Final Project ECSE 556 : BioNetRec, a GNN based and LLM refined recommendation system.

Hamza Chikhaoui
Department of Computer Science
McGill University
Montreal, QC, Canada
hamza.chikhaoui@mail.mcgill.ca

Abstract

Learning representations on heterogeneous graphs is an increasingly active area of research, driven by the need for expressive, low-dimensional embeddings that can effectively support various downstream tasks such as classification, link prediction, and recommendation. In this work, we propose a graph neural network (GNN)-based recommendation algorithm designed to leverage geometric learning techniques for edge prediction. Our approach utilizes the GraphSAGE convolution operator to build an inductive model capable of learning meaningful node representations from a subset of the IBKH graph. By focusing on inductive learning, our model is able to generalize to unseen data during inference, a critical advantage for dynamic and evolving graphs. We benchmark the performance of our method against traditionally transductive approaches, such as the Graph Convolutional Network (GCN), and show that with careful hyperparameter tuning, the GraphSAGE operator achieves competing performance to GCN while retaining its inductive nature. In addition to this, we fine tune a Large Language Model (gpt-4-turbo) to generate final natural language predictions aimed at making our solution more user friendly in the perspective of a possible deployment. These results highlight the potential of GraphSAGE as a powerful tool for edge prediction on heterogeneous graphs, and the potential deployment capabilities of BioNetRec.

1 Introduction

1.1 C1

One of the most common issues patients face in Canada is the lack of access to a family doctor [1]. One social reality that illustrates this fact has been the increase in internet research about health questions [2]. However, the volume and complexity of health information can easily exceed an individual's information processing capacity and lead to information overload [2], resulting in incomplete or inaccurate information consumption. In addition, the online information is usually not tailored to each patient's specific needs [3]. Furthermore, users' health literacy levels vary, with some lacking the necessary skills to comprehend medical terminology, assess the relevance of extracted data, or verify the validity of information sources. [4]. Overall, this process can be a challenging experience for individuals seeking health information on the internet [5,6].

Existing referral systems (both manual and algorithmic) such as the knowledge-based (and more precisely rule-based) methods, treat all patient referrals as deterministic—where the system suggests a single best match without considering how confident it is in that decision. Moreover, it appears as if most adopted algorithms of recommendation (at least the ones developed from 2010 to 2022) lack the compatibility with dynamic user modeling, have a limited scope of recommended items, and a

Preprint. Under review.

relatively small sample size in system evaluation[2]. Therefore, the problem we face here lies in how to provide more diverse and dynamic recommendations, i.e. building a recommendation algorithm that can handle large sample sizes, that would be able to make estimations based on detailed patient and professional data, and one that can adapt to new samples added to our pool of patients (in our case, the ability of our algorithm to handle new nodes added to the graph).

1.2 C2 (high level)

By doing so through a graph neural network (GNN) based approach, we can improve the scalability, and adaptability of the recommendations. We also aim to develop a solution that would minimize the need for human input during training, on which knowledge-based and rule-based approaches rely heavily.

1.3 C3

The reason why a solution is needed for this problem is that building a recommendation algorithm that requires constant retraining can prove to be very computationally inefficient, or very rigid to new information input. Unlike transductive methods, which require retraining when new nodes (e.g., patients or symptoms) are added, inductive GNNs like GraphSAGE can dynamically incorporate unseen nodes during inference. This flexibility is particularly valuable in medical contexts where the graph structure is constantly evolving with new patients, diseases, or treatments [7].

Moreover, geometric learning offers more explainable predictions compared to other methods, such as Large Language Models (LLMs). In geometric learning, the model provides a precise prediction, typically represented as the probability of an edge existing in a graph. This allows users to not only receive a prediction but also gain an estimate of the model's confidence in that prediction. In contrast, LLM based methods do not offer the same level of transparency or flexibility, as their outputs do not inherently provide probabilistic measures or confidence estimates for their predictions. This makes geometric learning particularly advantageous for tasks requiring interpretability.

On the other hand, methods that offer better explainability, such as rule-based approaches, often require significant human input and ongoing maintenance after deployment. Additionally, they lack the scalability and flexibility offered by GNN-based models, making them less adaptable for complex tasks or large-scale applications.

1.4 C4

A successful patient triage recommendation system must address several constraints. It should provide personalized recommendations by utilizing each patient's complete medical history and current health status, as they would be ideally stored in the knowledge graph from which the recommendations would be drawn.

Additionally, instead of binary predictions, the model should output probabilities that reflect its confidence in the existence of an edge, ensuring nuanced and interpretable recommendations. More specifically, the system should accurately predict the most suitable specialist for a patient while providing a confidence measure for each recommendation. This is especially important in cases where patient data is sparse, incomplete, or conflicting, as it allows healthcare providers to understand the uncertainty behind the system's predictions.

Scalability is another essential requirement, enabling the model to handle increasing amounts of data and an expanding number of nodes, such as newly added patients, symptoms, and specialists, without requiring significant retraining or reconfiguration. Furthermore, the system must be explainable, offering insights into the rationale behind its recommendations to build trust and facilitate adoption by healthcare professionals.

Generalizability is equally crucial, as the model must perform reliably across diverse datasets from different hospitals, which may vary in structure, scale, and completeness.

Finally, maintaining data anonymity is imperative to ensure patient privacy and comply with ethical standards and regulations.

We also note here that in a world with perfect data available, one would be able to consider creating a recommendation system that directly maps a patient to a specialist, provided that a ground truth from patient to specialist would exist. However, the data available in opensource datasets (such as open source MIMIC III [8] for example) is too sparse to build a satisfying model. The closest we have found to a dataset storing this kind of information is the iBKH dataset, onto which we will expand in the following section of this paper. This dataset constitutes a satisfactory alternative that we can develop with open source data. Using this type of dataset, we would be able to take a series of biomedical markers (anatomy, drug antecedents, symptoms and others that will be explored more in detail in the methods section of this paper) stored in a knowledge graph, and use the entirety of all interactions between these nodes to generate a prediction of which specialist to consult. This in turn would be used (if deployed) by inputting a series of biomedical markers to the model, and then outputting the prediction of edge existence between the graph and a specific therapeutic class (which is the closest node we have in the graph to model what kind of specialist to refer someone to). Then, once we would have identified what correct node models the class of therapeutic medication to prescribe to the patient, we could map this node to a specific specialist through natural language processing (via for example a pre-trained large language model (LLM)). We note here that this in turn allows us also not only to have a more user friendly final output of our model, but it would also allow us to provide (some) context to the prediction.

The natural language output, in addition to the probabilistic edge prediction is aimed at smoothing the potential customer adoption of the algorithm and provide some context to each prediction. This in turn is designed to ensure that the healthcare providers (that are expected to potentially use this system) would be able to understand the predictions of the model, whether the model is highly confident in the prediction or if additionally human input is needed.

2 Methods

2.1 C2 in more detail

As mentioned before, existing referral systems (both manual and algorithmic) treat patient referrals as deterministic without considering confidence in the recommendations. Therefore, our goal is to build a scalable, adaptable recommendation algorithm using network science to handle large datasets, new patient data, and to try to reduce human input during training. For the purpose of this project and to generate a proof of concept, we will be attempting to study the following problem definition : Can GNNs be used in a recommendation setting to generate predictions about which specialist to refer a patient provided a series of biomedical markers? We also recall here the constraints we have selected for our design. First, our model should be able to provide recommendations based not on one specific marker, but on a series of biomedical markers stored in a graph. Our model should also generate predictions that are not binary. Rather, the predictions of our model should be a confidence value we can interpret. The third deliverable of our design is our model's ability to learn from new nodes added to the knowledge graph without having to train the entire graph from scratch every time we add new nodes. Finally, the last deliverables of our model should be that our model ensures the anonymity of patients, and remains an explainable model whose learning behavior can be characterized through a series of insightful metrics.

2.2 C5

In a perfect setting for this project, we would have access to a dataset containing comprehensive information, including patient histories, specialist interactions, and medical outcomes. Unfortunately, the datasets we identified were largely unsuitable for our needs. Some were inaccessible, requiring joint certification from both the student and supervising professor, while others contained data that was excessively incomplete or sparse, making them impractical for training a graph neural network. Given these significant limitations, we opted for an alternative that proved to be a much better fit: the IBKH dataset, which offered a satisfactory solution for this project.

The iBKH (integrative Biomedical Knowledge Hub) is a comprehensive biomedical knowledge graph that integrates data from 17 publicly available biomedical databases, containing over 2.3 million entities across 11 categories (each modeled as nodes). Specifically, the iBKH includes 23,003 anatomy entities, 19,236 disease entities, 37,997 drug entities, 88,376 gene entities, 2,065,015 molecule entities, 1,361 symptom entities, 4,101 DSI entities, 137,568 DSP entities, 605 TC entities,

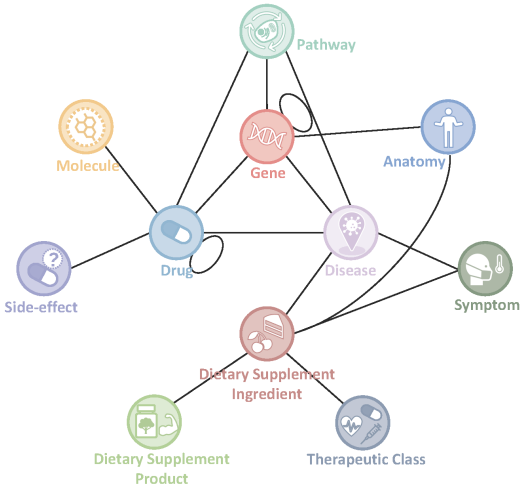

Figure 1: the iBKH dataset visual summary

2,988 pathway entities and 4,251 side-effect entities. When it comes to the relationships in the iBKH knowledge graph, there are 86 relation types within 18 kinds of entity pairs, including Anatomy-Gene, Drug-Disease, Drug-Drug, Drug-Gene, Disease-Disease, Disease-Gene, Disease-Symptom, Gene-Gene, DSI-Disease, DSI-Symptom, DSI-Drug, DSI-Anatomy, DSI-DSP, DSI-TC, Disease-Pathway, Drug-Pathway, Gene-Pathway and Drug-Side Effect. In total, iBKH contains 48,194,646 relations.

Naturally, modelling the entire graph for our project was not possible due to computational limitations, as we are running the entire code on Google Colab (i.e. with the default CPU - Intel Xeon with 13 GB of RAM). We have therefore selected a subset of the graph for our analysis, noting here that the code can be expanded to model each node and edge (provided the computational resources).

The subset of the graph we have chosen for this project is composed of the following nodes : Disease, Symptom, Dietary Supplement Ingredient (DSI), and Therapeutic Class (TC). The edges considered for our subgraph are the following undirected edges : Drug-Treats-Disease, Disease-Present-Symptom, DSI-has adverse reaction-Symptom, DSI-has therapeutic class-TC.

Qualitatively, our graph is composed of 8148 Disease nodes, 1235 Symptom nodes, 752 DSI nodes, 587 TC nodes, and 20123 Drug nodes. Our graph is also composed of 2717947 (Disease, treats, Drug) edges, 3357 (Disease, Present, Symptom) edges, 2093 (DSI, "has-adverse-reaction", Symptom) edges, and 5430 (DSI, has-therapeutic-class, TC) edges. This brings us to a total of 30845 nodes and 2728827 edges.

A summary of the previous information can be found in the following tables.

Table 1: Entities in the iBKH Knowledge Graph.

Category	**Entity Count**
Anatomy Entities	23,003
Disease Entities	19,236
Drug Entities	37,997
Gene Entities	88,376
Molecule Entities	2,065,015
Symptom Entities	1,361
DSI Entities	4,101
DSP Entities	137,568
TC Entities	605
Pathway Entities	2,988
Side-Effect Entities	4,251

Table 2: Node Statistics in the Subgraph.

Node Type	Node Count
Disease	8,148
Symptom	1,235
DSI (Dietary Supplement)	752
TC (Therapeutic Class)	587
Drug	20,123

Table 3: Edge Statistics in the Subgraph.

Edge Type	Edge Count
(Disease, treats, Drug)	2,717,947
(Disease, Present, Symptom)	3,357
(DSI, has adverse reaction, Symptom)	2,093
(DSI, has therapeutic class, TC)	5,430

Table 4: Key Summary Statistics.

Metric	Count
Total Nodes in Subgraph	30,845
Total Edges in Subgraph	2,728,827

There is no "specialist" node modeled in the iBKH dataset. However, the node that models the closest information to which specialist to visit is the therapeutic class node, which refer to the therapeutic class of the medication prescribed to the patient. For example, some therapeutic classes include nodes such as : "psychiatric", "endocrines", or "cardiovascular agents". These can then be mapped respectively to psychiatrist, endocrinologist, or cardiologist using a Large Language Model.

We also observe a discrepancy in how certain nodes are connected compared to others. Indeed, some further investigation was done about the connectivity of certain nodes, and it was found that for each node type, a subset of the nodes is more strongly connected than the rest. More on this can be found in the supplementary material section of this paper.

We have also attempted to generate some visualization for this subset of the graph. After running the code for more than 1h40mins, Collab crashed as a result of limited computational resources. A subset of 10% randomly sampled nodes in our subset of a graph was then plotted for illustration purposes. We note here that some nodes appear to be disconnected form the graph. This is due to the fact some of the nodes sampled randomly were selected without other nodes to which they were connected. The generated graph visualization is available in the supplementary material section of this paper.

As a note, we also mention here that we checked that the graph was fully connected. Because every node appearing in the dataset is linked to at least another node with a specific link, it was expected that the graph would be strongly connected (i.e. we were not expecting any node to be completely separated from the graph). We have confirmed this through testing and found that indeed the graph was fully connected. This assumption would theoretically have to also be tested if this code was to be implemented on the entire iBKH dataset.

2.3 C6

Our proposed design works in the following way. We start off by going over some preprocessing decisions that were made before diving deeper in the actual models.

First, we follow a preprocessing pipeline as displayed in Figure 2 to go from the dataset triplets (node, edge, node) stored in the iBKH information to a data in the form we can use for GNN training. Some samples of the way dataset triplets are stored in iBKH is available in the supplemental material section of this paper.

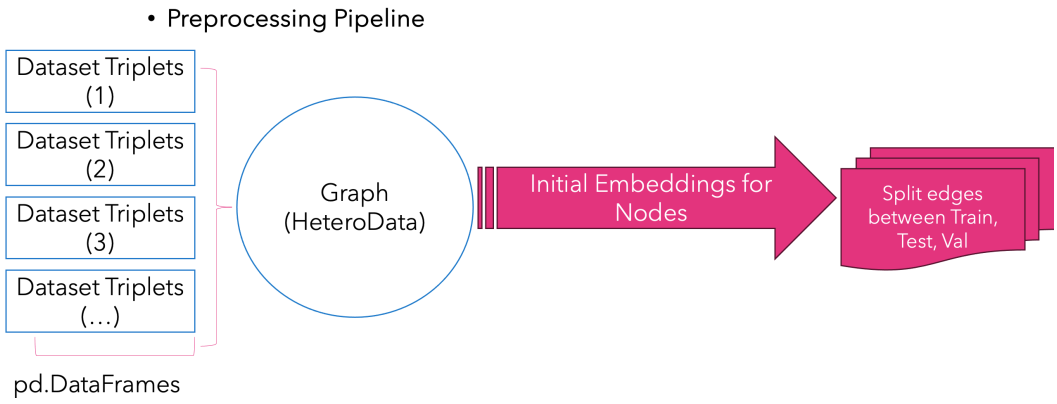

Figure 2: Preprocessing Pipieline

The first step is to store each dataset triplet into dataframes. Then we generate a HeteroData object. A HeteroData data structure, provided by the PyTorch Geometric library, is a container designed for heterogeneous graphs, where multiple types of nodes and edges coexist, each potentially having distinct features and relationships. It organizes data in a dictionary-like format, with node types and edge types serving as keys and their associated feature tensors (or embeddings) as values. For nodes, each type (e.g., Symptom, Disease, etc.) is processed separately, assigning unique numerical IDs through mapping dictionaries, consolidating overlapping IDs, and re-indexing them globally. For edges, source and destination nodes are mapped to their IDs based on these mappings, creating tensors that define the connections. The graph is implicitly undirected by merging overlapping node mappings and ensuring edges can be bidirectionally interpreted when used, resulting in a unified representation of all nodes and edges in the final graph.

Once we obtain the HeteroData object, we initialize embeddings for each of our nodes that we will use during message passing to train our model for the task of edge prediction. Because our nodes do not have any inherent features linked to each of them, we initialize our embeddings in one two ways.

First, we generated embeddings by using the PyTorch Embedding feature, which maps each node to dense vectors of a dimension that we have chosen through hyperparameter tunning (we have finally settled for the size of the embeddings to be 12). These embeddings are then initialized randomly and updated during model training to capture meaningful representations of the graph.

The other method we have used is to simply initialize our embeddings with 1s (i.e. initializing all node embeddings with the same constant value of 1). Both of these methods yielded for similar results.

Once our node embeddings are initialized, we then split our edges of interest (which are the edges leading to the TC nodes, i.e. between the DSI nodes and the TC nodes) in a training set, a testing set, and a validation set. We note here that we split the training edges between some edges for message passing, and others for supervision during training to prevent data leakage. More on the way the edges are split in the supplemental material section of this paper.

After this split, we feed our data to one of two models.

2.4 1. The model we test: SageConv

The model first computes node embeddings using two stacked SAGEConv layers, which iteratively aggregate and transform features from a node's local neighborhood. For two layers of convolution, this corresponds to a message aggregation from neighbors of each node, and neighbors of these neighbors [11]. The two layers are separated by a ReLU activation function layer. The GraphSAGE operator (SAGEConv) enables inductive learning by generalizing the embedding computation process to unseen nodes, as it aggregates neighbor features independently of the graph's specific structure.

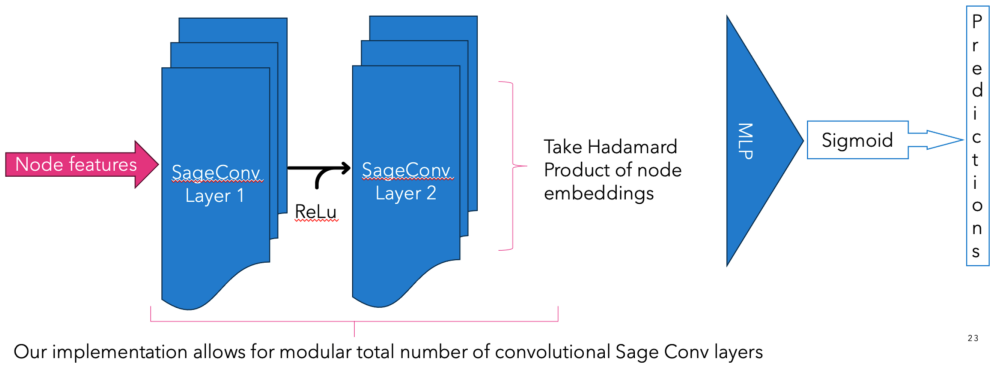

Figure 3: SageConv Architecture

The process begins by computing node embeddings. To predict the existence of an edge, the model combines the embeddings of the two connected nodes using the Hadamard product, which performs element-wise multiplication of the embedding vectors. The resulting vector is then passed through a multilayer perceptron to capture complex interactions. Finally, a sigmoid activation function is applied to the output of the perceptron to produce a probability score that indicates the likelihood of the edge's existence.

The SAGEConv operator enables inductive learning by employing a fixed aggregation mechanism, such as mean, sum, or max pooling, to aggregate information from a node's local neighborhood and compute its representation. This method does not depend on a specific adjacency matrix but instead relies on local connectivity patterns, making it generalizable to new or unseen nodes. This inductive capability is particularly valuable for dynamic or evolving graphs where the structure may change over time or new nodes may appear, ensuring the model remains robust and adaptable in such scenarios.

2.5 2. Our baseline competing model (C8) : GCNConv

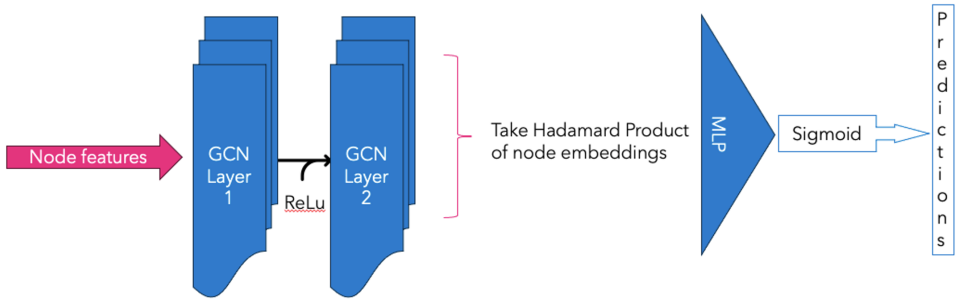

Figure 4: GCNConv Architecture

Our baseline model, on the other hand, leverages two graph convolutional layers (GCN) to compute node embeddings. In the forward pass, the model first uses the GCN convolutional layers to aggregate information from each node's neighbors and refine its representation. After applying a ReLU

activation to the first layer, the second GCN layer generates the final node embeddings, which are then used to predict edges. The predict method computes the element-wise product of the embeddings of the source and destination nodes of each edge, creating an edge feature vector. This vector is passed through an MLP to generate raw scores (logits), which are then mapped to probabilities between 0 and 1 using a sigmoid function. This ensures that the model outputs interpretable edge prediction scores that reflect the likelihood of an edge existing between a pair of nodes, based on their learned representations.

2.6 C7

To evaluate our code, we selected a range of metrics to determine the extent of learning achieved by each model.

First, we have decided to assess how well our models perform on the edge prediction task via 2 standard metrics : the AUROC and AUPRC curve. AUROC provides an overall measure of a model's ability to distinguish between classes across different decision thresholds, giving insight into its general discriminatory power. AUPRC focuses on precision and recall, making it ideal for testing models on imbalanced data with far fewer positive edges (existing edges labeled as 1) than negative edges (nonexistent edges labeled as 0). In some implementations during hyperparameter tuning, positive edges were underrepresented in the training data, thus validating the need for a metric to evaluate model performance on imbalanced datasets effectively.

Another metric we have used is the mean of the predictions made by our model at each epoch. This value is closely related to the value of the negative_sampling_ratio. Indeed, that mean is supposed to reflect the overall expected value of the predictions. For instance, our model's performance should output predictions with a mean of 0.5 if the neg_sampling_ratio is 1 (i.e. if there are as many positive edges as negative edges).

Furthermore, another metric that was considered is the plot of the test predictions against the test labels. We consider that our model is learning if the predictions match the labels as closely as possible.

Finally, the last metric we use to assess the learning made by our model is the curve of training vs validation loss at each epoch. A decreasing curve is a proof that our model's predictions are improving through training. The loss we have chosen to assess both of our models is the binary cross entropy (BCE) loss. BCE is well-suited for edge prediction tasks because it aligns with the binary nature of the problem, where edges either exist (labelled 1) or do not (labelled 0). It effectively handles probabilistic outputs by penalizing predictions based on their deviation from the true labels, ensuring smooth gradient updates for optimization. Additionally, BCE works well with imbalanced datasets. Our loss is then backpropagated to updated the weights of each of our models during their training.

3 Results

3.1 C9

3.1.1 SageConv Performance

We note here that the results we have achieved here are the product of a meticulous hyperparameter tuning. We also note that each set of hyperparameters yielded for vastly different results. However, we have noticed some constants amongst all implementations.

Different aggregation methods were implemented apart from the mean one for SageConv. We noticed that our SageConv operator yielded the best results for the "max" aggregation (also known as the pooling aggregation scheme). In this approach, each neighbor's feature vector is first transformed using learnable weights and passed through a non-linear activation function (e.g., ReLU). Then, the maximum value for each feature dimension across all neighbors is selected, creating a single aggregated vector. This vector is combined with the node's own features to produce an updated embedding. Our hypothesis behind why "max pooling" worked well in our case stems from the fact that this method highlights the most important features from a node's neighborhood by retaining the strongest signals. This approach then condenses the supposed most significant aspects of neighboring

nodes regardless of the number of neighbors. We will also note here this aggregation method ignores the number of neighbors, which may cause for our model to lose structural insights.

The following obtained results for our SageConv model were achieved with a data split of 80% of the edges used for training, 10% for validation, and 10% for testing. We also note that we have used here a disjoint training ratio of 0.2. This ratio corresponds to the percentage of edges not used for message passing. The exact breakdown of how edges were split, and how we did so in a way to prevent data leakage is available in the supplementary information section of this report. The negative sampling ratio (i.e. the ratio of sampled negative edges to sampled positive edges) was chosen to be 1 (for both models, to ensure comparable results). SageConv was trained on 1000 epochs (here again same as GCNConv).

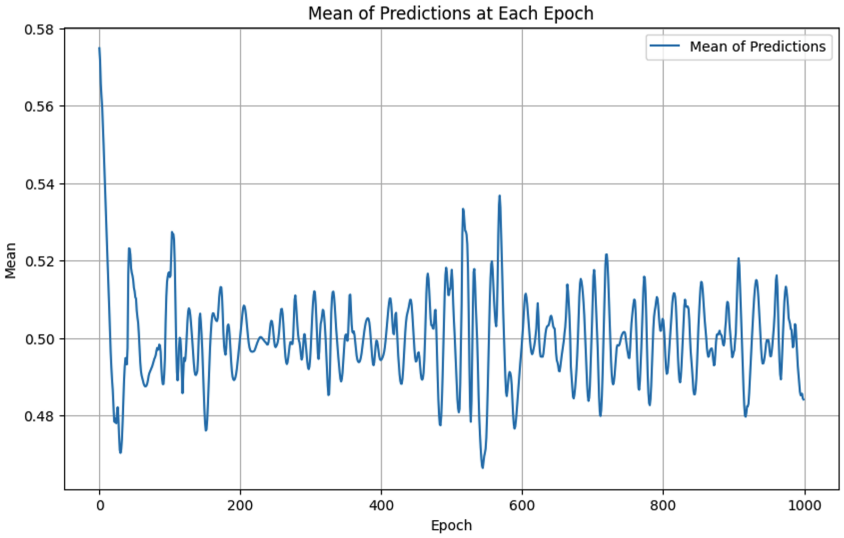

Figure 5: Mean of Predictions at each epoch SageConv

We observe that the predictions of our model converge to 0.5, which reflects the ratio of negative to positive edges in the data. This indicates that the model's predictions align with the chosen underlying edge distribution in the dataset.

Furthermore, the following ROC and PRC curves were obtained.

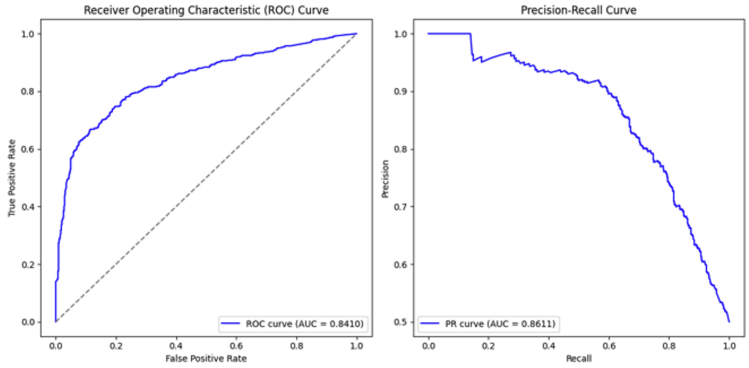

Figure 6: ROC Curve (Left) and Precision-Recall Curve (Right) for SageConv

On the testing set, SageConv achieved a test ROC-AUC of 0.8410, and a PR-AUC of 0.8611. Both of these results indicate that our model has a good ability to differentiate between positive and negative samples overall, regardless of class imbalance. More precisely, these results indicate that 84.10% of the time , our model correctly ranks a positive instance higher than a random negative one. The precision recall tradeoff on the other hand emphasizes the performance of our model on the minority class (positive class usually depending on the implementation). We note here that the PR curve was used during various implementations (where data imbalance was larger) to ensure that our model was correctly classifying both edge types even in the presence of data imbalance.

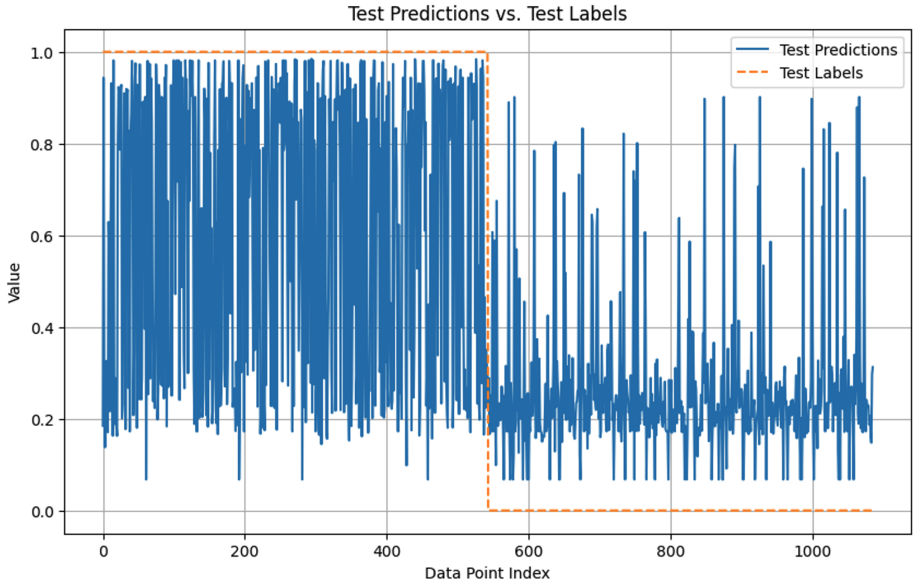

Figure 7: Test Predictions vs Test Labels for SageConv

From Figure 7, we observe that our model portrays a clear learning behavior, as the prediction values seem to follow closely the pattern of test labels.

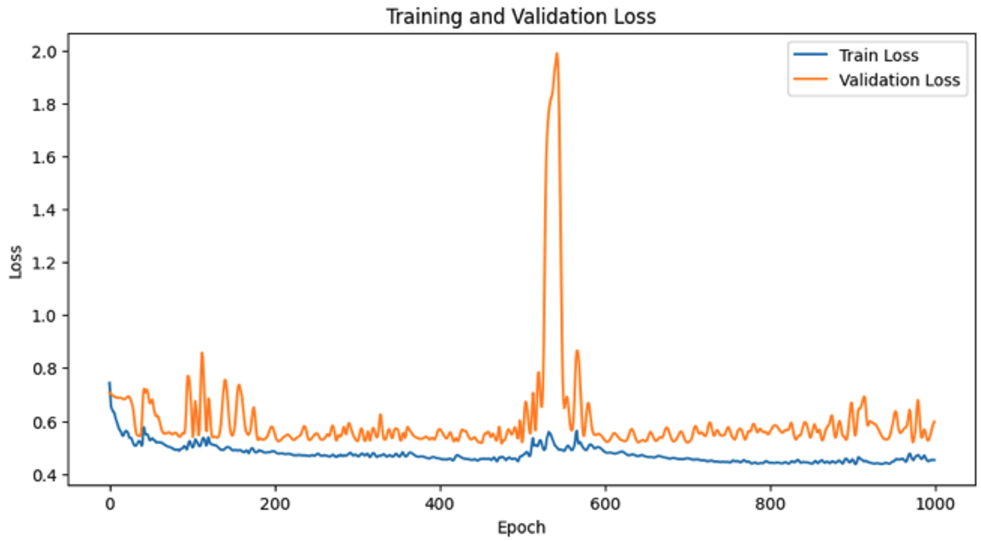

Figure 8: Training and Validation Loss for SageConv

Finally, we observe that the curve of training loss decreases the more the epochs increase. This further supports the claim that learning is done by our model. We also note the same overall pattern for the validation data, although some spikes due to noise are also noticed. We also note that these patterns have an inherently non-deterministic outcome in the sense that they depend on some random initialization and implementation. We also note here that as much as noise is introduced mid-training, it is quickly resolved by our model.

We note also here that our model satisfies the performance requirements previously mentioned. The predictions are not only accurate overall, but also explainable. The results we have achieved provide insightful information as to what exactly is learnt by the model, and the nature of predictions (in the form of probabilities) allows us to interpret the outputs of the model as a confidence measure of the existence of a node (or not). Furthermore, using the SageConv operator from GraphSage allows us to generate embeddings by different aggregation methods of neighboring nodes, which in turn allows our model to be more easily generalizable. This is further supported by the fact that the edges that we train our model on (i.e. the supervision ones) are separate from the ones used for message passing, thus illustrating the inductive capability of our model.

3.2 C10

3.2.1 GCNConv Performance

Here again, we note that the results we have achieved are the product of a fine hyperparameter tuning process. However, we also note here that in general, GCN was found to be somewhat more stable in terms of performance and hyperparameter change.

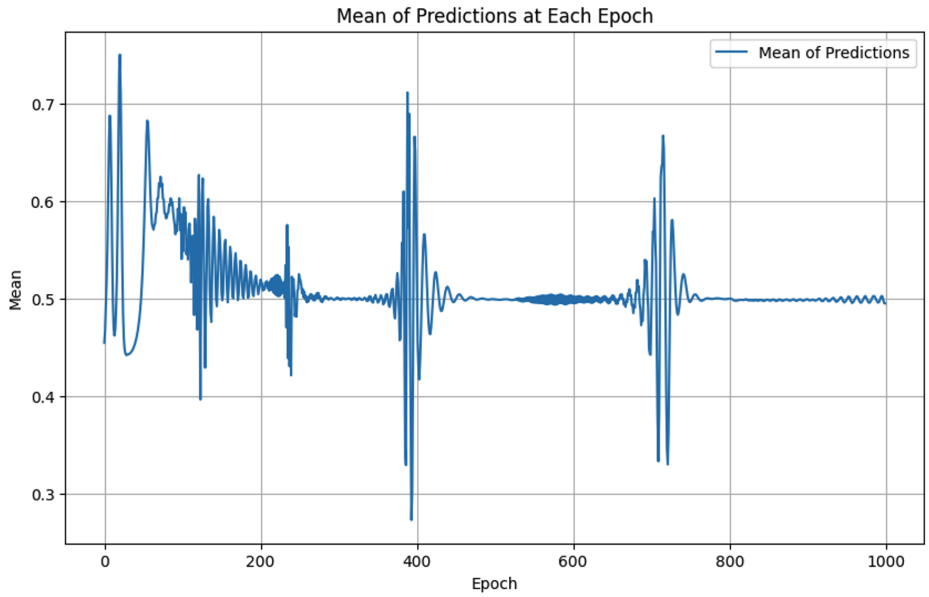

Figure 9: Mean of Predictions at each epoch GCNConv

Compared to our model, we note here that GCNConv is much more stable in terms of converging to the value of 0.5 which represents accurately the proportion of sampled negative edges to positive edges (same as before). We also note that the convergence is more precise and more centered around 0.5. We note however that some spikes are to be noticed, although the model is quick to converge again to a value of 0.5 after each spike.

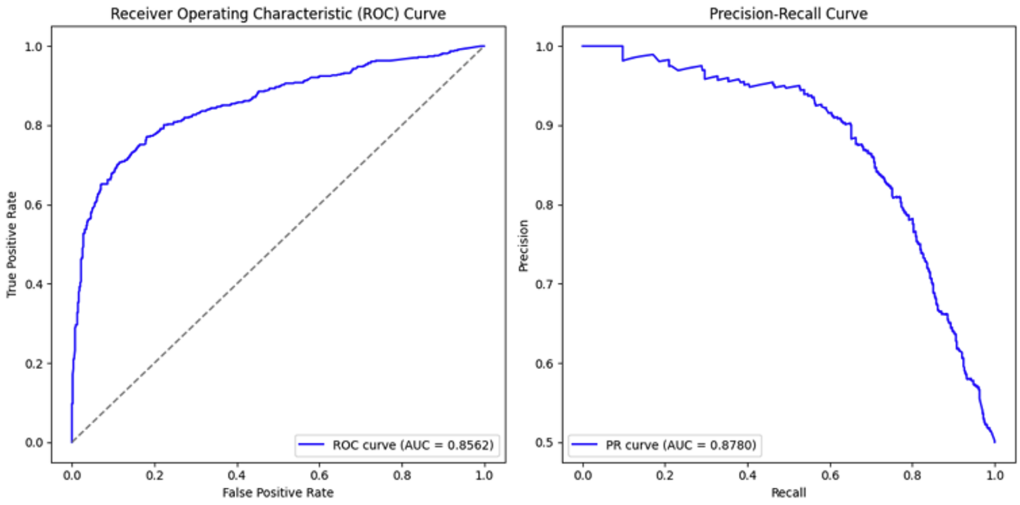

Figure 10: ROC Curve (Left) and Precision-Recall Curve (Right) for GCNConv

We note that GCNConv achieved a similar test ROC-AUC of 85.62%, and a similar test PR-AUC or 87.80%. Similar to the results we have achieved with SageConv, we observe that this performance shows the extent to which GCNConv has a good ability to differentiate between positive and negative samples overall, regardless of class imbalance.

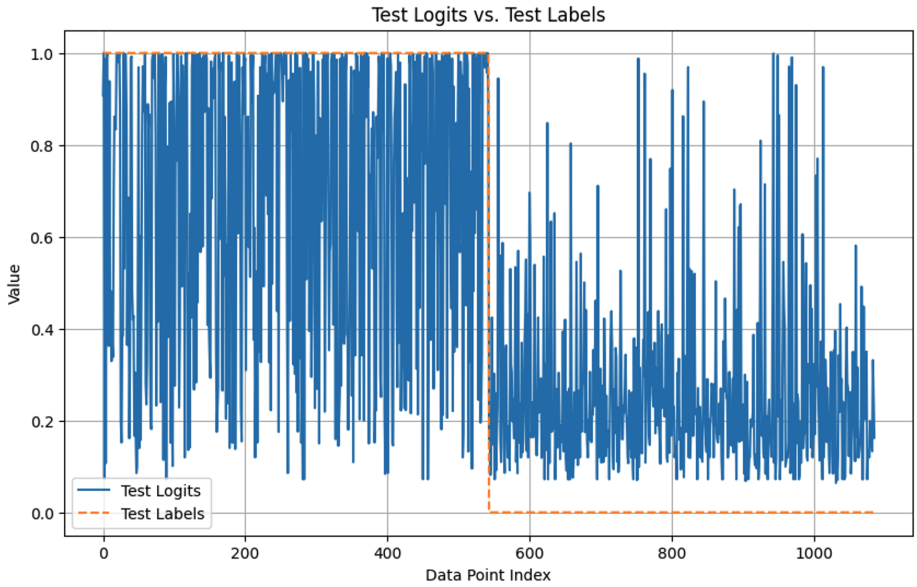

Figure 11: Test Predictions vs Test Labels for GCNConv

Here again, we observe that the predictions of GCNConv are able to closely follow the true value of the test labels. This can be interpreted as the ability of the model to generate embeddings that render the information of an edge existing between them or not.

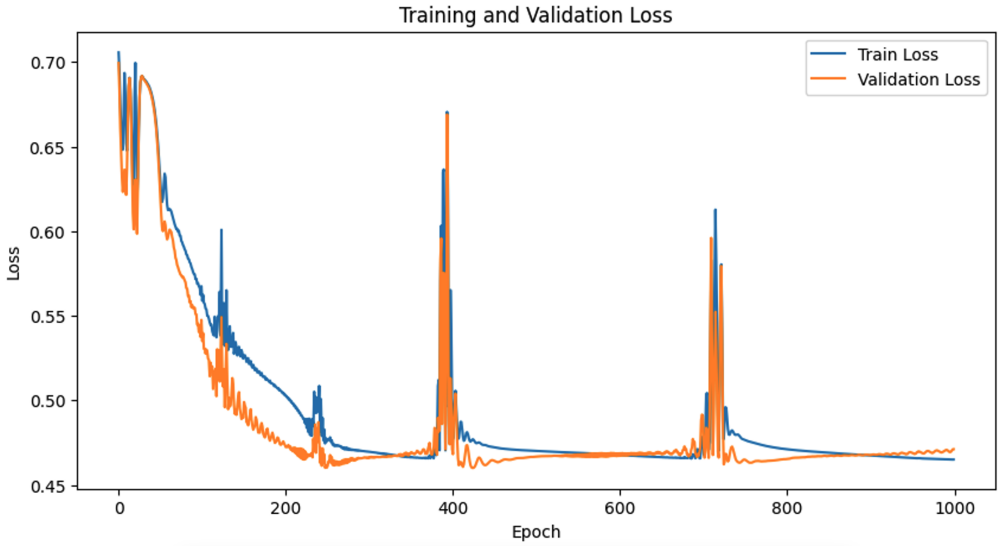

Figure 12: Taining and Validation Loss for GCNConv

We note here that there is a notable difference of learning profile between GCNConv and SageConv. For GCNConv, we observe that the decreasing behavior of the training and validation loss is more clearly pronounced than the one for SageConv. Another observation we can make is that the training and validation curves follow each other more closely than for SageConv. One hypothesis to explain this fact is that the SageConv operator generates its embeddings from neighboring nodes. Therefore, the loss is reduced gradually as neighboring node embeddings converge toward their final embeddings. This is linked for example to the mean prediction profile that does not converge as much for SageConv as it does for GCNConv. The embeddings of GCNConv on the other hand are obtained the following way :

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X} \odot \mathbf{\Theta} \tag{1}$$

Where

$$\hat{A} = \mathbf{A} + I \tag{2}$$

denotes the adjacency matrix with inserted self-loops and D the diagonal degree matrix, and I the identity matrix.

Therefore, this difference in convolution operators may explain the slower convergence and larger convergence interval of SageConv, and the more precise and faster convergence of GCNConv. More on how embeddings are calculated by each method can be found in the supplemental material section of this paper.

We also note that the way GCNConv calculates embeddings is not inductive. Therefore, embeddings for all of the nodes had to be generated during training, and all nodes had to be seen during training. When this was not the case, we could see a direct drop in performance of the model in the edge prediction task.

3.2.2 Output Large Language Model

In addition to the edge prediction task, we have built a large language model that would take as input the TC nodes we would have predicted edges for and outputs some predictions in the form of natural language. In addition to making this prediction more personal, this also allows our model to be flexible with new TC nodes by taking advantage of the large pretraining that has been done on "gpt-4-turbo".

We can cite some example outputs for illustration purposes.

Input TC : *"Helicobacter pylori IgA"*.

Prediction output of our finetuned LLM:

" A patient with concerns related to Helicobacter pylori IgA should visit a gastroenterologist, as this specialist is specifically trained to diagnose and treat conditions affecting the gastrointestinal tract, including infections like Helicobacter pylori."

Other examples :

Input : *"growth hormone"*

Output : *"A patient needing a growth hormone therapy should ideally visit an endocrinologist, as this specialist focuses on hormone-related disorders, including those involving the growth hormone."*

Input : *"Hypertensin" [with the spelling mistake]*

Output : "*A patient seeking treatment for "Hypertensin" should visit a Cardiologist, as this specialist is expert in managing conditions related to the cardiovascular system, including hypertension (high blood pressure), which appears to be associated with the term "Hypertensin".*

We note here that the use of a LLM allows for our model to be more robust to possible mistakes/noise in the data stored in iBKH. For example, our pretrained model directly (and correctly) resolved "Hypertensin" to "Hypertension", and correctly assessed that the patient should go visit a cardiologist.

In a possible future deployment, a user would input a series of biomedical markers to the graph. Then, the model would generate embeddings for each of these nodes and generate a prediction of edge existence between this series of node markers and a TC node. The TC node predicted with the most confidence would then be fed to the LLM which would generate a (more human) natural language output, indicating to the patient which medical specialist to visit visit provided the therapeutic class predicted by the model.

4 Discussion and Conclusion (C11)

To put it in a nutshell, our SageConv model proved to have results that were competitive with existing "non-inductive" methods such as GCNConv. Our model not only generated predictions that matched the distribution of classes within our sample space, but also converged towards definitive predictions. Furthermore, our model did so while remaining inductive. That is, our model managed to have a competing performance in terms of accuracy, precision and recall, while remaining expandable to new data without having to retrain the entire model from scratch.

Furthermore, we note that in general, SageConv displayed a better performance even when trained on fewer epochs, whereas we noticed that the performance of GCNConv was more dependant on a longer training (i.e. many epochs).

We notice that our model satisfied the constraint of being explainable to the end user. Indeed, our models is interpretable by a series of metrics and methods which allow us to study the model's reaction to hyperparameter tuning and new data input.

Our model also generates outputs which are not binary. Rather, these outputs represents the confidence of our model in the existence of an edge.

Furthermore, the addition of the Large Language Model to generate natural language outputs allows for a more human-friendly interaction with a possible end user.

We also note that our selection of iBKH as a training dataset allowed us to comply with our privacy constraints as the data in this knowledge graph is anonymized.

5 Limitations

One of the limitations of our model is its variability and the inherent presence of noise in the result. Indeed, it was noticed during training that the predictions of SageConv were sensitive to noise, and did not offer the same refined convergence patterns as GCNConv.

Another possible improvement of our model would be to use the entirety of the iBKH dataset (provided the computational resources).

Further analysis on the behaviour of our models when fed new data could be conducted. For example, this could be done by training and testing our models' performance on subgraphs of iBKH.

In this implementation, it was confirmed that the subset graph of iBKH we have used for this project was fully connected. However, in the event of future work implementing the previously mentioned methods on the entirety of the iBKH dataset, one should test the "fully connected" assumption again. In that event, (i.e. in the event of the entire iBKH graph to be composed of many individual components), a more refined testing approach would be to train our model on all possible node embeddings, but to only test it on smaller the smaller fully connected subgraphs. The idea behind this testing method would be to have valuable insights on the performance of our model on each of the larger subgraphs composing the iBKH knowledge graph.

A more refined use of the Large Language Model can also allow for better natural language predictions.

This is left as future work to expand this project further.

Supplementary Material

5.1 More information about the hyperparameter tuning :

Many hyperparameters were tuned for both of our models. The full exhaustive list includes:

- Way we initialize our embeddings (torch ones or random initialization of values between 0 and 1). We have noticed that similar results were achieved for both of these embeddings.
- Size of our node embeddings (6, 9, 12). It was noticed that larger node embeddings were related to larger computations for our model. We have finally settled for embeddings of size 12.
- Size of validation set (finally chosen to be 10% of data)
- Size of testing set (finally chosen to be 10% of data)
- Disjoint-train-ratio (which we have finally chosen to be 0.2). This ratio corresponds to the percentage of edges used for message passing (i.e. for supervision). More on this in the following section.
- Adding or not adding negative-train-samples. We have settled on keeping them during training.
- The size of neg-sampling-ratio. This ratio represents the ratio of sampled negative edges to the number of edges. It was crucial to test how well our model would perform with very imbalanced data. Therefore, we have experimented up to ratios of 10 negative samples to 1. Even in very imbalanced datasets, we show that our model's predictions follow closely the true value of edge existence.

Here are some additional results which show that our model predictions portray learning abilities even on imbalanced datasets (for the following plots, the imbalance is set to 10 negative edges to 1 positive edge).

For GCNConv :

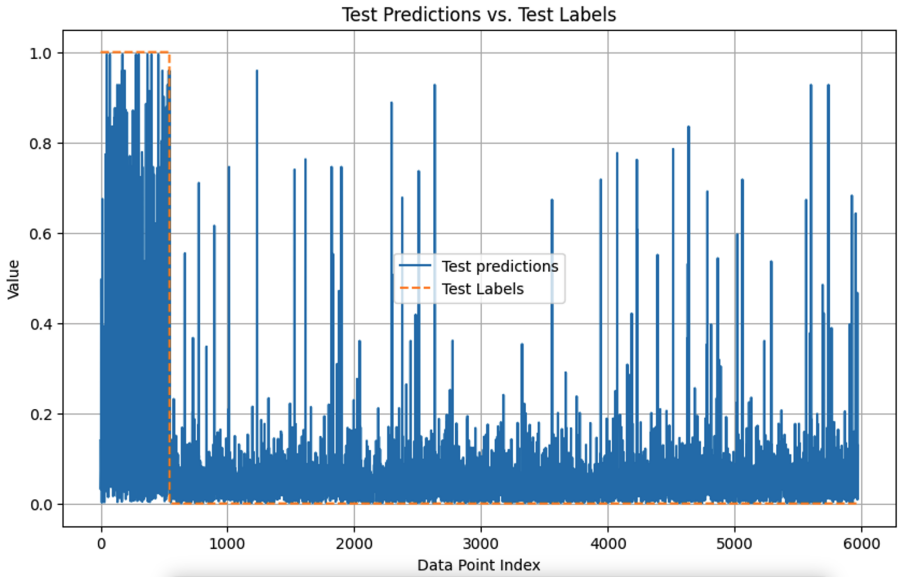

Figure 13: Test Prediction vs Test Labels for large imbalance in dataset GCNConv

For SageConv :

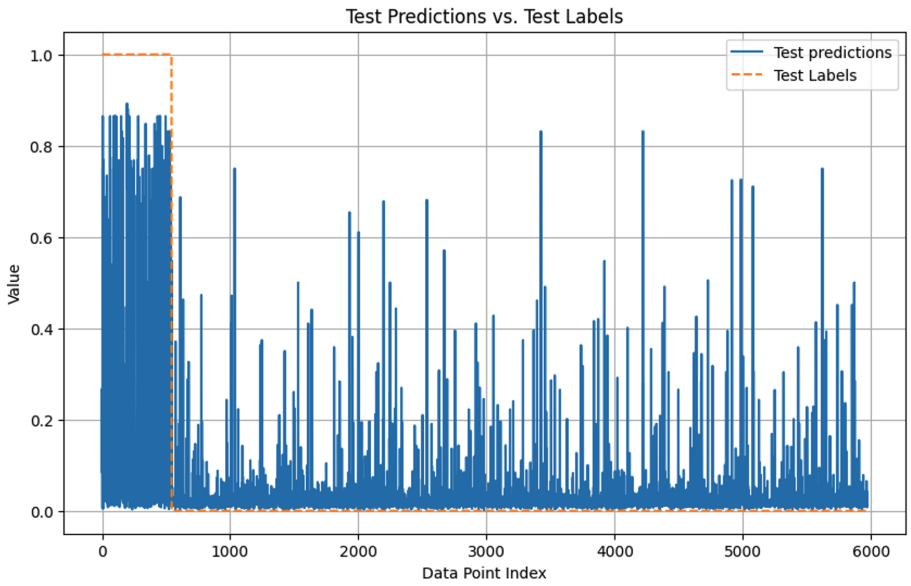

Figure 14: Test Prediction vs Test Labels for large imbalance in dataset SageConv

5.2 More information on how the data was split between training, testing and validation :

The data stored in our heterogeneous graph was split between test, training and validation set using the RandomLinkSplit method. If one wants to explore how data is effectively split between testing, training and validation, this information is stored in the edge_label data structure, and the edge_label_index data structure.

The edges used for message passing are stored in the edge_index data stucture, while the edges used for supervision were stored in a seperate data structure, namely edge_label_index.

In our case for example, we have partitioned our data in the following proportions : 80% for training, 10% for validation and 10% for testing. In total, we had 5430 positive edges between the DSI and TC nodes, which we have split into 543 edges for testing, 543 edges for validation, and 4344 for training. To each of these edges, we have added some negative edges. The proportion of negative edges we have added was controlled by the "neg_sampling_ratio" hyperparameter.

For the training data, the actual number of edges selected for training was controlled by the "disjoint_train_ratio" (DTR) hyperparameter. A DTR close to 0 would imply that most of these 4344 edges were seen during training (and that number of positive edges was then used to set the number of negative edges to add using "neg_sampling_ratio"). On the other hand, a DTR close to 1 would imply that almost no edge is seen during training. This parameter also allowed us to seperate our edges and to prevent data leakage between message passing edges and supervision edges.

In terms of data leakage, we first state that the edges stored in edge_index are used for message passing. In fact, according to documentation, "edge_index" is the default name for the *message passing* edges index, that is, the set of edges we allow the model to send messages across. On the other hand, "edge_label_index" refers to the so called *supervision* edges: the ground truth values we want to compare our model's performance against, with consecutive loss calculation and backpropagation. Both are stored in separate data structures and do not overlap between training, testing and validation. More on this in the following ressources [9][10].

Some definitions about concepts as they were used in this paper:

Definition 1. Heterogeneous Graph: A heterogeneous graph is typically defined as a graph $G = (V, E, Q, \phi)$, in which V is a set of vertices, E is a set of edges, and ϕ is a vertex type mapping function $\phi : V \to Q$. In a heterogeneous graph, the set of vertices can be partitioned into k disjoint

sets $V = V_1 \cup V_2 \cup \cdots \cup V_k$, where $k = |Q|$, $V_i \cap V_j = \emptyset, \forall i \neq j$, and each set of vertices V_i represents a distinct type of node with a unique set of properties).

Definition 2. Inductive Learning: For any type of graph-based model, inductive learning describes its ability to generalize to unseen nodes and edges based on patterns learned on the training graph. Thus, the ability of a model trained on $G_{\text{train}} = (V_{\text{train}}, E_{\text{train}})$ to make inferences on a graph $G_{\text{test}} = (V_{\text{test}}, E_{\text{test}})$ where $V_{\text{train}} \cap V_{\text{test}} = \emptyset$ and $E_{\text{train}} \cap E_{\text{test}} = \emptyset$.

5.3 More information about the connectivity of our graph :

Via study of graph connectivity

We have checked whether our graph was fully connected before feeding it to our models. It was indeed found that our graph was fully connected. This test should also be conducted should this code be used to learn on the entire iBKH knowledge graph.

Via statistics on the connectivity of our nodes

We have run some statistics to know which nodes were the most connected per type in our graph. We have obtained the following results:

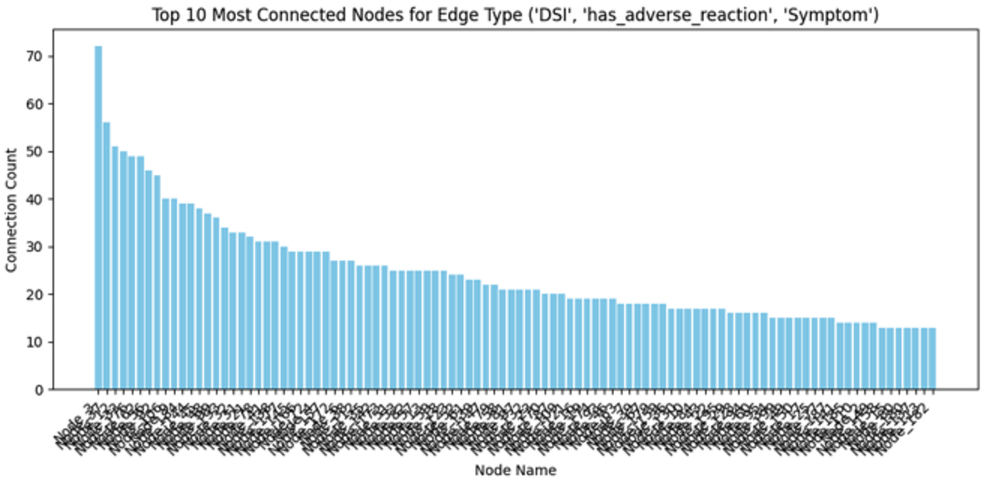

Figure 15: Plot of the 100 Most Connected Nodes for Edge type ('DSI', 'has adverse reaction', 'Symptom")

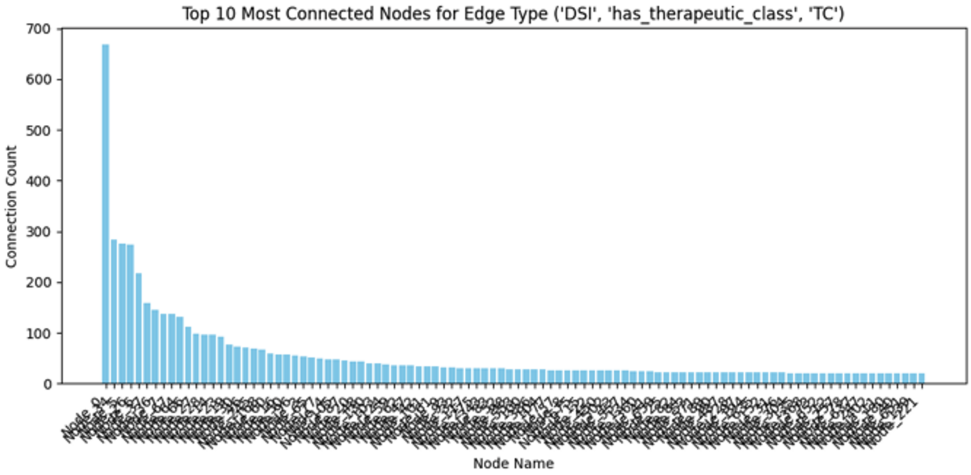

Figure 16: Plot of the 100 Most Connected Nodes for Edge type (’DSI’, ’has therapeutic class’, ’TC’)

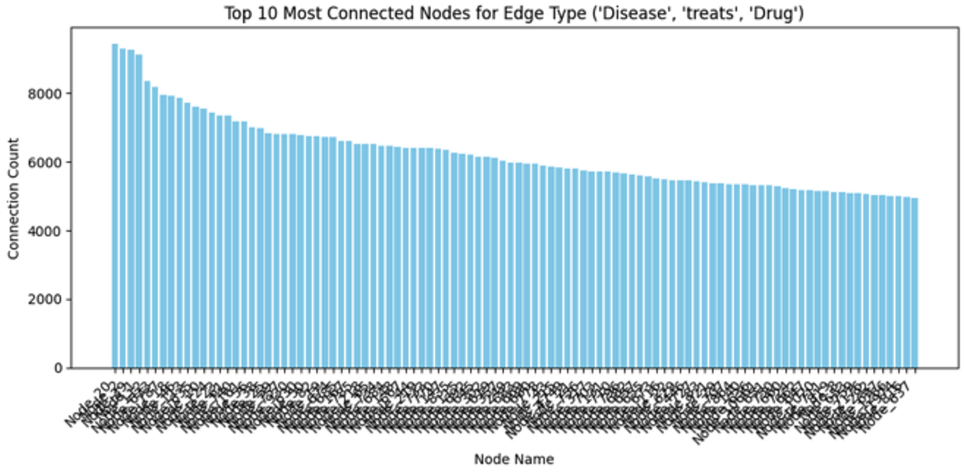

Figure 17: Plot of the 100 Most Connected Nodes for Edge type (’Drug’, ’Treats’, ’Disease’)

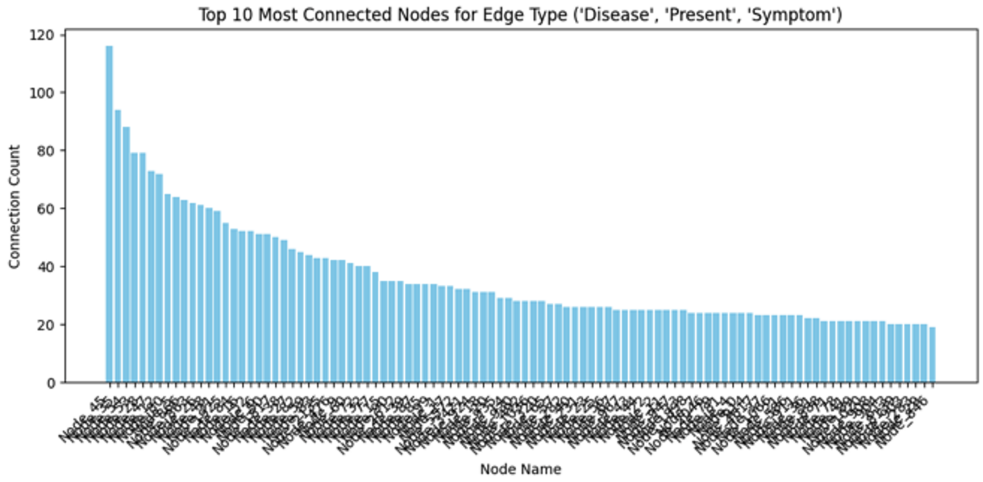

Figure 18: Plot of the 100 Most Connected Nodes for Edge type (’Disease’, ’Present’, ’Symptom’)

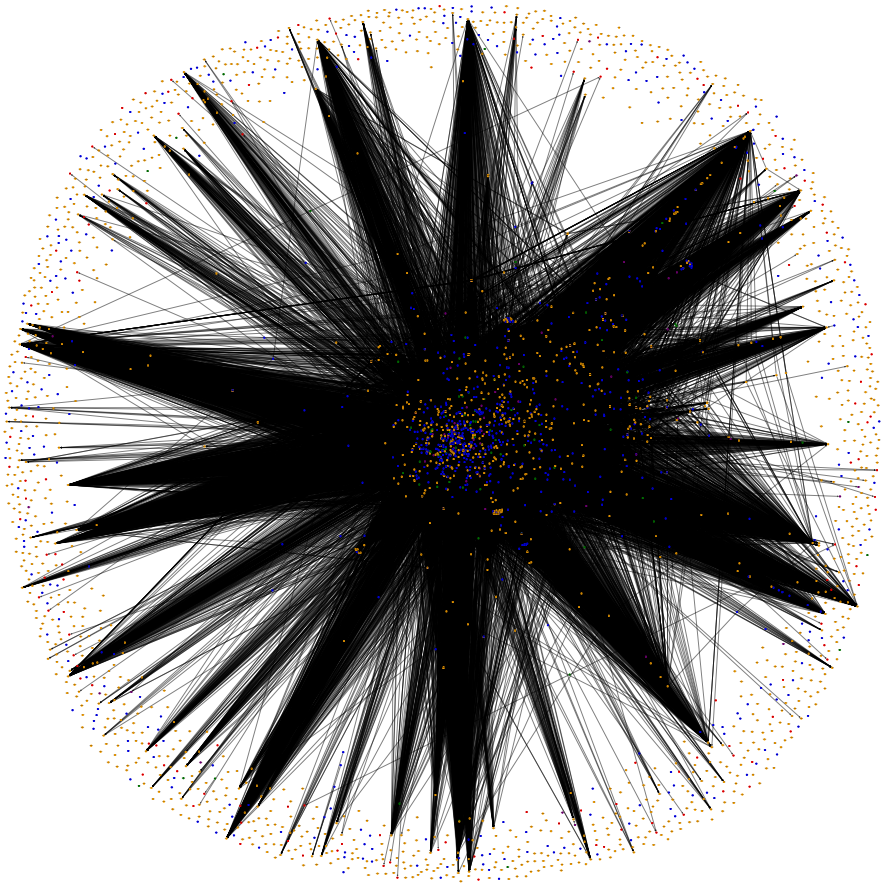

Figure 19: Plot of 10% of our studied subgraph of iBKH

References

[1] Health Canada, "Strengthening the Health Care System in Canada," *www.canada.ca*, Jul. 12, 2024. Available: https://www.canada.ca/en/health-canada/news/2024/07/strengthening-the-health-care-system-in-canada.html

[2] Y. Cai, F. Yu, M. Kumar, R. Gladney, and J. Mostafa, "Health Recommender Systems Development, Usage, and Evaluation from 2010 to 2022: A Scoping Review," *International Journal of Environmental Research and Public Health*, vol. 19, no. 22, p. 15115, Nov. 2022, doi: https://doi.org/10.3390/ijerph192215115.

[3] I. Khaleel, B. C. Wimmer, G. M. Peterson, S. T. R. Zaidi, E. Roehrer, E. Cummings, and K. Lee, "Health Information Overload among Health Consumers: A Scoping Review," *Patient Education and Counseling*, vol. 103, pp. 15–32, 2020, doi: 10.1016/j.pec.2019.08.008.

[4] E. L. Carter, G. Nunlee-Bland, and C. Callender, "A Patient-Centric, Provider-Assisted Diabetes Telehealth Self-Management Intervention for Urban Minorities," *Perspectives in Health Information Management*, vol. 8, p. 1b, 2011.

[5] M. Hardey, "Doctor in the House: The Internet as a Source of Lay Health Knowledge and the Challenge to Expertise," *Sociology of Health Illness*, vol. 21, pp. 820–835, 1999, doi: 10.1111/1467-9566.00185.

[6] M. Benigeri and P. Pluye, "Shortcomings of Health Information on the Internet," *Health Promotion International*, vol. 18, pp. 381–386, 2003, doi: 10.1093/heapro/dag409.

[7] Shaped AI, "Recommender System Family Tree: GNN," Available: https://www.shaped.ai/blog/recommender-system-family-tree-gnn#:~:text=GNNs%20have%20demonstrated%20effectiveness%20in,more%20accurate%20and%20diverse%20recommendations.

[8] "MIMIC-III - Deep Reinforcement Learning," *www.kaggle.com*. Available: https://www.kaggle.com/datasets/asjad99/mimiciii.

[9] Pyg-team, "Understanding how edge_index and edge_label_index relate to message passing," *GitHub*, Discussion #6923, Mar. 15, 2023. Available: https://github.com/pyg-team/pytorch_geometric/discussions/6923. Accessed: Dec. 03, 2024.

[10] Pyg-team, "Split Error in RandomLinkSplit," *GitHub*, Issue #3668, Dec. 10, 2021. Available: https://github.com/pyg-team/pytorch_geometric/issues/3668. Accessed: Dec. 03, 2024.

[11] M. Labonne, "Graph Convolutional Networks: Introduction to GNNs," *Medium*, Aug. 14, 2023. Available: https://towardsdatascience.com/graph-convolutional-networks-introduction-to-gnns-24b3f60d6c95. Accessed: Dec. 07, 2024.