Explainable Graph Neural Network for Recommendation System

Hugo Bouy[1] **and Rémi Kalbe**[2]

[1],[2],Illinois Institute Of Technology

[1]hbouy@hawk.iit.edu, [2]rkalbe@hawk.iit.edu,

Abstract -

We consider the problem of explaining the predictions of Graph Neural Networks (GNNs) in the specific field of recommendation systems. We explore the different aspects and challenges of a recommender system and conduct a short survey on popular models among GNNs for this specific task. We present our own graph data structure, engineered to unify various forms of graph data representation and streamline interactions within GNN frameworks. Building on this structure, we introduce a tailored explanation method based on ablation studies. Through our proposed LocalGNNAnalyzer, we systematically ablate graph components, measuring the impact on the network's predictions to identify influential factors. Then, we experiment with the popular SubgrpaphX framework on an hypergraph model for social recommendations.

1 Background of the project

1.1 Recommender system (RecSys)

In this section, we propose an in-depth exploration of the context surrounding our project.

Recommender systems have been more and more used over the past few years. Whether to keep the user engaged on a social media platform by suggesting him new publications he might be interested in, or to push the customer to buy more products on e-commerce websites, recommender systems have proven to be extremely effective.

The recommendation task can defined as inferring a user's preferences from previous user-item interactions or other static features, to recommend new relevant items. Formally, the goal is to estimate a preference for an item $i \in I$ by learning the user h^*_u and item representation h^*_i [1]:

$$y_{u,i} = f(h^*_u, h^*_i)$$

With $f(*)$ being a score function (i.e. dot product, cosine, multiple-layer perceptron, etc.) and $y_{u,i}$ representing the preference score for a user u on item i.

We considerate various scenario for recommender systems: [2]

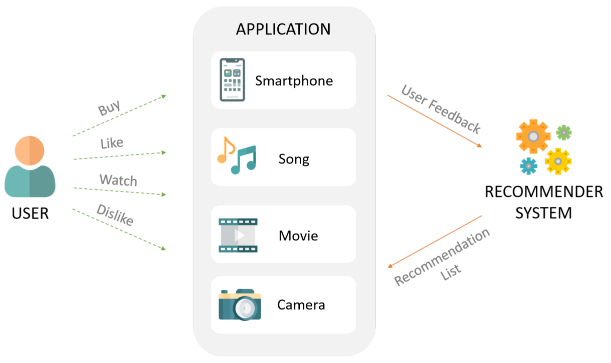

Figure 1. Illustration of a Recommendation System. Source: https://www.mdpi.com/2076-3417/10/16/5510

- **Social recommendation**: Leverages users social relationships to make new predictions. This scenario is based on the social influence theory stating that users with social relationships tends to share common interests.
- **Sequential recommendation**: Includes the temporal order of users interactions with items to produce a recommendation.
- **Session-based recommendation**: Bases the prediction on previous sessions interactions usually over a short period of time. This scenario is especially used to make predictions for anonymous users.
- **Cross-domain recommendation**: Exploits the knowledge from source domain to make a recommendation in another target domain.
- **Multi-behavior recommendation**: Uses various types of user/item interactions such as clicks or favorite to make predictions.
- **Bundle recommendation**: Aims to generate predictions in bundle which is a famous marketing strategy on e-commerce websites.

Our project focuses on **social recommendation** as it one of the most common scenario.

As the research on recommender system kept on growing, new machine learning methods have been developed. Historically, we can summarize the major milestones of recommendation methods as the following: [3]

- **Item-based neighborhood methods**: Directly recommend similar items to the ones the user previously interacted with
- **Representation Learning based methods**: Encode both users and items as vectors (i.e. embedding) in a shared space to allow mathematical comparisons (e.g. matrix factorization models)
- **Deep learning models**: Dominant methodology today, capture the non-linear and non-trivial user-item relationships.

Among the deep learning algorithms, Graph learning-based methods, especially Graph Neural Networks, have shown great potential in the research, and are prevalent methods used in recommender systems.

1.2 Graph Neural Networks (GNN)

GNN are a class of Artificial Neural Networks built for processing data represented as graph. Their relevance for recommender systems lies in the fact that user/items interactions, items relations and user social relationships can be naturally represented with graphs.

GNNs usually use **pairwise message** passing to leverage the graph structure of the data and apply transformations on the data, while preserving the graph symmetries. As an example, a basic GNN would use a separate multi-layer perceptron (MLP) to process each component of the graph and return a learned node-vector, per-edge embedding and a single embedding for the entire graph.

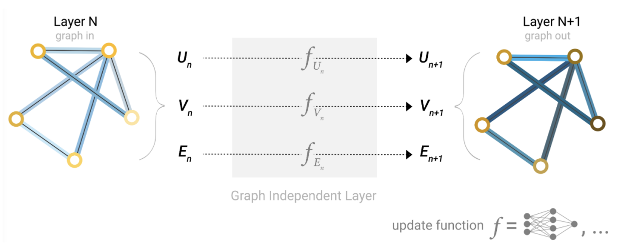

Figure 2. A single layer of a simple GNN. A graph is the input, and each component (V,E,U) gets updated by a MLP to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model. Source: https://distill.pub/2021/gnn-intro/

Of course, more complex GNNs stack such layers together.

In recommendation tasks, nodes usually represent users and items while edges denote the interactions or relationships between them. Famous GNN framework widely adopted for RecSys are:

- **Graph Convolutional Network (GCN)**: Performs convolutional operations on the graph by approximating the first-order eigendecomposition of the graph Laplacian, to iteratively aggregate information from neighbors and update their embedding.
- **Graph Sample and Aggregation (GraphSAGE)**: Samples a fixed size of neighborhood for each node to aggregate information (usually with mean, sum or max pooling) and create a node representation that captures the various scales of the graph structure.
- **Graph Attention Network (GAT)**: Leverages attention mechanisms to map the different importance weights of neighbors nodes.
- **Graph Gated Neural Network (GGNN)**: Is a type of recurrent GNN which adopt a gated recurrent unit (GRU) to update node embedding iteratively. GGNN are very effective but might face scalability issues when applied to large graphs.
- **Hypegraph Graph Neural Network (HGNN)**: Encodes high-order data correlation in a hypergraph structure and then apply convolutional transformations.

Of course, concepts for these generics models can be mixed together to create more specific models.

1.3 Datasets for recommendation tasks

Depending on the recommendation scenario, it exists numerous datasets to train a RecSys. Among the most famous ones, we can note:

- **MovieLens** datasets: familly of datasets assembled by GroupLens Research from the Movielens website consisting in movie ratings. The datasets were collected from various periods of time and are available in different sizes. As an example the MovieLens-1M contains 1 million movie ratings from 6000 users on 4000 movies (Released in 02/2023)
- **LastFM**: a dataset containing 1200 songs with additional information about the artist, album, tags, countries of listeners, year of publication, number of listeners, etc. collected from the LastFM website. This dataset can be used for various recommendation scenarios even for social recommendation as it includes relationships between users.

- **Epinions**: well-suited dataset for social recommendation built from the 'who-trust-whome' online social network Epinions.com (closed in 2018). Members were able to "trust" one another to decide which reviews of a product to be shown. It contains 75,879 nodes and 50,8837 edges.

Other worth mentioned datasets (not detailed in our paper) include the Yelp, Flickr and Amazon datasets which are also often used for training recommendation systems.

1.4 Evaluation metrics for RecSys

It exists several ways to evaluate a recommendation system. For our scenario of Social Recommendation, the most use-full and widely adopted methods are:

- **Hit@K (Hit rate)**: Measures for the top-K recommendations of a user the actual proportion he interacted with (items the user has rated, liked or watched). If there is a match, the hit rate is increased by 1 and then normalized over a complete training set.
- **Precision@K and Recall@K**: Precision measures the fraction of items the user would click among the recommended K items. Recall measure the proportion of the number of user interactions with the recommended K items. Due to their definition, precision and recall are usually low for a recommendation system compare to the precision and recall of a classification model for example.
- **AUC**: Probability that the model make a recommendation on an item the user actually interacted with compare to a non-interacted item.

2 Problem definition

As we discussed previously, GNNs have demonstrated excellent performances on recommendation tasks, and some GNN models are considered as state-of-the-art by the scientific community. However, one area that remains relatively unexplored is how to explain the predictions made by theses GNN which are mostly considered as black boxes. Indeed, more and more users, and even institutions such as the European Union, tend to ask for more transparency on how the users data are processed and used.

The objective of explanation for deep learning models is to investigate the intrinsic connections that underlie the predictions generated. We distinguish 2 types/categories of explanation:

- **Instance-level explanation**: Explanations that are specific to each input graph, providing input-dependent explanations.
- **Model-level explanation**: Offers broad insights and a high-level comprehension to understand deep learning models.

Our project focuses on the instance-level problem of explanation for GNN, in the context of the social recommendation task.

The scientific literature already provides several methodologies for explainable GNNs. These methods can be classified in the 2 above-mentioned categories. For instance level methods, the 3 main algorithms are:

- **GNNExplainer**: Identifies the subgraph that exerts the most substantial influence on the prediction. However, GNNExplainer requires retraining in each prediction scenario making it not easily scalable.
- **PCExplainer**: Employs the extraction of pertinent subgraph for a given prediction with the addition of indication feature dependencies through conditional probabilities.
- **SubgraphX**: Considered as one of the most effective techniques, it identifies the important subgraphs explaining the prediction by performing a MonteCarlo tree search (detailed in a dedicated section).

Model-level methods include:

- **XGNN**: Trains a graph generator to generate graphs that maximize a specific target graph prediction
- **GNNInterpreter**: Numerical optimization technique to acquire the explanation graph through continuous relaxation

Our project introduces one additional instance level explanation method based on an intuition explained in the next section.

3 Algorithms and Results

3.1 Our Explanation Algorithm

As previously introduce, Graph Neural Networks (GNNs), while powerful in leveraging relational structures of data, often act as black boxes, leaving the rationale behind their predictions opaque. To demystify these intricate decision-making processes, we propose a novel explanation algorithm tailored for GNNs in the domain of recommendation systems.

Our algorithm is grounded on the principle of ablation studies, a technique traditionally employed in neuroscience [4] and more recently adapted for neural networks. The crux of our approach lies in systematically altering the input graph by ablating, or removing, certain nodes or edges and observing the variations in the GNN's output.

Through these perturbations, we aim to highlight the elements within the graph that wield substantial influence on the model's predictions.

3.1.1 Intuition

The intuition behind our method stems from a simple yet profound understanding that the relational ties encapsulated in the graph structure hold the key to the recommendations generated. By severing these ties iteratively, we can pinpoint the most influential substructures contributing to the prediction.

Fundamentally, our method assumes that the removal of non-critical elements will have a nominal effect on the overall outcome, while the omission of crucial components will result in significant deviations. This hypothesis allows us to rank the elements based on their impact and provide a localized explanation, centered on the most potent subgraph related to the predicted recommendation.

3.1.2 Algorithm Design

Our explanation algorithm is articulated into several methodical steps, each integral to the accurate dissection of the network's inner workings. Initially, the procedure commences with the selection of a target node — often representing an item or a user in a recommendation context — for which the prediction explanation is sought.

Following this, the algorithm enters an iterative phase where edges connected to the target node undergo systematic removal. This ablation process is meticulously recorded, logging the differential in prediction confidence or scores. The impact of each ablated edge is thus calculated using an *ImpactCalculator* module, elucidated in Section 3.1.7, which leverages the magnitude of change in model output as a metric of significance.

The iterative removals proceed until a predefined condition is met, such as the exhaustion of edges or the attainment of a specified ablation depth. Each step's results are collated into a leaderboard that ranks the edges according to their computed impact scores.

Lastly, to construct a coherent narrative around the model's decision, the ranked list of influential edges is displayed for visualization. This final representation effectively conveys the contributory pathways within the graph that the GNN deems most pertinent to its recommendation decision.

3.1.3 Implementation Insights

Implementational challenges, such as the preservation of graph consistency post-ablation and the computational optimization of the impact calculation, were addressed in our methodology. By creating our own efficient *Graph* manipulation library detailed in Subsection 3.1.5 and designing the *LocalGNNAnalyzer* — detailed in Subsection 3.1.6 — we ensured our approach could scale to larger, more complex relational datasets typical of real-world scenarios.

Our algorithm's modular design enables compatibility across various GNN architectures. It is also constructed with the versatility to handle different types of prediction tasks, from node classification to link prediction within the recommendation systems framework.

3.1.4 Result Interpretation

The resulting analysis provides end-users with granular insight into the GNN's decision-making process. By delineating the most influential factors, our method goes beyond mere prediction to offer a window into the causal relationships within the data that drive the recommendation engine.

Our approach facilitates trust in GNN-powered recommendation systems by illuminating the rationale behind predictions. Moreover, it lays the foundation for further exploration and refinement of explanation methods in graph-based machine learning applications.

The pseudocode summarizing our algorithmic procedure is presented below:

Algorithm 1 High Level Overview of our Algorithm

```
1: Initialize the GNN model and the LocalGNNAnalyzer
2: Define PREDICTFN as the prediction function of the GNN
3: for each starting node do
4:     Prepare ablation plan with LocalGNNAnalyzer
5:     while LocalGNNAnalyzer has next step do
6:         Predict using GNN with PREDICTFN
7:         Execute ablation step in LocalGNNAnalyzer
8:     Collect interpretation of GNN predictions
```

3.1.5 Graph Data Structure Abstraction

In the topic of graph data, diversity in representation often poses a challenge to uniform processing and analysis. To tackle this, we have crafted a highly flexible Graph data structure within our framework, enabling us the freedom to interact with various graph representations seamlessly.

Design Philosophy Our design philosophy centers on abstracting the graph's complexities while also ensuring versatility in dataset interaction. This abstraction primarily takes form as the Graph class, which encapsulates the graph in terms of nodes and edges, represented through efficient Polars DataFrames. Nodes and edges can be manipulated through intuitive methods, allowing a dataset-agnostic algorithm development.

Import and Export Functionality Importing datasets is streamlined via our Graph class, which supports common formats such as CSV, JSON, and Parquet, as well as Python's native List and NumPy array structures. This allows for smooth and flexible dataset integrations, regardless of their inherent format. The real innovation, however, lies in our export functionality. After potential transformations and analysis, our class is capable of reconstructing and exporting the graph back to its original structure (keeping our potential ablations), without the need to retain a copy. This reconstruction hinges on the dynamic metadata tracking we employ, which systematically captures the necessary information for reversal.

Innovative Data Interaction We have engineered solutions for common challenges faced in graph analysis tasks. By addressing the need for efficient node lookup, edge alteration, and neighborhood computation, we remove barriers that otherwise hinder streamlined interaction with graph data. Our class facilitates node and edge insertion and removal, maintaining relational integrity without the added overhead of manual tracking.

Furthermore, the class allows direct query of node neighbors, encapsulating the logic and complexity of relational connections behind simple, clean method calls.

3.1.6 LocalGNNAnalyzer for Explanation Process

To effectively illustrate the rationale behind GNN predictions, we introduce our LocalGNNAnalyzer, it is a system designed specifically to dissect the predictive decisions of GNNs and shine a light on the influential components of the input graph that sway these decisions.

Mechanism At the heart of this system is a carefully orchestrated ablation study—the incremental process of removing graph components (such as nodes or edges) and observing the fluctuations in prediction output. This systematic removal reveals the degree of importance each component possesses in the context of the model's predictions. The LocalGNNAnalyzer records and evaluates the resulting impact scores through our bespoke LocalImpactCalculator, allowing us to trace and rank the contributions of different graph elements.

Ablation Planning The initiation of our system begins with a carefully crafted ablation plan, mapping out the sequence of components to be analyzed. With the starting node as the focal point, the Analyzer generates a sequence of edges slated for ablation. The resulting plan is a thoughtful pathway designed to unravel the most impactful entities influencing the recommendation.

Execution and Interpretation Implementing the ablation involves methodically muting the identified edges and reinstating them with calculated considerations of their attributes. The Analyzer keeps a dynamic record of these changes in a leaderboard, which acts as a live scoreboard updating the ranks of the edges based on their level of impact. This leaderboard is then used to deduce the starting node for the next iteration of analysis, ensuring multifaceted examination.

The culmination of the process is an interpretation phase where the method distills the complex ablation outcomes into a digestible breakdown. The end result is a ranked list of influential paths, pieced together to form a narrative that explains the recommendation decision.

Modularity of the LocalGNNAnalyzer The design of our LocalGNNAnalyzer stands out for its modularity and ease of integration into the workflow of GNN-based systems. With its built-in feedback loop, the Analyzer can be effortlessly incorporated, much like an environment in reinforcement learning. It operates in a sequential manner: initiating the analysis with an ablation plan, making predictions, executing ablation steps, and finally obtaining an interpretation—all structured in a loop conducive to continuous evaluation.

The loop begins by preparing an ablation plan—selecting which edges to remove based on their potential influence on the model's output. Following this setup, our system enters a feedback phase where it performs predictions with the GNN model for the current graph state. Afterward, it executes the ablation step, carefully removing edges as per the plan, then reconstructs the graph including previously ablated edges to ensure the graph's integrity is maintained for subsequent iterations. This cyclical nature of prediction, ablation, and reconstitution of the graph allows us to delve deeper into the network, one layer at a time, peeling back the layers of complexity to reveal the most influential substructures.

What renders this process particularly powerful is the Analyzer's capability to handle the graph datastructure, irrespective of the analysis stage, and transform it back into a tensor format that can be directly utilized for subsequent GNN predictions. Such a feature streamlines the process of iterative analysis, mirroring a continuous feedback loop that offers regular insights into the evolving influence map of the graph components. It reflects an environment that dynamically responds to the changes within the system, adaptive enough to facilitate ablation studies at varying depths, unravelling key findings at each level. The feedback loop can be observed in the provided algorithm 2.

Algorithm 2 LocalGNNAnalyzer Feedback Loop

```
Require: $model$, the trained GNN
Require: $data$, the graph data
Require: $node\_id$, the node to start ablation
Require: $depth$, the maximum depth of analysis

$alz \leftarrow$ Initialize $analyzer$
$alz.prepare\_ablation\_plan(node\_id, depth)$

while $alz.has\_next\_step()$ do
    $prediction \leftarrow$ Predict($model, data, node\_id$)
    Recreate $data$ from $alz.graph$
    $data \leftarrow alz.execute\_ablation\_step(prediction)$
end while

$interpretation \leftarrow$ Get interpretation from $alz$
return $interpretation$
```

3.1.7 Local Impact Calculator and Ablation Analysis

Integral to the LocalGNNAnalyzer is the Local Impact Calculator, a component that quantifies the effect of graph modifications on the predictive output of GNNs. It utilizes a variety of calculative methods to measure impact, each suitable for different analysis purposes and prediction types, such as the *Absolute Difference*, *Probability Change*, and the *Class Difference Matrix* methods.

Impact Calculation Methodology The *Absolute Difference* method captures the simplest form of impact by computing the absolute difference in prediction scores before and after ablation. On the other hand, the *Probability Change* method is well-suited for classification tasks, where it assesses the deviation in the probability distribution of class predictions as a result of graph alterations.

In scenarios demanding nuanced differentiation between classes, the *Class Difference Matrix* method excels, leveraging a predefined matrix that articulates the impact significance when the predicted class changes. This matrix enables fine-grained analysis of class transitions, particularly relevant in recommendation systems where the transition between item preferences must be profoundly understood.

Leaderboard Mechanism An innovative leaderboard mechanism lies at the core of the analyzer's functionality, ranking the impact of ablated components and serving as the decision matrix for subsequent iterations. When the analyzer exhausts the current plan, it employs the leaderboard to select the next node for deeper ablation analysis. This selection process is crucial, ensuring a focused and informed exploration of the graph's influential regions.

The operation of selecting the next starting node is contingent on the impact scores, whereby an edge previously ablated with the highest impact score intimates a new area of the graph warranting further exploration. This strategy facilitates a depth-first analysis, progressively revealing the graph's complexity layer by layer and rendering the explanatory process iterative and comprehensive.

3.1.8 Experimentation Overview with LocalGNNAnalyzer

Our exploratory work involved applying the LocalGNNAnalyzer to perform ablation studies on a GNN with a simple dataset (cora [5]) utilizing our Graph class.

In our experiment, the analyzer orchestrated a feedback loop of ablation and prediction. Each cycle entailed removing edges, predicting with the altered graph, and then reconstructing it for the next iteration. Through this process, we aimed to identify critical substructures that significantly impact model output.

While the experiment demonstrated the potential effectiveness of our approach, it is not yet complete. The algorithm successfully executed some steps of ablation and subsequent prediction, however some issues prevents us from going deeper in the graph, and the final stage of generating interpretable results is still in development.

Code and interim results are openly shared on our GitHub repository: `https://github.com/RemiKalbe/IIT-ML-PROJECT`.

3.2 Comparison with Subgraph-X

While most methods (including ours) focus on explaining GNNs by identifying important nodes, edges or nodes features, Subgraph-X is able to provide input-dependent subgraph-level explanations. [6] This property of Subgraph-X makes it especially relevant for the recommendation task as the resulting explanation would be able to identify which item/users relations lead to the prediction.

The Subgraph-X task can be formally written as following: Considering the set of connected subgraphs G denoted as $\{G_1, ...G_i, ..., G_n\}$ with n being the number of different connected subgraphs in G, the explanation of prediction y for input graph G can be defined as:

$$G^* = \arg_{|G_i \leqslant N_{min}|} \max(\text{Score}(f(.), G, G_i))$$

Where Score(.,.,.) is a scoring function evaluating the importance of a subgraph given the trained GNN and the input graph.

To be scalable to large-scale and complex graphs, Subgraph-X doesn't use a brute-force method, but performs a Monte Carlo Tree Search. The details of this algorithm can be found in its original publication paper. [6]

As a first experiment with Subgraph-X, we performed some explanation on the BA-Shapes dataset which has the major advantage of being very visual. This dataset was first proposed by Ying et. al. in their paper [7] and is a synthetic dataset for node classification constructed from Barabási–Albert (BA) graphs. To do so, we used the Dive-Into-Graphs python library which includes an implementation of the Subgraph-X algorithm, as well as Pytorch and Torch Geometric libraries. For this basic example, we used a pre-trained GCN model.

A example of explanation provided by Subgraph-X is the following:

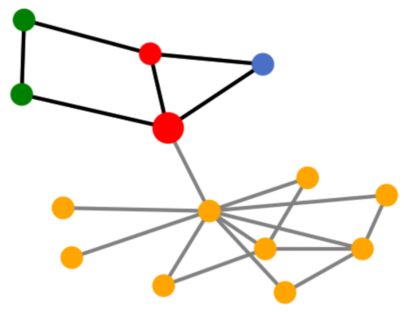

Figure 3. Explanation provided by Subgraph-X on one graph from the BA-Dataset. The different colors correspond to class labels. The prediction is the larger circle, while the subgraph responsible for the prediction is represented in bold.

After understanding the behavior of Subgraph-X, we conducted a second experiment on a social recommendation scenario. To this end, we used the LastFM dataset, processed to only consider the user paired network relationships and the artists they listen to.

To perform the predictions, we used a HGNN model named Multi-Channel Hypergraph Convolutional Network (MHCN) introduced by Yu and Yin et. al. in 2022 [8]. This model leverages both a Hypergraph representation and convolutions to produce recommendations. It applies convolutions over 3 channels on the user embedding with specific Triangle motifs, allowing to reveal the underlying structure of social relations. These motifs can be divided in 3 groups:

- **Social Motifs**: Summarize all possible triangular explicit relation between users over the social network (i.e. users connected to each other = 'friends')

- **Joint Motifs**: Represent the network relation of 'friends purchasing the same item' (in our project, 'friends listening to the same artists') based on the social influence theory previously mentioned.

- **Purchase Motif**: This motif is used to highlight users who have no explicit social connections but can still be mapped to a higher social order (i.e. users not directly friends but sharing similar interests).

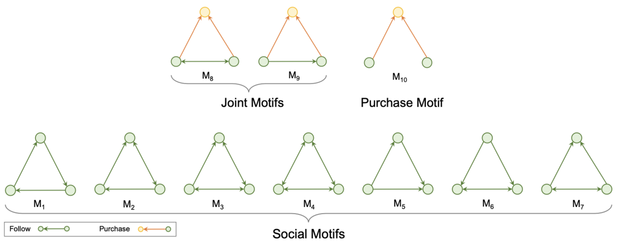

Figure 4. Triangle motifs used in the MHCN model. The green circles denote users and the yellow circles denote items. Source: Yu and Yin, et. al. [8]

The item embedding features on their side do not go through the convolutional layers mentioned above but through a simple graph convolution along with the user-embedding and the resulting vector of attention applied after the 3 triangular convolutions, as we can observe in the model overview.

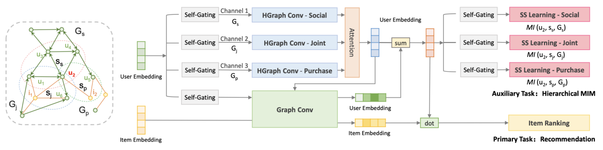

Figure 5. Overview of the MHCN model (1-layer) Each triangle in the left graph is a hyperedge and also an instance of defined motifs. G_s, G_j and G_p denote the three motif-induced hypergraphs constructed based on social, joint, and purchase motifs, respectively. s_s s_j, and s_p in the three dotted ellipses denote three ego-networks with u_2 as the center, which are subgraphs of G_s, G_j and G_p,respectively. Source: Source: Yu and Yin, et. al. [8]

We trained the model with the following recommended hyper-parameters: batch_size = 4096, embedding_size = 64, n_layers = 2, ssl_reg: 1e-05 and reg_weight = 1e-05.

With a training on 100 epochs, we reached the following results on the validation and test set:

	Val set	Test set
Hit@10	**0.608**	**0.636**
Precision@10	0.0955	0.102
Recall@10	0.1936	0.208

Table 1. Hit@10, precision@10 and recall@10 on the LastFM dataset with the MHCN model

With our model trained and able to make relevant prediction, we focused on explaining the predictions using the

SubgraphX method previously introduced. Unfortunately, and despite the numerous attempts, we did not manage to do so in the time constraint of this project. This unfinished aspect of our project may be a good starting point for a future research project.

4 Conclusion

Our project embarked on the quest to demystify the predictive processes of Graph Neural Networks within the domain of recommendation systems. We introduced a Graph data structure that was integral to our algorithm and facilitated local ablation analysis via the LocalGNNAnalyzer to shed light on GNN decision-making. We found that not only can straightforward approaches yield valuable insights (as demonstrated by our method), but also that intricate algorithms like SubgraphX can unravel deeper data relationships, providing explanations that resonate more naturally with human reasoning.

While our exploration is yet a work in progress, it has proven to be a profound learning journey in the machine learning landscape of GNNs. The initiative has set the stage for subsequent research endeavors, allowing for the enhancement of our existing methods and the investigation of intricate tools like SubgraphX.

5 Individual contributions

Rémi Kalbe

- Development of 'homemade' graph explanation algorithm
- Report writing (part concerned)

Hugo Bouy

- Survey on recommendation systems and GNN
- Experimentation with SubgraphX
- Report writing (part concerned)

Link to our github: https://github.com/RemiKalbe/IIT-ML-PROJECT

References

[1] Wu Shiwen, Sun Fei, Zhang Wentao, Xie Xu, and Cui Bin. Graph neural networks in recommender systems: A survey. Arxiv: `https://arxiv.org/abs/2011.02260`, 2020. Accepted by ACM Computing Surveys (CSUR).

[2] Gao Chen, Zheng Yu, Li Nian, Li Yinfeng, Qin Yingrong, Piao Jinghua, Quan Yuhan, Chang Jianxin, Jin Depeng, He Xiangnan, and Li Yong. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. Arxiv: `https://arxiv.org/abs/2109.12843`, 2021. Accepted by ACM Transactions on Recommender Systems.

[3] Mohammadi Amir, Reza. Explainable graph neural network recommenders; challenges and opportunities. ACM: `https://dl.acm.org/doi/10.1145/3604915.3608875`, 2023.

[4] Meyes Richard, Lu Melanie, Waubert de Puiseau Constantin, and Tobias Meisen. Ablation studies in artificial neural networks. Arxiv: `https://arxiv.org/abs/1901.08644`, 2019.

[5] Sen Prithviraj, Namata Galileo, Mark, Bilgic Mustafa, Getoor Lise, Gallagher Brian, and Eliassi-Rad Tina. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[6] Yuan Hao, Yu Haiyang, Wang Jie, Li Kang, and Ji Shuiwang. On explainability of graph neural networks via subgraph explorations. Arxiv: `https://arxiv.org/abs/2102.05152`, 2021. Accepted by ICML 2021.

[7] Ying Rex, Bourgeois Dylan, You Jiaxuan, Zitnik Marinka, and Leskovec Jure. Gnnexplainer: Generating explanations for graph neural networks. Arxiv: `https://arxiv.org/abs/1903.03894`, 2019.

[8] Yu Junliang, Yin Hongzhi, Li Jundong, Wang Qinyong, Quoc Nguyen, Hung Viet, and Zhang Xiangliang. Self-supervised multi-channel hypergraph convolutional network for social recommendation. Arxiv: `https://arxiv.org/abs/2101.06448`, 2021. Accepted by WWW'21.