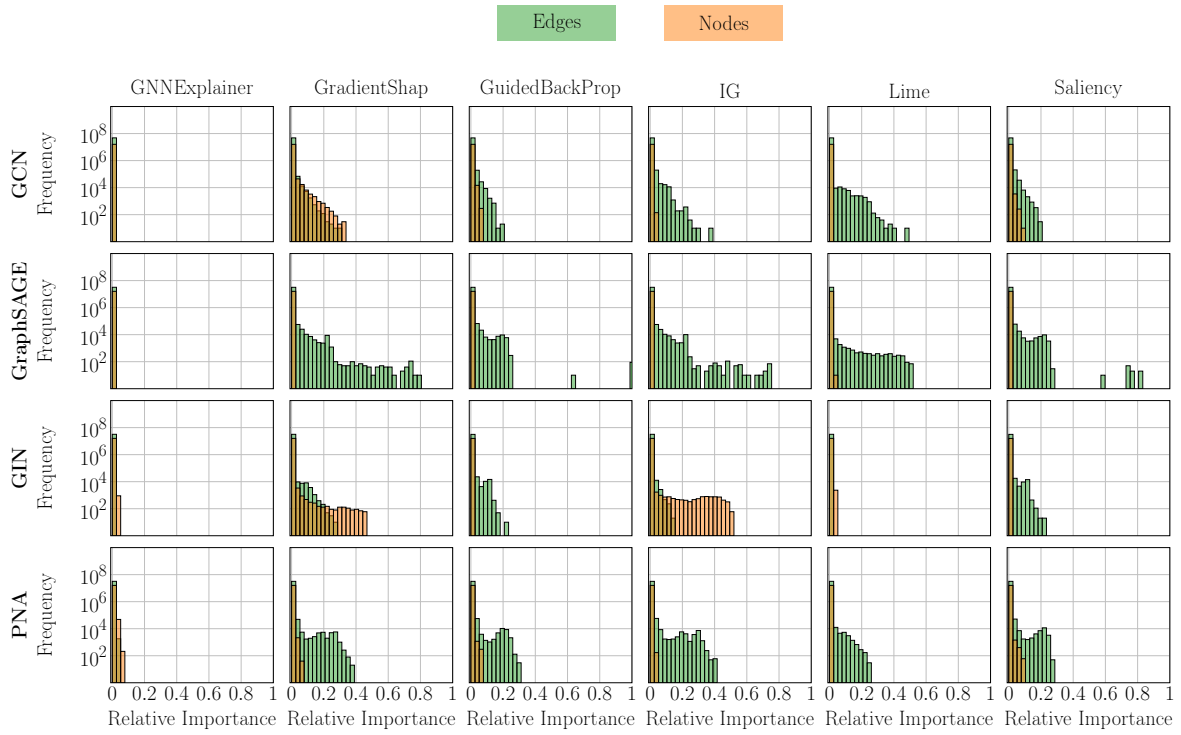


Figure 5.13: Distribution of attribution scores among the nodes and the edges for the Neighborhood Benchmark. Here, the XAI methods, GNNExplainer, GradientShap, GuidedBackProp, IG, Lime, and Saliency, and a are compared in providing explanations for the models GCN, GraphSAGE, GIN, and PNA.

5.5 Summation Benchmark: Identify Important Node Features and Noise

In the experiments presented in the preceding chapters, we investigate the ability of XAI methods to identify relevant or irrelevant edges and entire nodes. In addition, we investigate the attributions of individual node features in the Titanic Benchmark. However, no ground truth information on node features is available for the Titanic dataset. For this reason, we create a fourth benchmark called Summation. This experiment investigates the contribution of the node features in the prediction. More specifically, we compare the attributions of these features according to XAI methods with the ground truth of the Summation dataset.

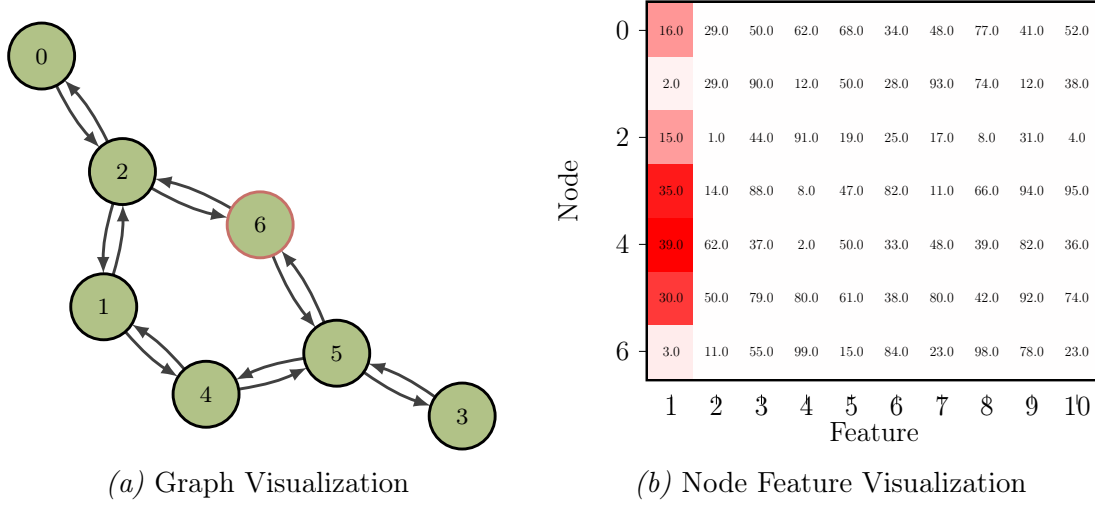
Following is a detailed explanation of the experimental setup. The Summation dataset and the GNNs employed are discussed. Besides, we present the performance of the GNNs and compare the results obtained using various XAI methods.

5.5.1 Dataset and Model

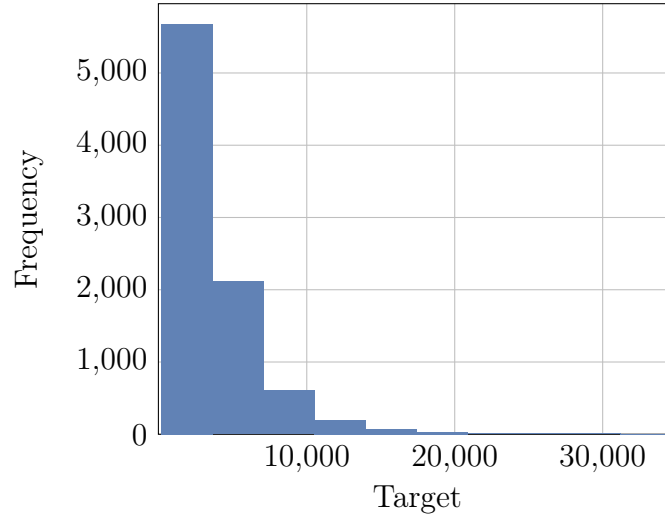
To study the attribution of node features, we need a dataset for which we know how much each node feature contributes to the prediction. Thus, we create a dataset with the task of calculating the sum of node features from all neighboring nodes within two hops. More specifically, we consider only one node feature of each node for computing the sum. We call this node feature the key node feature. Figure 5.14 shows an example graph. First, we know that all node features, other than the key feature, are irrelevant to the prediction. Second, we know that in most cases, the ground truth of the key feature is the same as the feature value. This is because if we scale this feature to zero, the target deviates by exactly the amount associated with the feature. We construct a dataset of 100 graphs with varying numbers of nodes (between 100 and 1000). The graph structure is randomly constructed using the Barabási–Albert model. Moreover, each node has ten features, where the first feature is the key feature. We inspect, on the one hand, the distribution of attributions along all node features and, on the other hand, the correlation between the attribution of the key node feature and its expected relevance score. It is important to note that we do not expect an explanation correlation of 1, as there might be branches in the graph that lead to a repeated contribution of a key feature to the target. However, the number of such cases is negligible. The target distribution of the dataset is shown in Figure 5.15.

Similar to the other experiments, we train four different GNN architecture types on this dataset. The models contain two graph layers with a hidden dimension of 14 and a final linear layer. The structure of the GNNs is identical to the models used for the Titanic and Neighborhood datasets, as shown in Figure 5.8. For all models, we use a learning rate of 0.05.

After the dataset and GNNs are presented, the models and the relevance value created by the XAI approaches can be evaluated.



*Figure 5.14: **Summation Benchmark.*** The sum of the first features of all surrounding nodes that are no more than two hops distant is computed for each node. This instance entails predicting the node shown in red, which has a target value of 140. Each node’s number corresponds to its index. The ground truth importance of the first feature is determined by its value, whereas the other features do not carry any predictive information. The darker the feature, the more important it is.



*Figure 5.15: **Target distribution for all nodes in the test set on the Summation benchmark.*** While most nodes have a target between 0 and 5000; there are also a few nodes with targets up to 30000.

5.5.2 Results

Table 5.6 provides a summary of the model results. As a baseline, we assign each node the average target. This results in an MAE of 2260.60 on the test set. Even if GCN performs better, the test MAE of 1371.13 is still high compared to the other models. GIN and GraphSAGE achieve lower scores and, therefore, better results. With an MAE of 146.34, PNA shows the highest performance. This corresponds to an average deviation from the target value of 4%.

Table 5.6: MAE on the Summation dataset. Included are the models GCN, GraphSAGE, GIN, and PNA, as well as the performance when the mean of all targets is assigned as a prediction to each node, called Mean Model.

	GCN	GRAPHSAGE	GIN	PNA	MEAN MODEL
TRAINING	1337.35	198.58	165.29	135.12	2141.53
VALIDATION	1367.90	208.88	165.36	133.43	2279.39
TESTING	1371.13	215.84	178.83	146.34	2260.66

The first question we ask is whether the XAI algorithms can detect that the key node features are the most relevant node inputs while the others contain only noise. Figure 5.16 shows the distribution of relevance among node features, including all nodes within the 2-hop neighborhood of the prediction nodes. For GradientShap and IG, we find that the difference between the average importance of the key feature and all other features is the largest among all XAI methods for all models. Those two explanation methods demonstrate the expected pattern of results. The gap in relevance along the features indicates that all GNNs obtain most information for the prediction from the first feature. For GuidedBackProp and Saliency, we observe low attributions scores for all features. However, we see a difference when comparing the average importance of the first feature with that of the other features. Low attribution values for Saliency and GuidedBackProp for continuous inputs are also found in Figure 5.10. The definition of those methods is that an input is important when a small change in its value significantly changes the outcome. However, for this dataset, we have a rather simple linear correlation between input and output. This might be a reason why the values are comparatively small. With Lime, we also note little variations in the distribution of feature relevances, although these are not as large as in the other methods mentioned so far. With the parameters chosen, we assume that Lime generally is less expressive than the other methods when assigning importance. This is because we observe similar results in the Titanic results in Figure 5.10. Moreover, for the Neighborhood dataset, we note this concern in Figure 5.13. Lime depends largely on the parameter that defines the neighborhood size, which might be a factor for the expressivity of the method. Another reason for the weaker performance could be that Lime uses a linear model on a small locality around the input to find explanations. However, a GNN might be even highly nonlinear in this small locality. Finally, we also examine the results of the GNNExplainer. We find that the expected results are not obtained, just as in previous experiments.

We also ask whether the attributions generated by the XAI algorithms for the key

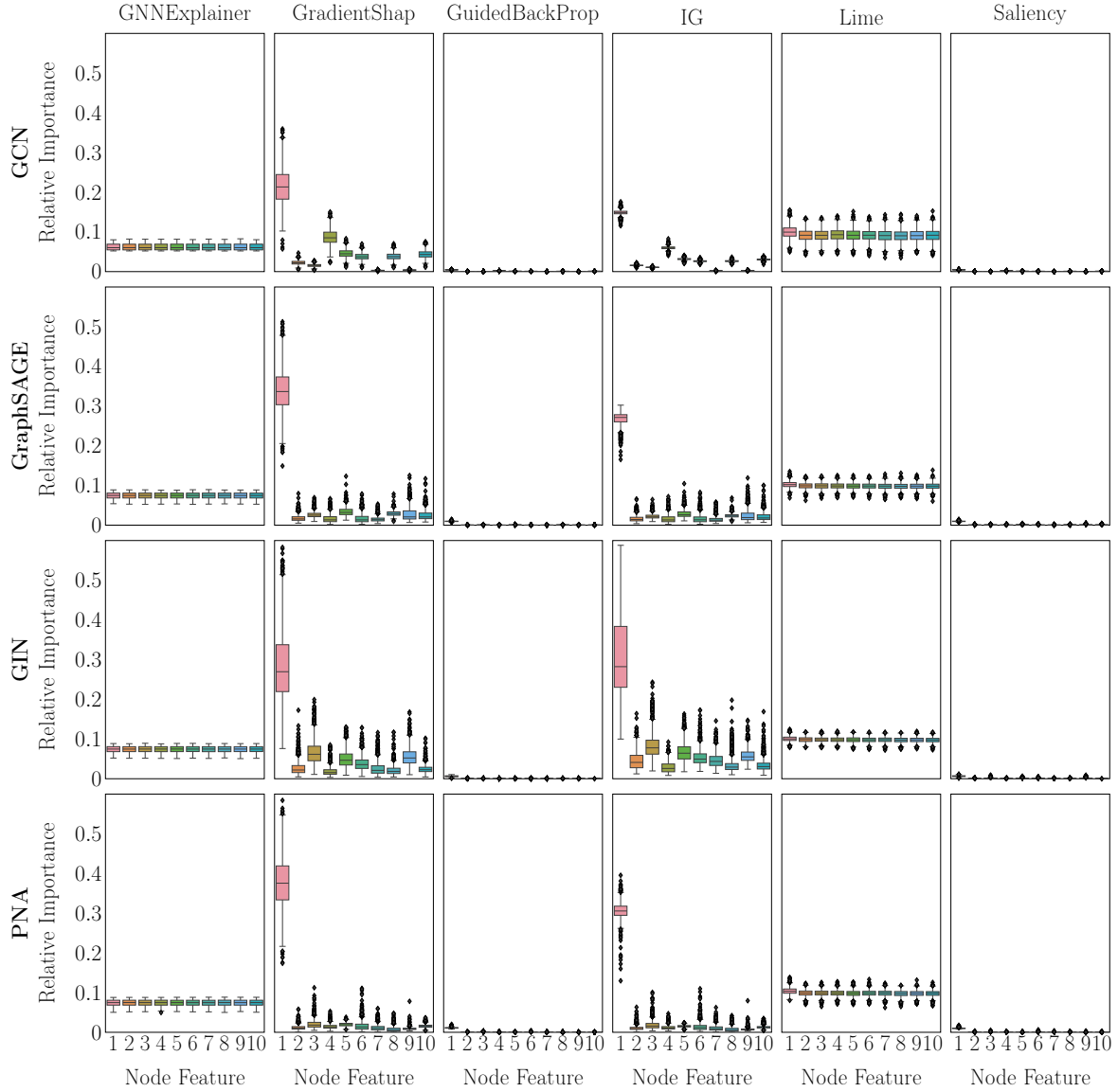


Figure 5.16: Distribution of relative importance for each feature. Here we consider only those nodes in the two-hop neighborhood of the test instance. We compare the results of the XAI methods GNNExplainer, GradientShap, GuidedBackProp, IG, Lime, and Saliency with the models GCN, GraphSAGE, GIN, and PNA.

feature correlate with the feature values. Table 5.7 provides the explanation correlation values for each GNN model and explanation approach. High correlation values can only be obtained with GradientShap and IG. Although the GIN model is second best at solving the ML problem, no explanation approach applied to this model provides explanations that match the ground truth. Figure 5.17 shows the expected relevance values and the obtained one using GradientShap for one instance. We see that GIN also assigns high importance to some features whose feature values are among the highest. One reason for the low correlation could be that GIN only approximates the sum, e.g., by using the node degrees as well as the maximum and mean node feature values instead of calculating the sum for all values. For Saliency, GuidedBackProp, Lime, and GNNExplainer, we do not see the desired correlation for any model. For the generic methods, the cause can be related to the factors described in the previous paragraph.

Table 5.7: Explanation correlation for the Summation dataset across different graph model architectures and across multiple explanation methods. With GradientShap, the highest correlation can be reported.

	GCN	GIN	GRAPHSAGE	PNA
GNNEXPLAINER	-0.001664	-0.002418	0.004641	0.003874
GRADIENTSHAP	0.350897	0.004624	0.404445	0.674790
GUIDEDBACKPROP	0.018180	0.025298	0.010494	0.026048
IG	0.368840	0.079039	0.500082	0.615169
LIME	0.044189	0.018089	0.031062	0.033799
SALIENCY	0.018180	0.020300	0.008693	0.025107

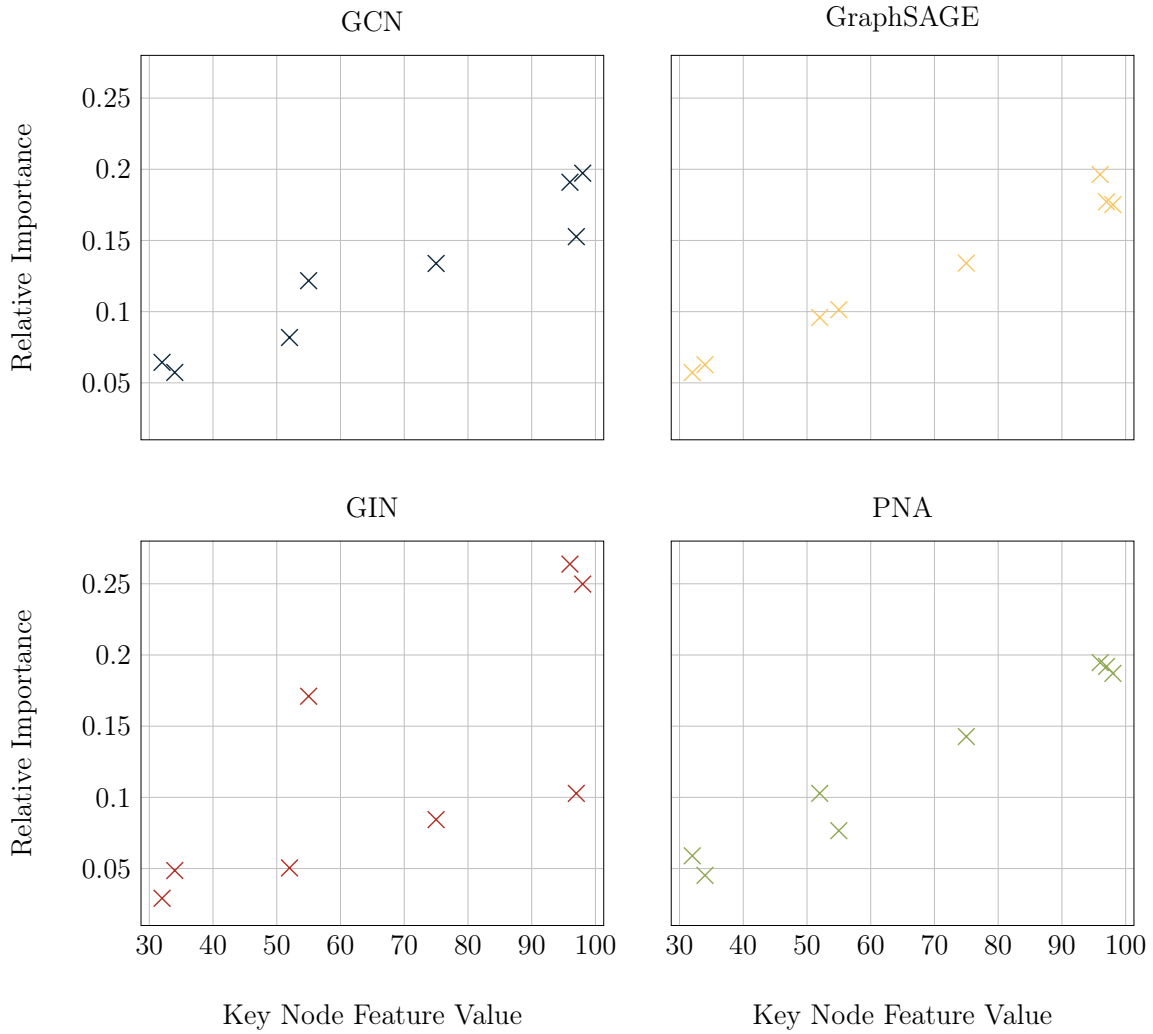


Figure 5.17: Relative importance of the key features in relation to their feature value. We compare the attributions generated with GradientShap on the models GCN, GraphSAGE, GIN, and PNA.

In addition, Appendix A.4 shows the explanation correlation as a function of the node distance from the prediction node. Besides, the explanation accuracy is shown when comparing the most relevant features identified by the explanation methods to the key feature.

In summary, only GradientShap and IG can identify the most critical node features. The GNN-specific approach, GNNExplainer, and Saliency, Lime, and GuidedBackProp perform significantly worse in detecting the importance of continuous large feature inputs.

5.6 Runtime Comparison

We also applied the benchmarks to study the runtime of each explanation method. Not only the reliability of the methods' attribution is important for the usability of explanation methods, but also their feasibility, which includes the runtime. We ran the infection benchmark on an NVIDIA Tesla K80 GPU. Figure 5.18 shows the average time per explanation. Gradient-based approaches often operate comparatively. IG and GradientShap are slightly more time-consuming than Saliency and GuidedBackProp, as the interpolation requires the computation of multiple gradients (50 in our experiments). Lime is the slowest among the generic methods since it requires training a new model for each explanation. The same applies to GNNExplainer, which changes the edge weights and node inputs several times to find an optimum. PGExplainer, on the other hand, is much faster since a model is trained only a single time, and per explanation, one inference on the model is performed.

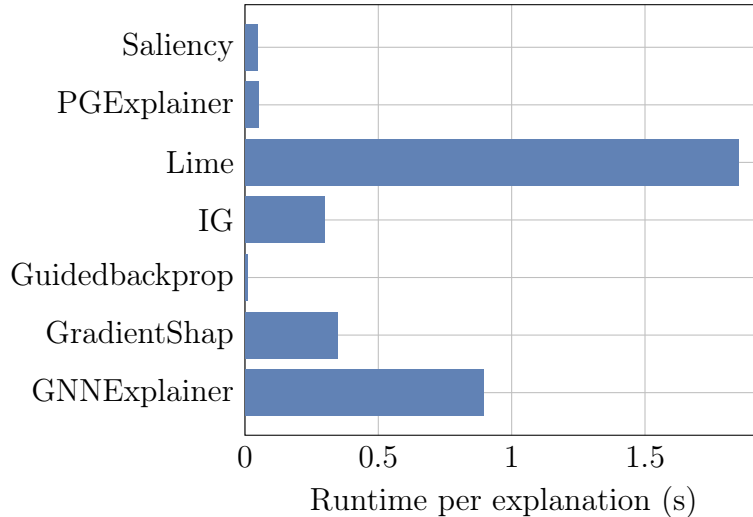


Figure 5.18: Runtime for several explanation methods.

Chapter 6

Conclusion

Given the increasing popularity of GNNs and that, like other ML models, they do not provide insight into their decision-making process, we investigated the reliability of explanation approaches on these models in this thesis. While multiple XAI algorithms have been developed specifically for GNNs recently, we questioned whether generic methods are equally applicable. We designed an interface that allows Captum’s generic methods to be utilized in PyG GNNs without requiring additional implementations. We found that these generic approaches surpass state-of-the-art GNN-specific algorithms in all our tests.

We developed several benchmarks for the experiments in this work, all containing ground truth values on explanations. Across all tasks, we can find high correlations between the explanations generated by XAI methods and the ground truths. By studying the explanations in more detail for edges, entire nodes, and node features, we could discover both advantages and disadvantages of each method. PGExplainer can only provide explanations for edges. We found that this technique is the least accurate of all. Furthermore, we discovered that they regard patterns as significant that are globally relevant but irrelevant in the local context and hence deliver incorrect explanations. In contrast to earlier research, our investigations showed that PGExplainer did not outperform GNNExplainer. However, GNNExplainer performed only slightly better. Indeed, these low performances of GNNExplainer support those of the latest research. We obtain better results with GuidedBackProp and Saliency, which frequently generate similar explanations due to their design parallels. Both approaches have by far the fastest execution time. Moreover, they are proficient in identifying the importance of edges and entire nodes in accordance with the ground truth. However, when explaining node features, we find those two techniques struggle to explain continuous features with large values (we tested between 1 and 100). Lime is by far the slowest approach. Although Lime generated explanations consistent with ground truth, we found that the explanation scores in all experiments were lower with this method than with the other explanation approaches, implying that the explanations are less expressive. Finally, we researched IG and GradientShap, and while comparing both approaches, we also discovered similarities in their explanations. These algorithms have high performance on all benchmarks. To our knowledge, GradientShap has not been tested on GNNs,

whereas IG has been examined in prior research. For IG, our findings support the latest research. IG and GradientShap can overcome the weaknesses of Saliency and GuidedBackProp by interpolating over several inputs instead of using only one, while this slows down the generation of explanations.

All explanation approaches are tested on four different datasets. Additionally, we aimed to verify the capability of explanation methods not only on one GNN but on different architectures. We discovered that the overall performance of the explanation approaches was similar among models as long as the model’s performance was likewise constant. On the other hand, when the model performance varied, we also discovered performance differences in the ground truth comparison for the XAI approaches across the models. This indicates that the generic procedures are appropriate for all types of models studied in this thesis. In terms of GNN accuracy, we notice that PNA outperforms GCN, GIN, and GraphSAGE on a wide range of tasks. Consequently, this model’s explanations best aligned with the ground truth. To conclude, using PNA in combination with IG or GradientShap can be an appropriate starting point for developers and researchers looking for both an accurate prediction model and a reliable explanation method.

While we tested several XAI approaches on GNNs, we advise that further GNN-specific and generic methods be investigated. Our benchmark dataset can be used as a foundation for such objectives. In addition to the earlier comments about the methods, it should be emphasized that part of the explanation strategies studied are based on hyperparameters. For the selection of these parameters in our work, we mostly drew on already-existing standard parameters; however, we encourage additional investigation into the extent to which the choice of these parameters influences the quality of the GNN explanation. Finally, additional criteria such as faithfulness, consistency, and stability can provide additional insight into the trustworthiness of the various explanation methodologies for a thorough overview. While specific approaches have already been tested on these criteria for GNNs, we believe this research should be expanded to include other methods, particularly generic methods that performed well in our experiments, such as GradientShap.

Bibliography

- Agarwal, C., Zitnik, M., and Lakkaraju, H. Probing gnn explainers: A rigorous theoretical and empirical analysis of gnn explanation methods. In *International Conference on Artificial Intelligence and Statistics*, pp. 8969–8996. PMLR, 2022.
- Ancona, M., Ceolini, E., Öztireli, C., and Gross, M. Gradient-based attribution methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pp. 169–191. Springer, 2019.
- Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.
- Baehrens, D., Schroeter, T., Harmeling, S., Kawanabe, M., Hansen, K., and Müller, K.-R. How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11:1803–1831, 2010.
- Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- Bajaj, M., Chu, L., Xue, Z. Y., Pei, J., Wang, L., Lam, P. C.-H., and Zhang, Y. Robust counterfactual explanations on graph neural networks. *Advances in Neural Information Processing Systems*, 34:5644–5655, 2021.
- Baldassarre, F. and Azizpour, H. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686*, 2019.
- Bau, D., Zhu, J.-Y., Strobel, H., Lapedriza, A., Zhou, B., and Torralba, A. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078, 2020.
- Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
- Burrell, J. How the machine ‘thinks’: Understanding opacity in machine learning algorithms. *Big data & society*, 3(1):2053951715622512, 2016.
- Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33: 13260–13271, 2020.

- Das, A. and Rad, P. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371*, 2020.
- Doshi-Velez, F. and Kim, B. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.
- Faber, L., Moghaddam, A. K., and Wattenhofer, R. Contrastive graph neural network explanation. *arXiv preprint arXiv:2010.13663*, 2020.
- Faber, L., K. Moghaddam, A., and Wattenhofer, R. When comparing to ground truth is wrong: On evaluating gnn explanation methods. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining*, KDD '21, pp. 332–341. Association for Computing Machinery, 2021. ISBN 9781450383325. doi: 10.1145/3447548.3467283. URL <https://doi.org/10.1145/3447548.3467283>.
- Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.
- Frasconi, P., Gori, M., and Sperduti, A. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998.
- Funke, T., Khosla, M., and Anand, A. Hard masking for explaining graph neural networks. 2020.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.
- Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT press, 2016.
- Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.
- Holzinger, A., Saranti, A., Molnar, C., Biecek, P., and Samek, W. Explainable ai methods-a brief overview. In *International Workshop on Extending Explainable AI Beyond Deep Models and Classifiers*, pp. 13–38. Springer, 2022.
- Hooker, S., Erhan, D., Kindermans, P.-J., and Kim, B. A benchmark for interpretability methods in deep neural networks. *Advances in neural information processing systems*, 32, 2019.
- Huang, Q., Yamada, M., Tian, Y., Singh, D., and Chang, Y. Graphlime: Local interpretable model explanations for graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2022.
- Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Alsallakh, B., Reynolds, J., Melnikov, A., Kliushkina, N., Araya, C., Yan, S., and Reblitz-Richardson, O. Captum: A unified and generic model interpretability library for pytorch, 2020.

-
- LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature*, 521(7553):436–444, 2015.
- Li, P., Yang, Y., Pagnucco, M., and Song, Y. Explainability in graph neural networks: An experimental survey. *arXiv preprint arXiv:2203.09258*, 2022.
- Lin, W., Lan, H., and Li, B. Generative causal explanations for graph neural networks. In *International Conference on Machine Learning*, pp. 6666–6679. PMLR, 2021.
- Lucic, A., Ter Hoeve, M. A., Tolomei, G., De Rijke, M., and Silvestri, F. Cfgnnexplainer: Counterfactual explanations for graph neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 4499–4511. PMLR, 2022.
- Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.
- Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., and Zhang, X. Parameterized explainer for graph neural network. *Advances in neural information processing systems*, 33:19620–19631, 2020.
- Micheli, A. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.
- Molnar, C. *Interpretable machine learning*. Lulu. com, 2020.
- Montavon, G., Samek, W., and Müller, K.-R. Methods for interpreting and understanding deep neural networks. *Digital signal processing*, 73:1–15, 2018.
- Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.
- Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.
- Nielsen, M. A. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.
- Pope, P. E., Kolouri, S., Rostami, M., Martin, C. E., and Hoffmann, H. Explainability methods for graph convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10772–10781, 2019.
- Rao, J., Zheng, S., and Yang, Y. Quantitative evaluation of explainable graph neural networks for molecular property prediction. *arXiv preprint arXiv:2107.04119*, 2021.
- Ribeiro, M. T., Singh, S., and Guestrin, C. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

- Samek, W. and Müller, K.-R. Towards explainable artificial intelligence. In *Explainable AI: interpreting, explaining and visualizing deep learning*, pp. 5–22. Springer, 2019.
- Samek, W., Binder, A., Montavon, G., Lapuschkin, S., and Müller, K.-R. Evaluating the visualization of what a deep neural network has learned. *IEEE transactions on neural networks and learning systems*, 28(11):2660–2673, 2016.
- Samek, W., Wiegand, T., and Müller, K.-R. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*, 2017.
- Sanchez-Lengeling, B., Wei, J., Lee, B., Reif, E., Wang, P., Qian, W., McCloskey, K., Colwell, L., and Wiltchko, A. Evaluating attribution for graph neural networks. *Advances in neural information processing systems*, 33:5898–5910, 2020.
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.
- Schlichtkrull, M. S., De Cao, N., and Titov, I. Interpreting graph neural networks for nlp with differentiable edge masking. *arXiv preprint arXiv:2010.00577*, 2020.
- Schnake, T., Eberle, O., Lederer, J., Nakajima, S., Schütt, K. T., Müller, K.-R., and Montavon, G. Higher-order explanations of graph neural networks via relevant walks. *arXiv preprint arXiv:2006.03589*, 2020.
- Shrikumar, A., Greenside, P., and Kundaje, A. Learning important features through propagating activation differences. In *International conference on machine learning*, pp. 3145–3153. PMLR, 2017.
- Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
- Sperduti, A. and Starita, A. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.
- Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.
- Sundararajan, M., Taly, A., and Yan, Q. Axiomatic attribution for deep networks. In *International conference on machine learning*, pp. 3319–3328. PMLR, 2017.
- Tang, J. and Liao, R. Graph neural networks for node classification. In Wu, L., Cui, P., Pei, J., and Zhao, L. (eds.), *Graph Neural Networks: Foundations, Frontiers, and Applications*, pp. 41–61. Springer Singapore, Singapore, 2022.
- Thomas, A. W., Heekeren, H. R., Müller, K.-R., and Samek, W. Analyzing neuroimaging data through recurrent deep learning models. *Frontiers in neuroscience*, 13:1321, 2019.
- Vu, M. and Thai, M. T. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. *Advances in neural information processing systems*, 33: 12225–12235, 2020.

-
- Wang, X., Wu, Y., Zhang, A., He, X., and Chua, T.-s. Causal screening to interpret graph neural networks. 2020.
- Wang, X., Wu, Y., Zhang, A., He, X., and Chua, T.-S. Towards multi-grained explainability for graph neural networks. *Advances in Neural Information Processing Systems*, 34:18446–18458, 2021.
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.
- Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.
- Ye, Y. and Ji, S. Sparse graph attention networks. *IEEE Transactions on Knowledge and Data Engineering*, 2021.
- Ying, Z., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.
- Yuan, H., Tang, J., Hu, X., and Ji, S. Xgmn: Towards model-level explanations of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 430–438, 2020.
- Yuan, H., Yu, H., Wang, J., Li, K., and Ji, S. On explainability of graph neural networks via subgraph explorations. In *International Conference on Machine Learning*, pp. 12241–12252. PMLR, 2021.
- Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pp. 818–833. Springer, 2014.

Appendix A

Appendices

A.1 Infection Benchmark

For determining whether the performance is stable regardless of the distance between the edge and the node being classified, we separate the explanation accuracy by distance, cf. Figure A.1. All algorithms except PGExplainer and GNNExplainer identify the edge directly next to the node successfully. While the techniques GradientShap, GuidedBackProp, IG, and Saliency work well for all edges regardless of distance for SumGNN, the same approaches worsen in performance for the other models as the edge distance increases.

A.2 Titanic Benchmark

Table A.1 provides the explanation accuracy when we select the 13 most essential inputs, edges or node features and evaluate whether they correspond to the node features of the node being classified. We do not expect 100% explanation accuracy since not all node features need to be important, but the lower the score, the more will be assigned to the inputs considered as noise. We expect varying explanation accuracies among the GNNs considering the same XAI method. The highest explanation accuracy can be achieved with Saliency on the PNA model with 56.59%. This means that, on average, 56.59% out of the 13 features in the relevant node are among the most important attributes. Also, across all models, Saliency yields the highest explanation accuracy. Moreover, we see that all generic methods have higher scores than GNNExplainer. For the GCN model, the GNNExplainer recognizes, on average less than one of the important node features as such. We also observe quite a large variation within a method but across different models. All methods show lower correlations with GCN and GIN compared to GraphSAGE and PNA, which is consistent with the performance of the models.

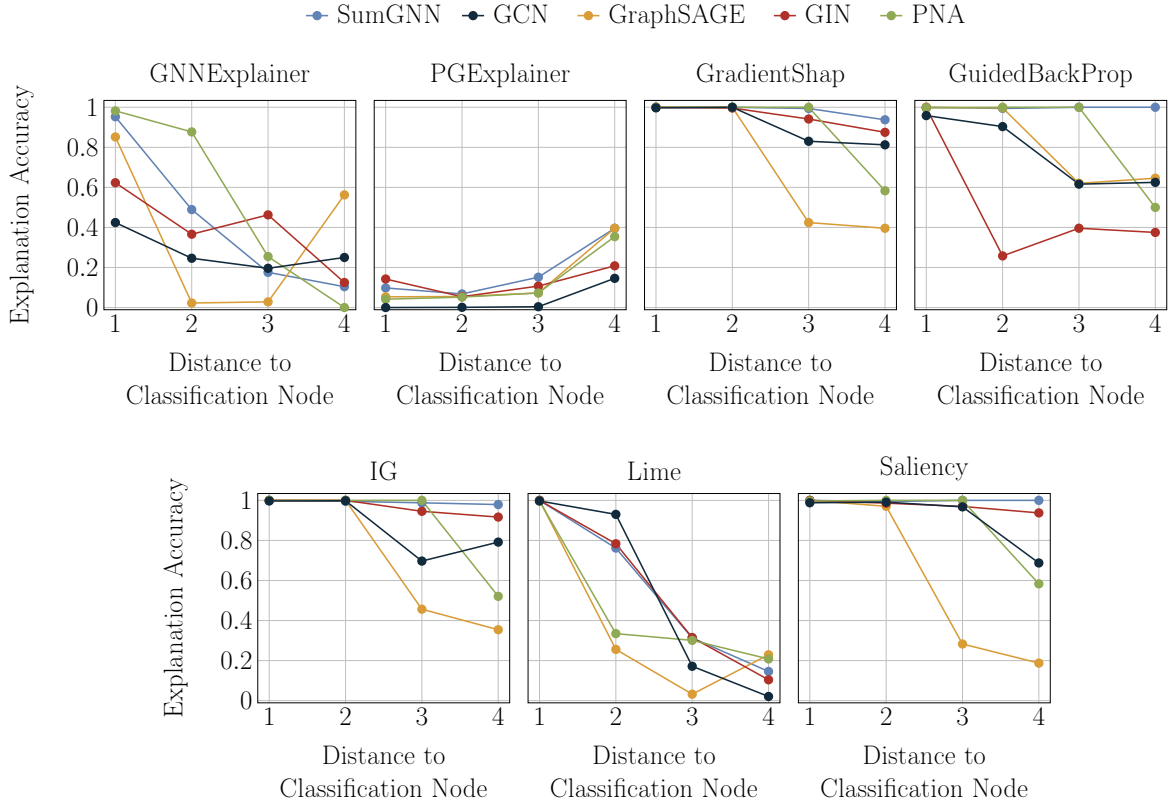


Figure A.1: Distribution of the explanation accuracy depending on the distance of the edge. While SumGNN performs consistently well for most XAI methods across all distances, we see a decrease in performance for other models when edges are more distant.

Table A.1: Explanation accuracy (in %) comparing the most important inputs according to the explainers with those considered important given the dataset. We compare the performance of GNNExplainer, GradientShap, GuidedBackProp, IG, Lime, and Saliency on the models GCN, GIN, GraphSAGE, and PNA. Saliency has the highest explanation accuracy among all XAI methods.

	GCN	GRAPHSAGE	GIN	PNA	MEAN \pm STD
GNNEXPLAINER	4.13	10.04	10.12	15.70	9.01 \pm 4.36
GRADIENTSHAP	26.50	38.78	31.91	37.60	28.90 \pm 12.33
GUIDEDBACKPROP	17.31	35.80	24.65	56.73	30.50 \pm 15.28
IG	24.47	39.54	38.29	39.27	30.55 \pm 12.94
LIME	12.79	35.39	14.74	28.52	20.62 \pm 10.01
SALIENCY	27.25	59.34	33.32	59.95	39.94 \pm 17.93

A.3 Neighborhood Benchmark

Figure A.2 displays the explanation correlation based on the distances of an edge to the prediction node. We find comparable patterns to the Infection dataset findings. For GraphSAGE and PNA, the XAI methods GradientShap, GuidedBackProp, IG, and Saliency can maintain a high correlation value, regardless of the distance separating the edge and the node. All those methods demonstrate a decrease in the explanation correlation for GIN. This indicates that the model is not able to retrieve all the information from the distant edges. Lime has a low correlation value for all methods for those more distant edges. Moreover, a low correlation is found for GNNExplainer, independent of model and distance.

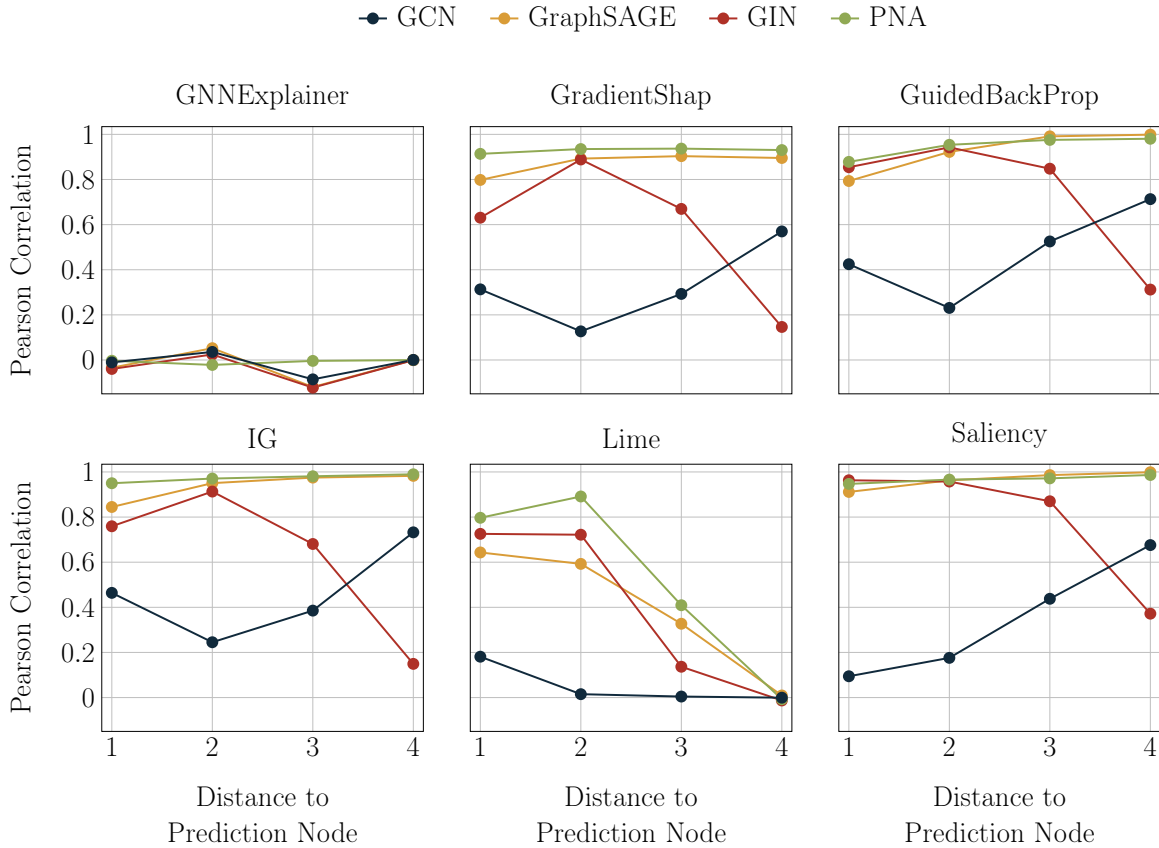


Figure A.2: Distribution of explanation correlation plotted against the distance of the edge to the prediction node.

A.4 Summation Benchmark

Table A.2 provides the explanation accuracy given that we select the most important feature among a node and assess whether it matches the key node feature. The highest explanation accuracy over all GNNs can be achieved with GuidedBackProp. However, Saliency, GradientShap, and IG also achieve values close to 100%. Only GNNExplainer and Lime consider a different node feature as the most important one for the GNNs.

Table A.2: Explanation accuracy (in %) across different graph model architectures and explainability methods for the Summation dataset. The correspondence between the most important feature and the key feature is measured per node and averaged.

	GCN	GIN	GRAPHSAGE	PNA
GNNExplainer	0.10	0.10	0.10	0.10
GradientShap	0.76	0.71	0.91	0.94
GuidedBackProp	1.00	0.97	1.00	1.00
IG	0.76	0.59	0.91	0.94
LIME	0.10	0.11	0.11	0.12
Saliency	1.00	0.89	0.96	0.99

Figure A.3 shows the explanation correlation in relation to the distances of a node to the prediction node. Compared to the graphs for the Neighborhood and Infection dataset, the overall correlation values are significantly lower. For PNA, GIN, and GCN, the XAI methods GradientShap and IG report a high correlation value regardless of the distance of the node to the prediction node. For GraphSAGE, these explanation methods even show an increase in correlation the further away the node is. For all other approaches, the scores are generally low.

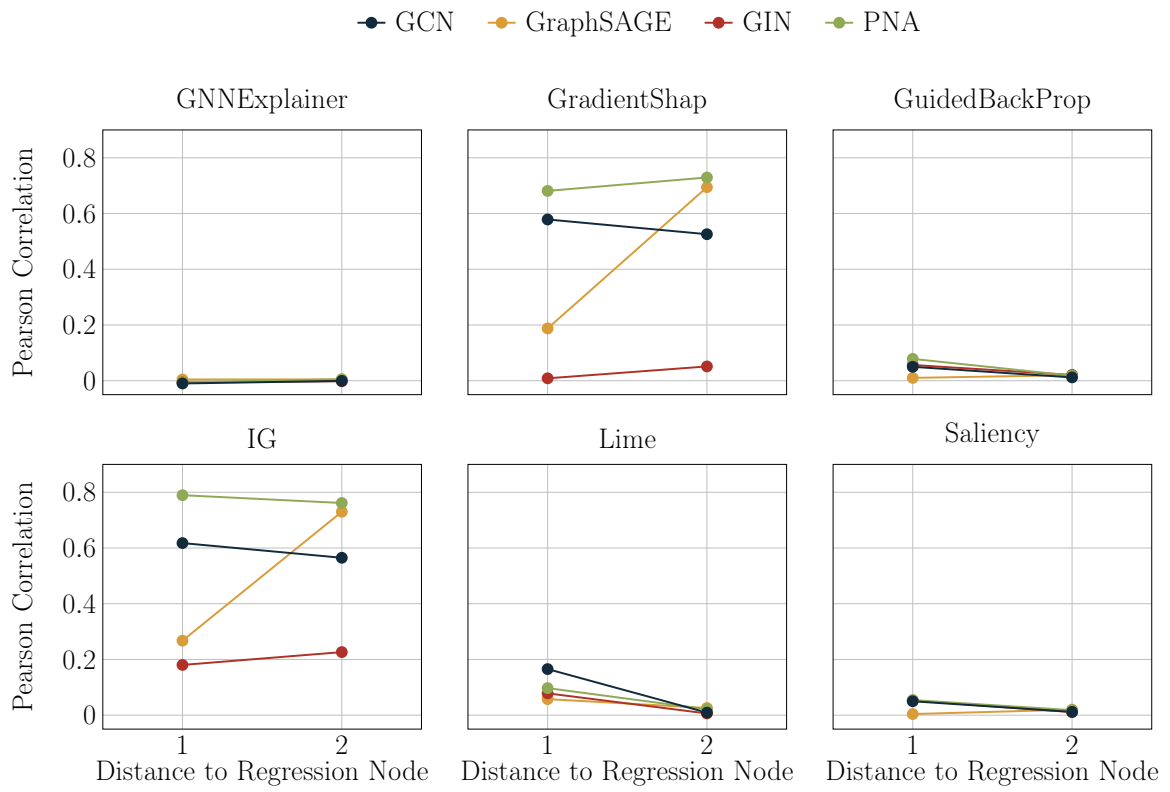


Figure A.3: Distribution of explanation correlation plotted against the distance of the node to the prediction node.