Explainable AI

Mini-Project Report

Rupesh Sapkota, Ashish Prajapati, Sahil Agichani

Junior Data Science Group
Paderborn University
Paderborn, Germany

1 Abstract

The use of machine learning and artificial intelligence has grown rapidly in the past decade and their implementation has extended into various critical domains, such as healthcare, finance, and engineering and they have achieved state-of-the-art performances in various applications. However, these models are often seen as 'black box' systems because their decision-making processes are not transparent. To build trust and ensure ethical considerations, it is important to provide human-understandable explanations for their predictions in real-world applications. In this work, we have built a GNN model based on Relational Graph Convolution and trained it on a node classification dataset and explained the prediction made by the model using a state-of-the-art explainer 'GNN Explainer' and recognized important nodes and features for the prediction of certain nodes, which can be taken as the explanation of the GNN. We also have extended our work to visualize the relevant sub-graph for the classification of the nodes and graph in an attempt to understand the working of Explaibale AI methods.

2 Introduction

With the advancement in deep learning models and computational capabilities, the field of machine learning and artificial intelligence has gained quite the momentum over the years in the past decade. The availability of powerful deep learning models along with the concepts such as transfer learning, the use of machine learning, and artificial intelligence has extended to different domain-specific applications such as healthcare, finance, engineering, and so on. While these models have achieved remarkable performance in the areas such as natural language processing [8], image recognition [6], and graph classification [2], the lack of transparency in their decision-making processes has resulted in them being labelled as 'black box' models. These models need to be accompanied by human-understandable explanations of their predictions to ensure transparency, accountability, and ethical considerations to build trust for their application in real-world scenarios.

Graph neural networks (GNNs) have also gained significant attention in recent years for their ability to model complex graph structures and have achieved state-of-the-art results in tasks such as node classification [3], link prediction [13], and graph classification [14]. Similarly, Different variations of graph neural networks have been proposed recently such as Graph Convolution Networks (GCN) [2], and Graph Attention Networks (GAT) [9], which have proved the effectiveness of GNNs in various use cases. However, the lack of interpretability of GNNs has been a significant concern, especially in critical application areas such as molecular chemistry [7] and social networks [15]. A wide range of graph neural network explainers such as GNNExplainer [11] and SubGraphX [12] have been proposed to address this issue.

In this project, we trained a graph neural network (GCN) on a heterogeneous dataset called "MUTAG" [4]. The GCN was based on the Relational Graph Convolutional Layer (R-GCN) model [5]. We used the HeteroGNNExplainer model, which is provided by the Deep Graph Library (DGL) [10], to explain the predictions of the GCN on graph and node level using feature masks, edge masks as well as subgraphs.

3 Data Analysis

To train the GNN model for the Explanation, we use the state-of-the-art node classification dataset called "MUTAG". The MUTAG dataset is a protein classification dataset that is used in the field of graph classification tasks in machine learning and graph mining research. It is commonly used to evaluate the performance of graph-based classification algorithms. The term "MUTAG" stands for "MUTagenicity AGents," and the dataset was primarily created for the prediction of the mutagenicity of chemical compounds.

The dataset contains 188 chemical compounds, each represented as a graph, where the nodes represent atoms and the edges represent chemical bonds. Each node is labelled with the type of atom, and each edge is labelled with the type of bond. This dataset, known as the Mutag dataset, is specifically focused on mutagenic aromatic and heteroaromatic nitro compounds. Its purpose is to assist in the development of predictive models for mutagenicity assessment, which is a critical step in drug discovery. The graphs in the dataset are categorized as either mutagenic or non-mutagenic, indicating their respective mutagenicity property.

One of the defining characteristics of the Mutag dataset is its graph representation. Each chemical compound is represented as a graph, where the atoms and bonds of the compound are represented as nodes and edges, respectively. This graph-based representation allows researchers to leverage graph mining algorithms and techniques to extract meaningful patterns and features from the compounds.

Another important characteristic of the Mutag dataset is its class imbalance. Out of the 188 compounds, only 37 are mutagenic, while the remaining 151 are non-mutagenic. This class imbalance poses a significant challenge to the devel-

opment of accurate predictive models. Class imbalance is a common problem in real-world datasets, and the Mutag dataset provides an excellent platform to study and address this issue.

The following points provide high level overview of the used datasets:

- The dataset contains a total of 27,163 nodes, but we only considered nodes of type "d".
- There are 148,100 edges in the dataset, including reverse edges. The edges represent the connections between nodes in the graph representation of the compounds.
- The target category in the MUTAG dataset is labelled as "d".
- The dataset consists of two classes for the node classification task: mutagenic and non-mutagenic.
- The MUTAG dataset is split into train and test sets for evaluating the node classification models. The training set consists of 272 instances and the test set comprises 68 instances.

4 GNN Training and Evaluation

Following the implementation of Relational Graph Convolution Layer[5], we developed a NodeClassifier model for training. The NodeClassifier class is a generic GNN model that can be used to classify nodes in a graph. The class takes in a graph and a set of node features and returns the node predictions. The class uses a message-passing approach to aggregate information from neighbouring nodes and update the node embeddings. The class also uses dropout to regularize the model and prevent overfitting. The model implements a RelGraphConvLayer layer, which performs relational graph convolution, which involves aggregating and combining information from neighbouring nodes based on the specific relation types.

The forward method of the NodeClassifier class takes in a graph and a set of node features and returns the node predictions. The method first gets the node embeddings from the self. embeds dictionary. Then, the method iterates through the self. layers list and applies each layer to the node embeddings. The final layer of the model outputs the node predictions. The default hidden layers for GNN is set to 3 but can be changed through the command line arguments while running the model.

The NodeClassifier class is a versatile GNN model that can be used for a variety of tasks, such as node classification, link prediction, and graph classification. The class is easy to use and can be easily customized to fit the specific needs of a particular task. In this work, we have implemented the NodeClassifier model to perform node classification tasks on the Mutag Dataset, which was discussed in the previous section as well.

For the training of the model, we provide the user with the ability to give the parameters required for the training of the model as command line arguments. The parameters include the number of hidden layers in the model, the number

of training loops(epochs), the learning rate of the model and so on. Similarly, we evaluate the training using training and validation accuracy as well as their losses during each training epoch and log them on the terminal.

Similarly, we also explain the GNN using a heterogenous GNN Explainer and log the learned edge and feature masks as well as store them in a file. We also produce individual sub-graph as an explanation for different nodes and store them in the data directory.

5 Explaining GNN

To explain the output of the graph Neural Network, we used a state-of-the-art model-agnostic explainability method called GNNExplainer. It learns soft masks for edges and node features to explain the predictions. The masks are optimized by maximizing the mutual information between the original graph's predictions and the predictions of the graph with the masks applied. The masks can have values between 0 and 1, where 0 indicates unimportance and 1 indicates significance.

In our implementation, We used the heterogeneous variant of GNNExplainer called HeteroGNNExplainer provided by the Deep Graph Library(DGL)[10] to handle the heterogenous dataset 'MUTAG'. HeteroGNNExplainer considers different types of nodes and edges for learning the masks, making it more effective. We produced explanations at both the graph and node levels, and as the entire graph consists of 27163 nodes, which is quite large, we perform the explanations for only 10 random nodes with prediction category d in each run of the explanation module. Most of the explanations were produced considering the 1-hop neighbourhood relations of the node.

For the graph explanation part, we have used the **'explain_graph'** function which takes the graph features as input and provides the node feature masks and edge masks that are important to provide the explanation for the output (or prediction) made by the graph neural network. A feature mask is a dictionary that maps node types to their respective node feature importance masks. The masks are of shape $[D_t]$, where $[D_t]$ is the size of the node feature vector for node type t [10]. The values in the masks are between 0 and 1, where a higher value indicates that the feature is more important. The feature mask for node type **'d'**, as shown below, is a list of values that represent the importance of each feature in the graph for the node category **'d'**.

'd': tensor([0.4874, 0.4961, 0.5037, ... 0.5571, 0.4602, 0.4723])

Similarly, the edge masks provide the dictionary that maps canonical edge types to learned edge importance masks. The masks are of shape ($[E_t]$, where ($[E_t]$ is the number of edges for canonical edge type t in the graph. The values in the masks are within the range (0, 1), where a higher value indicates a more important edge [10]. The edge masks are logged on the screen on every run of explanations.

For the node classification task, we used the **'explain_node'** function which takes the node type, the node to be explained, the input graph and graph fea-

tures as input. The function returns the new node index for the centre node (node to be explained), a sub-graph essential for explaining the node within the k hop neighbourhood, feature masks and edge masks as the explanation of the provided input node to the explainer. Figure 1 shows the explanation for

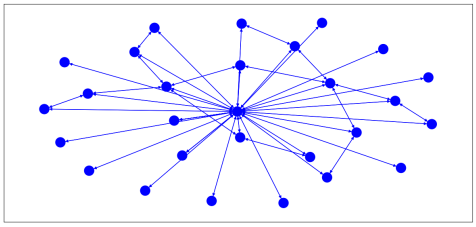

Fig. 1. 1 hop sub-graph explanation for node no 9193

node no 1745. In the sub-graph, the node that is explained is the centre node (node no 5). All the surrounding nodes are relevant nodes within the 1-hop neighbourhood of the centre node that were essential for classifying the centre node. As the explainer model provided a heterogenous graph as an explanation, but the Deep Graph Library (DGL) doesn't provide support for the visualization of the graphs, we had to use another library for the visualization, NetworkX. But during the visualization process, the heterogenous data had to be converted into homogenous data as networkX was also not able to support the heterogeneous graph. A lot of graph information is compromised during the conversion to homogenous and visualization of the sub-graph.

The edge mask provided as the explanation contains the importance of each edge for a given node. The keys in the dictionary are tuples of the form (edge type, relation type, feature name), and the values are tensors of floats. The values in the tensors represent the importance of the edge for the node.

Table 1. Edge importance for feature 'd' of node 1276

Edge and Relation type	Importance
('_Literal', 'rev-micronuc_f', 'd')	tensor([0.4423, 0.5703, 0.4710,..., 0.4504, 0.5467])
('_Literal', 'rev-micronuc_m', 'd')	tensor([0.4347, 0.4830, 0.4561, ..., 0.4563, 0.5687])

The output value of 0.4423 for the edge ('_Literal', 'rev-micronuc_f', 'd') suggests that this relationship between the node and its neighbour, labelled as **'_Literal'** with the attribute **"rev-micronuc_f'**, holds significance for the **'d'** node. This numerical value represents a probability, indicating the likelihood of the edge being important for the feature. As the dataset is heterogenous

and supports multiple edges between the nodes, the importance tensor consists of E_t tensors, where E_t is the number of edges for canonical edge type t in the subgraph[10]. A probability of 1 indicates definite importance, while 0 signifies no importance. A probability of 0.5 suggests an equal chance of it being important or not. Therefore, in this scenario, the probability of 0.5703 for one of the edges suggests that the **'_Literal'** relationship plays a reasonably important role in the **'d'** feature in the node. This implies that the type of the node likely has some influence on the number of bonds in the molecule. The feature importance score for other edges also represents their importance in the classification of the given node.

Similar to the graph classification, the node classification of HeteroGNNExplainer also provides feature masks that represent the importance of each node type for the classification of the particular node. The output for the feature mask was given as a dictionary object that associates the learned node feature importance masks (values) with the respective node types (keys). As in previously mentioned examples, the values for masks range between (0, 1) and are better as they get higher. The example below shows the feature importance of node **'d'** in the classification:
'd': tensor([0.4619, 0.4376, 0.4185, ..., 0.4587, 0.5149, 0.4221, 0.5020, 0.4915, 0.4490])

The feature mask contains 16 values as each node was designed to have 16 features and each value represents the importance score of each feature.

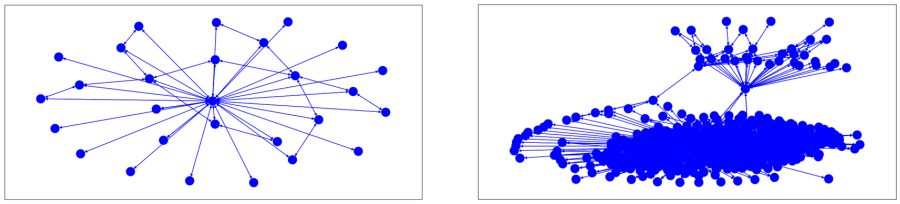

Fig. 2. 1 hop vs 2-hop sub-graph explanation for node no 9193

We also attempted to generate the explanation while considering the 2-hop neighbourhood of the nodes. The Sub-graph explanation for the 2 hops of node 1276 of the graph is shown along with the 1-hop explanation. The analysis showed that considering 2 hops did not have much effect on the node feature importance in our case while the resulting sub-graph was too complex to analyse visually.

Additionally, we attempted to evaluate the quality of the explanation but the DGL library does not provide the evaluation metrics for the evaluation of HeteroGNExplainer. While we tried with pytorch_geometric library for the explanation, the documentation of the library stated that the pytorch_geometric

variant of GNNExplainer doesn't support the heterogeneous graphs and raised the exception **(Heterogeneous graphs not yet supported ...)**. [1].

6 Contribution

The contribution to the project work is almost equal for each of the team members. Rupesh mostly contributed to the GNN Explanation part in implementation as well as in the documentation part along with a small contribution to the GNN implementation. Aashish contributed to the implementation of the GNN module as well as dataset processing along with support in a few of the documentation sections. Similarly, Sahil contributed to various helper and utilities functions as well as maintaining code quality and automation of results and evaluation. Although we named the section for the contribution, each member had equally participated in every aspect of the project work.

7 Conclusion

With the use of machine learning and Artificial Intelligence in real-life scenarios and critical domains, it has become more important to fully understand the working of these models, which have been labelled as 'Black Box Models' due to their lack of transparency. In order to understand the working of these models, in this mini project we explored the field of Explainable AI by training a Graph Neural Network based on Heterogenous Relational Graph Convolution using a heterogenous dataset called 'MUTAG' and explaining the predictions made by the model using the state-of-the-art GNNExplainer. The experiments also showed that the GNN model was able to identify important nodes and features that were relevant to the prediction of the node class as well as relevant sub-graphs that were associated with the prediction of the node class. This work certainly proves that explainable AI can help uncover and understand the working of so-called 'Black Box Models' and build confidence in them for their use in critical applications by making them more transparent.

References

1. PyTorch Geometric. Pytorch geometric documentation, 2023.
2. Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
3. Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 338–348, 2020.
4. Petar Ristoski, Gerben Klaas Dirk De Vries, and Heiko Paulheim. A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15*, pages 186–194. Springer, 2016.

5. Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.
6. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
7. Ruoxi Sun, Hanjun Dai, and Adams Wei Yu. Does gnn pretraining help molecular representation? *Advances in Neural Information Processing Systems*, 35:12096–12109, 2022.
8. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
9. Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.
10. Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.
11. Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32, 2019.
12. Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In *International Conference on Machine Learning*, pages 12241–12252. PMLR, 2021.
13. Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.
14. Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.
15. Yanfu Zhang, Shangqian Gao, Jian Pei, and Heng Huang. Improving social network embedding via new second-order continuous graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2515–2523, 2022.