DSBA 6520 Project Final Report

Project Title: A Comparison of Machine Learning Methods in Heterogenous Graph Recommender Systems

Adam Kicklighter

1. Introduction

1.1 Project Overview

1.1.1 Problem Statement

Recommender systems are a critical component of successful eCommerce platforms and digital industry. With the proliferation of goods and services available in the market, consumers face a problem of choice: how to find and select additional offerings that meet their interests. Effective recommender systems create value by presenting users with relevant suggestions that simplify the customer experience. In recent years, Graph Learning based Recommender Systems (GLRS) have emerged as viable solutions to the problem [1]. GLRS approaches represent important objects in a network (users, items, attributes) as nodes and links in a graph and then apply machine learning algorithms to model user interactions with items in the system. The model then makes predictions on which items align with user preferences and recommends them to customers.

Two significant challenges must be addressed to implement GLRS at scale in the real-world: 1). Recommender systems must be able to handle unseen data on large graphs and 2). Recommender systems must be able to handle "cold-start" scenarios where there is little information on new users and new items in the network. For the first challenge, early GLRS attempted to learn the characteristics of the complete network graph [2]. While this proved successful for fixed graphs with constrained size, training on the entire network becomes computationally impractical when the network grows dynamically and reaches a large scale. Additionally, traditional approaches to GLRS do not generalize well to unseen data outside the training set [3]. For the second challenge, early GLRS struggled to learn the relationships between items and users that have few interactions in the network and therefore were unable to make meaningful recommendations in such cases [4]. Cold-start problems compound when the network itself is relatively immature and has insufficient user or item data to accurately represent a sample of the total addressable market of potential customers. Therefore, there is a significant business impact to be gained deploying a GLRS that can meet these challenges and take advantage of the value creation properties in networks that reach critical mass [5].

In response to these challenges, academia and industry have proposed advanced solutions to modeling bipartite graphs of users and items. At the time of this project, the state-of-the-art consists of a family of algorithms descending from the GraphSAGE method, where sub-samples of graph neighborhoods are modeled and used to learn the features of the graph structure before being passed through a layered machine learning stack to generate model predictions. While these approaches deliver good results from low-dimensional node embeddings, is it possible to match or outperform said sophisticated models with more traditional approaches, thereby simplifying the process and reducing operational complexity?

To those ends, this project will compare two approaches in formulating a GLRS to make predictions on unseen nodes in a heterogeneous graph and determine the superior method in an experiment on benchmark data. The first approach will apply an extension of the GraphSAGE family of algorithms to make predictions based on low-dimensional link embeddings of the graph structure. The second approach will use a matrix of one-hot encoded features to create unique vectors for user-item combinations and then predict user ratings as node attributes via a feed forward multilayer perceptron. Given that a multilayer perceptron with at least one hidden layer can universally approximate any measurable function, can an MLP regressor (with appropriate feature engineering) perform comparably to an implementation of the GraphSAGE family [6]? Comparing these two models will illustrate which best solves the simultaneous issue of handling dynamically evolving large graphs and handling cold-start data points in a network.

1.2 Project Approach

1.2.1 Analyze the Results of a Competitive Experiment

Using a demonstration of a link prediction task with Heterogenous GraphSAGE (HinSAGE), analyze the results of the experiment conducted by the StellarGraph library team [7]. The HinSAGE algorithm attempts to address the two foundational challenges by using graph sub-sampling to handle unseen data on large graphs, learning the features of a node's n-hop neighborhood structure to circumvent the cold-start problem. A more detailed description of the dataset, algorithm, and outcome of the experiment will be provided in subsequent sections.

1.2.2 Conduct an Independent User-Rating Prediction Experiment with a Different Methodology

Using the same benchmark dataset that was selected for the HinSAGE experiment, conduct an independent rating prediction test with a feed forward multilayer perceptron and compare the results. This project's implementation of an MLP regressor will iterate across different sized partitions of training data to validate its ability to handle large graphs (rather than training on the entire graph) and its ability to handle cold-start scenarios where there is limited representation of a user or item in a training set (via smaller training partitions simulating instances with sparse records). A more detailed description of the feature engineering, model selection, and evaluation of results will be provided in a subsequent section.

1.2.3 Compare the Experiments and Define Success

To determine which of the two approaches is more successful, this work will use the following evaluation framework. Both the HinSAGE experiment and MLP regressor will score the models using the root mean square error and the mean absolute error. Additionally, both methods will compare a histogram plot of the true ratings and predicted ratings to visually depict the distribution of scores between the ground truth and the model output. If the RMSE, MAE, and histogram from MLP regressor experiment are more performant than the HinSAGE demonstration, said method will be deemed more successful in the user-rating prediction task. When possible, the causal differences in performance will be inferred and an interpretation offered.

Before providing a more thorough description of the methodology, results, and findings, a review of related works will be covered.

2. Literature Review

2.1 Graph Learning based Recommender Systems: A Review [1]

2.1.1 Summary

Wang et al. assert there is no unified formalization of GLRS to date and therefore provide an overview of the state-of-the-art, organizing their summary by types of graph data representations (general interaction, sequential interaction, and side information) and by categories of algorithmic learning approaches (random walks, graph embeddings, and graph neural networks). At a high-level, the authors generalize that a graph $G = \{v, \varepsilon\}$ has users and items represented as nodes in v and interactions/relationships between as edges in ε, from which a GLRS model $M(\theta)$ can be constructed to generate optimal recommendation results R with optimized model parameters θ that are learned from the topological information and content of G. Stated succinctly, $R = arg\,\underset{\theta}{max} f(M(\theta)|G)$. G can be either homogeneous or heterogenous networks and R can be predicted ratings or rankings for item.

This paper sets the groundwork for this project in the treatment of graphs as mathematical models and using algorithms to solve hard computational problems (particularly with node embeddings and neural network approaches). The type of networks in scope focused on bigraphs with edge attributes (where nodes between users and items are links with a user-assigned score to an item). The common link between this paper and the other related works is the holistic overview of the algorithms available to solve GLRS challenges; the subsequent papers take a narrower view on specific subsets of approaches.

2.2 Graph Convolutional Matrix Completion [2]

2.2.1 Summary

Berg et al. developed a model that considers graph link prediction as a matrix completion task. They propose a graph auto-encoder framework based on differentiable message passing on bipartite information graphs. The results are competitive with collaborative filtering benchmarks, but when combined with network feature information (such as social networks), the performance outpaces other state-of-the-art methods. Specifically, they formulate their model as a forward-pass GC-MC model where a graph convolutional encoder $[u, v] = f(x, M_1, \dots, M_R)$ passes and transforms messages from user to item nodes (and vice versa), followed by a bilinear decoder model that predicts entries of (reconstructed) rating matrix $M = g(u, v)$, based on pairs of user and item embeddings. In their experiments, this approach produced results that were superior to five other algorithms, with an RMSE of 0.905 compared to an unweighted mean RMSE of 1.09 from the comparable approaches (max of 1.653 and min of 0.929). Without incorporating side information, the model performance is nearly equal at 0.910, demonstrating strong results even in absence of feature-rich edge attributes. Additionally, the authors test the model against the cold-start problem, predicting ratings for users with only 1-10 other ratings, both with and without features. At its worst performing combination, the GC-MC model registered a 0.98 RMSE – better than 2 of the 5 benchmark cases. Finally, the paper scales the model to test against datasets that are 10x – 100x the size of the initial training set, which illustrates the model's potential to scale to large network graphs.

This paper extended the concept of graph autoencoders to a greater degree and in many ways is a bridge between the higher-level subject survey in 2.1 and the most advanced reading in 2.3. Additionally, treating link prediction as a matrix task is somewhat similar to the method I adopted using a matrix of one-hot encoded features (with and without node attributes) to create vectors of independent variables. I also adopted a similar approach of running experiments with and without side information (node attributes) to evaluate the impact of those additional variables.

2.3 Inductive Representation Learning on Large Graphs [3]

2.3.1 Summary

Hamilton et al. extend the concept of low-dimensional embeddings (graph encodings) into a general inductive framework that efficiently generates features for unseen nodes. This approach overcomes the constraint of previously described methods that depend on encoding the entire graph structure to make link predictions, and importantly, does so in a way that can handle new data that was not present during training. This advancement represents the evolution of training with transductive methods to training with inductive methods – a major step forward in addressing the two foundational challenges described in this project's introduction.

The authors present a model called GraphSAGE, which can quickly generate embeddings for unseen nodes on entirely new subgraphs, thus making real-world applications that require high throughput possible. Instead of training distinct embedding vectors for all graph nodes, the algorithm trains a set of aggregator functions that learn to aggregate feature information from local node neighborhoods.

The process begins by generating embeddings for a graph $G = (v, \varepsilon)$ where the features for all nodes $x_v, \forall v \in v$ are used as input. Each node $v \in v$ aggregates representations of the nodes in its immediate neighborhood $\{h_u^{k-1}, \forall u \in N(v)\}$ into a single vector $h_{N(v)}^{k-1}$. $k - 1$ represents that each aggregation depends on the representations generated at the previous iteration, where $k = 0$ is the base case representation defined as the input node features. After aggregating neighborhood feature vectors, the algorithm concatenates the node's current representation h_v^{k-1} with the aggregated neighborhood vector $h_{N(v)}^{k-1}$, which is then fed through a fully connected layer with a non-linear σ activation function, transforming the representation to be used in the next step of the algorithm. Different aggregating functions can be used throughout the process with different implications on performance. The model can then be tuned with graph-based loss functions that adjusts the weight matrices $W^k, \forall k \in \{1, \dots, k\}$ and parameters of the aggregator function using stochastic gradient descent. The results of the paper's experiments show improved performance by an average of 51% in F1-scores on a classification task over comparable benchmarks.

This paper is ultimately the inspiration for the HinSAGE algorithm developed by the StellarGraph team, which is of course the primary benchmark to which I compare my own methodology in this project. While it handles the two foundational challenges described in the introduction skillfully, are there other means that can provide comparable or superior predictions?

2.4 Project Relevance to Related Works

2.4.1 Project Relevance

This project relates to the literature through its focus on the same core challenges: how can one efficiently optimize the information encoded in network graphs to make meaningful predictions for downstream tasks? The three papers summarized each sequentially build upon the prior approaches before culminating in the impressive GraphSAGE algorithm. The achievement of Hamilton et al. and their advancement of inductive learning methods is a powerful step forward in the field, but their work focused on binary or multiclass classification tasks. How does the GraphSAGE family perform in continuous value predictions, and can those results be produced through the feature engineering of tabular datasets in lieu of building complex heterogeneous graph objects in a computer program? These are some of the essential questions I will attempt to answer in the following sections.

3. Algorithmic Descriptions and Modeling Methodology

3.1 Data Collection and Processing

3.1.1 MovieLens Benchmark Data

Both this work and the HinSAGE experiment were conducted on the MovieLens 100K dataset [8]. This data comes from the GroupLens Research Group founded within the University of Minnesota. It contains ratings from users to movies on a 5-star rating for 100,000 interactions created by 943 users for 1,682 movies, collected between September 1997 and April 1998 by MovieLens, a movie recommendation service. Users were selected at random for inclusion but all had rated at least 20 movies and must have had complete demographic information available. Each user is represented by an id. All of the movies included had at least one rating provided, also represented by an id. Movies may come from any of the following genres: Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, or no genre listed.

The mean rating given by users was a 3.53 out of 5, with a median and mode of 4. The highest mean rating of any user was 4.87 (23 movies rated) and the lowest was 1.49 (435 movies rated). The distribution of the count of movies rated followed a power law, with the average user leaving 106 ratings (mean) but with a median of 20. The highest number of ratings left by any user was 737 and 36% of the users left more ratings than the mean average.

The summary statistics for the list of movies were similar. The highest mean rating of any movie was 5 (with no more than 3 reviews for perfectly rated movies) and the lowest was 1 (with no more than 5 reviews for such films). This count distribution was likewise a power law, with the mean average number of ratings left was 59 (median 27 and mode of 1). The highest number of ratings for a single movie was 583 (with the lowest of 1), and 32% of movies had more ratings than the mean.

Three datafiles were used to compile the training dataset. The first principal file is a dataframe of user-movie ratings, where each line of data represents one rating of one movie by one user, and has the format `userId,movieId,rating,timestamp`. Users and items are numbered consecutively from 1 and the data is randomly ordered. Ratings are made on a 5-star scale from 1 to 5. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

The second principal file is a dataframe of movie features, which includes each movie listed sequentially by its id, along with its title, release date, URL to its imdb page, and a vector encoding of which genres it can be categorized, corresponding to the genres listed previously in this description. The genre categorical variables were already transformed into one-hot encodings by the GroupLens team.

The final principal file is a dataframe of user features including the user id (also listed sequentially), along with the user's age, gender, occupation, and zip code. Ages range from 7 to 73 and possible occupations may be administrator, artist, doctor, educator, engineer, entertainment, executive, healthcare, homemaker, lawyer, librarian, marketing, none, other, programmer, retired, salesman, scientist, student, technician, or writer.

Additional files are available in the MovieLens 100K compressed datasets, and their descriptions are reviewable in the Read Me in the GroupLens repository [9].

3.1.2 Data Processing

The following steps were taken to render the dataframe(s) amenable for modeling. First, a concatenation of the userId-movieId was taken for each of the 100,000 observations, representing an event id of the user-movie rating instance. Next, the contents of the rating dataframe were appended to this event id without modification. Following that, the age, gender, occupation, and zip code of each user was mapped to the corresponding user ids for each event. Finally, a similar approach was completed for movie attributes, where the title, release date, and genre encodings were mapped to each of their respective movie ids.

A new column for year was created to extract only the year value of the release date variable. Of the 100,000 observations, 9 had null values for this year feature. Said nulls were imputed based on the mode of other observations, which was 1996. Similarly, 129 observations had nulls for zip code, and nulls were again imputed by the mode of 55414 (a zip code in Minneapolis, Minnesota). Ultimately, zip code was not used as a feature in the modeling dataset and was discarded, but it was retained for this step of the process. This phase resulted in a dataframe that included the event id, user id, movie id, rating, time stamp, user age, user gender, user occupation, user zip code, movie title, release date, year, and genre encodings for each user-movie rating event.

As for the dataset used by the HinSAGE experiment, the authors used only the user id, movie id, and ratings links. The experiment in this work ran iterations of predictions both with and without the additional user and movie features to determine the impact of the graph structure alone and the impact of node attributes on the evaluation metrics.

3.2 Algorithmic Descriptions: HinSAGE

3.2.1 Heterogenous GraphSAGE (HinSAGE)

HinSAGE [10] is a generalization of the GraphSAGE model postulated by Hamilton et al. [3] and extended to heterogeneous graphs with different node and edge types. As such, much of the mathematical notation and formalisms are similar to those laid out in subsection 2.3.1. A detailed reference from its documentation is available in **Appendix III**.

The model uses a two-layer approach: 1). Output a pair of node embeddings for users and items from an input of labeled user,item node pairs corresponding to user-item ratings 2). Feed the embeddings into a link regression layer that applies a binary operator to the node embedding (concatenating them) to construct link embeddings; link embeddings are then passed through the link regression layer to obtain predicted user-item ratings. The model minimizes the loss function (MSE between predicted and true ratings) using SGD. The problem is treated as a supervised link attribute inference problem on a user-item network with nodes of two types (users and items) and links corresponding to user-item ratings, where ratings are real number values (range 1 to 5).

3.2.2 Updating Features for Homogenous Graphs

Features are first aggregated from the neighbors of a given node and then passed through a layer under one of two methods depending on the conditions around concatenation. The output for a given node in a certain layer is either the concatenation of or the addition of a weight matrices for the given node and for its neighborhood, which is then added to an optional bias. The weight matrices are trainable parameters (optimized via gradient descent) and are shared by all nodes. The results of the operation on the matrices and the bias are then passed through a nonlinear activation function, which serves as the input of the next forward layer.

3.2.3 Handling Heterogeneity

Additional weight matrices are provided to account for heterogeneous data. The GraphSAGE model is extended by having separate neighborhood weight matrices for every unique ordered tuple (N1, E, N2) where N represents different node types and E represents edge types. Additionally, every node type will have its own self-feature matrix. Thus, there is a unique weight matrix for each distinct node type plus a unique matrix for each distinct node type/edge type connection with the other node types.

Once the set of matrices are available, the features of a node neighborhood are aggregated as they were in a homogeneous scenario. Concatenation can be conducted either for just a single edge type or for all edge types between the node type and its neighbors. Alternatively, the self and neighbor matrices could be added together instead of concatenated, just as they were for homogenous types. This operation is then passed through a nonlinear activation function and fed onto the next layer.

In both homogenous and heterogeneous cases, the mean aggregator function could instead be swapped with a GCN aggregator.

3.2.4 Layers & Loss Calculations

HinSAGE takes a two-layer approach to supervised inference. The HinSAGE model itself generates a final layer that represents node embeddings of the target node and its neighborhood, and then an additional Keras layer sits on top of those embeddings to accomplish the downstream task – in the benchmark experiment's case, link prediction. With those two layers stacked, the algorithm can be optimized end-to-end in the same process, tuning the matrices in the node embedding layer to minimize the loss function from the Keras layer.

3.2.5 Additional Layers & Loss Calculation for Link Prediction

With the node embeddings generated for the disjoint nodes (users-objects), HinSAGE then combines those embeddings with a binary operator that obtains embeddings representing links. For edges such as the HinSAGE demo's dataset, a simple concatenation produces reasonable embeddings.

3.3 Algorithmic Description: Multilayer Perceptron Regression

3.3.1 MLP Regressor

The multilayer perceptron regressor is an application of a traditional feedforward artificial network with fully connected nodes between the input, hidden, and output layers. All of the input features for a given observation in the training dataframe are fed into their respective nodes, where they are linearly combined with the product of their weights and biases before being fed into an activation function. The output of that layer is then fed as the input into the subsequent layer and the process is repeated before culminating in (for this case) a single continuous value of output (a predicted rating for a given movie from a given user). The weight matrices for the nodes are iteratively updated and optimized through a loss function via stochastic gradient descent. The impact that a neural network's architecture can have on modeling the objective function is significant and variable [6]. As such, this project ran different iterations of the experiment with different model architectures (depicted in Table 1 of the Experimental Results). A full reproduction of the code to implement the MLP regressor used in this work – as well as the necessary processing steps prior to the model – can be found in **Appendix II**.

3.3.2 Architecture and Hyperparameters

Two layer architectures were used in the experiment. The first used 100 nodes in a single hidden layer (per the default of the library used). The second used 32 nodes in each of two hidden layers (following the approach of the HinSAGE experiment). The performance of each architecture is compared in Table 1 of the experimental results section. Of note, selecting an optimal layer design is a matter of domain expertise and practitioner experience. As pointed out in the *Neural Smithing* text, there is yet to exist an analytical method to determine the optimal layer depth and width of fully connected ANNs, and it is a near certainty that the ones used in this experiment are not the most performant possible design [6]. However, the initial results were promising enough with the two architectures used that further layer adjustments were not made. Additional experimentation with varying architectures and grid searched cross validation is an opportunity for future work which could improve performance.

Similarly, there is a wide range of possible input values for the MLP hyperparameters. A full list of available options and their default values can be found in the library documentation [11]. For this project's implementation of the MLP regressor, no activation function was applied to the linear combination of input, weight, and biased operations – the linear output of the network was passed on. The 'adam' variation of the SGD optimizer was used to conduct the back propagation, and the number of epochs were limited to 20 iterations (again matching the HinSAGE parameters).

3.3.3 Feature Engineering

Before training the MLP regressor, further work needed to be done to the input dataframe to make the features meaningful for learning. In addition to the work described in section 3.1, one-hot encodings were generated for users, movies, occupation, year, and gender and then concatenated with the genre encodings already provided by the original datasets. K-1 features were used for each of these one-hot encoding sets. Finally, the values for age were scaled with a min-max scaler, rendering all independent variables in the dataframe as values between 0 and 1. The final dataframe was a 100,000 by 2,734 matrix of rows and columns, where ratings, event ids, user age, user gender, user, movie, year, and genre were the available dependent and independent variables.

This step, although ubiquitous and common across machine learning practices, was one of the primary contributions I made in building a GLRS and comparing the results to other approaches. As described in the experimental results sections, the transformation of the input data in this format led to superior results on the evaluation metrics out-of-the-box (without the side information node attributes). Said differently, rendering the bipartite user-movie graph as an adjacency matrix that represented a user-movie interaction as either a 0 or 1 encoded rich enough data for the MLP regressor to make competitive predictions without factoring in the node attributes. The results imply possible data leakage, with the network simply learning the average rating given by a user and the average rating given for a movie and predicting the intersection of the two. Perhaps the network simply learned the identity of users and movies? If this were the case, it would present a material hurdle to solving the two foundational challenges described in the paper introduction. If the MLP regressor function was simply memorizing the average ratings for node combinations, it would inadequately handle cold-start scenarios or node combinations all together missing from the training data, as would be the case in a production environment with the user-item churn in industry. However, even when keeping the training partition unrealistically small (0.10 of the dataframe), the MLP regressor predicts movie ratings with an RMSE in-line with the HinSAGE algorithm (1.051 vs. 1.036). This improves to 1.043 if one includes the node attribute features. Thus, it’s reasonable to conclude that the MLP regressor is learning generalizable features of the graph structure even from small samples of the training data rather than the potential data leakage issue raised above.

Finally, it is also remarkable how my approach is evidently not penalized for increasing the dimensionality and sparsity of the feature space. However, this would presumably be a limitation of my approach in a production setting, where creating a one-hot encoding variable for every user-item combination would create a considerable computational and operational burden on a network with billions of nodes, as seen in prominent industry applications. Running a comparable experiment against datasets from such contexts would presumably show that HinSAGE’s low-dimensional embeddings and sub-sampling methods would be more performant in both run time and prediction metrics. A possible area of opportunity for future work would be discovering a threshold where this trade-off becomes material if that is the case.

3.4 Experimental Methodology

3.4.1 Iterations and Design Principles

Before moving onto the results, I will briefly describe the approach to iterating the experiment. The MLP regressor model was trained and tested in 16 iterations. Each of the two architectures (1 hidden layer and 2 hidden layers) were run in batches that included or excluded the additional node attribute features, and each of these combinations of architectures/features were repeated 4 times at different training partition sizes (0.3, 0.5, 0.7, and 0.9). This approach was meant to establish the predictive impact of the node attribute features (age, gender, genre, release year) and the model’s ability to handle novel or unseen data (assuming smaller training partitions simulated cold-start scenarios or dynamically evolving graphs). If the predictive results on these smaller training sets were within range of the larger partitions, one can infer how well the approach generalizes or overfits, if at all. The results of these 16 iterations and the HinSAGE experiment are compared in Table 1.

4. Experimental Results

4.1 Overview

4.1.1 High-Level Results

From the outset, all the MLP regressor iterations outperformed the HinSAGE experiment. The worst performing variation (single layer, node attributes excluded, training partition of 30%) produced an RMSE 6.5% better than the HinSAGE results (0.968 vs 1.036). The best performing iteration (2 layers with node attributes and a training partition of 90%) produced an RMSE improvement of 9.4% (0.939 vs 1.036). However, said training partition is rather large in practice and would raise questions of over-fitting with poor generalization in production. As a result, the following analysis and interpretation will focus on comparable training partitions to the HinSAGE experiment at 70% of the dataframe. Subsequent examination will compare the HinSAGE results to both the MLP regressors architectures with and without node attributes at a 70% train-test split.

4.1.2 Scoring Matrix

Table 1

Model Type	Layer Architecture	Training Partition	Node Attributes	RMSE	MAE
HinSAGE	(32, 32)	0.7	No	1.036	0.833
MLP Regressor	(100)	0.9	No	0.941	0.747
MLP Regressor	(100)	0.7	No	0.949	0.750
MLP Regressor	(100)	0.5	No	0.952	0.752
MLP Regressor	(100)	0.3	No	0.968	0.762
MLP Regressor	(32, 32)	0.9	No	0.939	0.742
MLP Regressor	(32, 32)	0.7	No	0.948	0.747
MLP Regressor	(32, 32)	0.5	No	0.954	0.762
MLP Regressor	(32, 32)	0.3	No	0.965	0.761
MLP Regressor	(100)	0.9	Yes	0.939	0.744
MLP Regressor	(100)	0.7	Yes	0.946	0.747
MLP Regressor	(100)	0.5	Yes	0.950	0.749
MLP Regressor	(100)	0.3	Yes	0.965	0.760
MLP Regressor	(32, 32)	0.9	Yes	0.939	0.740
MLP Regressor	(32, 32)	0.7	Yes	0.947	0.749
MLP Regressor	(32, 32)	0.5	Yes	0.949	0.752
MLP Regressor	(32, 32)	0.3	Yes	0.946	0.766

4.2 Results and Findings

4.2.1 Impact of Node Features

As discussed in the prior section, training the network on the graph adjacency matrix alone was sufficient to produce more performant results, but including the note attribute side information did improve RMSE by 10 to 30 basis points for the respective MLP architectures. An area of possible future work could lie in analyzing which of the node attributes (user age, user gender, user occupation, movie release year, movie genre, etc.) contributed most to the performance gains, but for the sake of this project, the improvements were somewhat immaterial to the overall scope, so said investigation was not conducted.

4.2.2 Model Comparisons

Figure 3 and 4 in **Appendix I** depicts the evaluation metric performance across the algorithms in scope. At a 70% training partition, the best performing model was the MLP regressor with one hidden layer with node features included. As such, this version of the MLP was used to generate the training loss (figures 1 and 2) and histogram plots (figures 5 and 6) for comparison to the HinSAGE performance.

Both models used the mean square error as the loss metric during training, and the plots depict the performance across 20 training epochs. While the HinSAGE model continued to see improvement across epochs 2 through 20 (approximately a 5% improvement in MSE), the MLP regressor reached near-optimal performance on the loss metric after only 1 epoch. In a one-off experiment, the same architecture was trained across 500 epochs to validate the improvement in the loss measures. The run-time was interrupted after 40 minutes of training on 1 GPU, so the results are unavailable to interpret, but given that the model had yet to converge after an extensive time, one can infer that further optimization of the MLP regressor is possible but will likely results in negligible improvements given the plot of this experiment.

Finally, examining the histogram of true vs predicted ratings for both models can offer some insight into the MLP regressors superior results. The HinSAGE plot (figure 5) shows a higher concentration of predicted scores clustering between a rating of 3 and 4, while the MLP regressor plot (figure 6) shows more distribution of predicted ratings between 2 to 3 and between 4 to 5. Based on this visual examination, the MLP regressor was better able to account for user ratings in the tails of the histogram, and the density of the plot more closely follows the distribution of the ground truth curve. An area of possible future work would be limiting the prediction of both models to integer values, either by rounding to the nearest integer or some other constraint) and evaluating the performance. Given that the dataset does not allow ratings with decimal numbers, how much could predictions be improved by imposing the same condition on the predicted ratings?

4.2.3 Interpretations and Conclusions

Based on the experimental results and evaluations, one can conclude that the MLP regressor produced superior results to the HinSAGE algorithm and would thus lead to a better Graph Learning based Recommender System. Given its lower root mean square error and mean absolute error across all architecture varieties, the perceptron algorithm was better able to predict the rating a user would assign to a movie. Implementing this capability into a GLRS would mean that the MLP regressor could predict the score a user would assign an unseen movie and the system could rank those unseen movies from high-to-low before presenting the user with the movies with the highest predicted ratings as recommendations for viewing.

But what inferences could one draw about *why* this was the case? One possibility is that what the HinSAGE approach gains in flexibility and speed via node neighborhood sub-sampling and learning from aggregated graph features it losses in thoroughness. In other words, the MLP regressor is training on the entirety of the feature set (+2,700 dimensions in this work) while the HinSAGE method is optimizing a function that learns how to aggregate features from near-by-neighborhoods. This additional layer of abstraction might compromise the model’s ability to competitively generalize. This is essentially a comparison of a low-dimensional approach against a high-dimensional approach. That said, is it wise to emphasize the MLP regressor’s superior evaluation metrics over the HinSAGE algorithm’s flexibility? Not necessarily. In the constraints of this experiment, it’s convenient to draw that conclusion, but in high-throughput industry applications, it could very well be the case that the HinSAGE approach produces better network adoption and user retention through runtime optimization and “good enough” recommendations. Similar to a PageRank study, is having the *most precise* rank of an item set more performant than simply having the correct rank in order? As is often the case, a real-world comparison of two GLRS built from these different algorithms may come down to product analysis, such as A/B testing, user retention, cost to acquire new users, time spent during sessions, and other financial or business KPIs, rather than statistical evaluation metrics. In industry, the best evaluation metric is sometimes humorously said to be revenue, and such an analysis may lead to different conclusions than the ones presented in this experiment.

But with the information at hand, the MLP regressor wins the day.

4.3 Possible Future Work

4.3.1 Additional Opportunities for Advancement

Throughout this work, various suggestions for future work and further experimental opportunities have been mentioned. Succinctly restated, some of these have been architecture and hyperparameter optimization for the fully connected feed forward layers in both algorithms via grid search cross validation, discovering a threshold for network size where the performance trade off in high-dimensional vs low-dimensional approaches is crossed, establishing which types of node attributes yield the largest improvements in predictive performance, and constraining the values of model output to the same possible integers from the real-world dataset. Others not previously mentioned include supplementing the graph structure information with the node attribute features for the HinSAGE algorithm. It’s possible including those additional features would return only marginal improvements for the evaluation metrics (as seen with the MLP regressor) but perhaps the incremental gains would be more appreciable. Finally, one could also test additional continuous-value prediction methods against both results, such as OLS, gradient boosted trees, multi-adaptive regression splines, etc. to determine if yet another option is preferable to those already tried.

Regardless, GLRS research will surely continue to be an active field of inquiry as competitors race to acquire the requisite number of users for their networks to hit critical mass. Time will tell whether increasingly sophisticated algorithms, clever applications of currently existing technologies, or a synthesis of both lead to the best performing recommender systems. Either way, as the saying goes, the blend of graphs and machine learning will have much to recommend it as GLRS evolve to provide more value to the market.

References

[1] [Wang et al., 2021] Shoujin Wang, Liang Hu, Yan Wang, et al. Graph Learning based Recommender Systems: A Review. *arXiv preprint arXiv:2105.06339v1 [cs.IR] 13 May 2021*

[2] [Berg, et al., 2017] Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph Convolutional Matrix Completion. *arXiv preprint arXiv:1706.02263v2 [stat.ML] 25 Oct 2017*.

[3] [Hamilton et al., 2018]. William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. *arXiv preprint arXiv:1706.02216v4 [cs.SI] 10 Sep 2018.*

[4] [Maksimov et al., 2020] Ivan Maksimov, Rodrigo Rivera-Castro, and Evgeny Burnaev. Addressing Cold Start in Recommender Systems with Hierarchical Graph Neural Networks. *arXiv preprint arXiv:2009.03455v2 [cs.LG] 1 Dec 2020*.

[5] [James Currier et al., 2022] James Currier and the NFX Team. The Network Effects Bible. https://www.nfx.com/post/network-effects-bible

[6] Russell Reed, Robert J Marks II. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Bradford Books. 1999.

[7] Link Prediction with Heterogenous GraphSAGE (HinSAGE). https://stellargraph.readthedocs.io/en/stable/demos/link-prediction/hinsage-link-prediction.html

[8] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

[9] MovieLens 100K READ ME. https://files.grouplens.org/datasets/movielens/ml-100k-README.txt

[10] StellarGraph Documentation. Heterogenous GraphSAGE (HinSAGE). https://stellargraph.readthedocs.io/en/stable/hinsage.html

[11] MLPRegressor Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

Appendix I

Figure 1

HinSAGE Training Loss

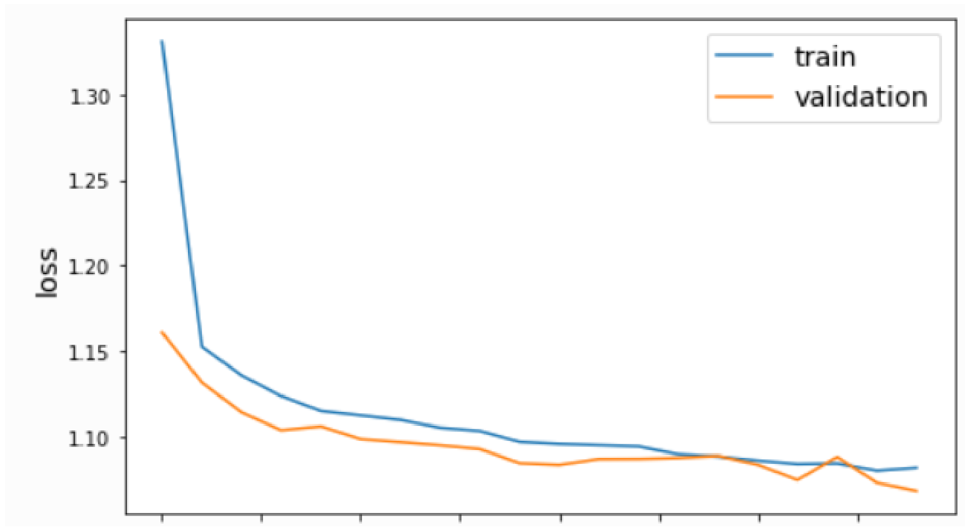


Figure 2

MLP Regressor Training Loss

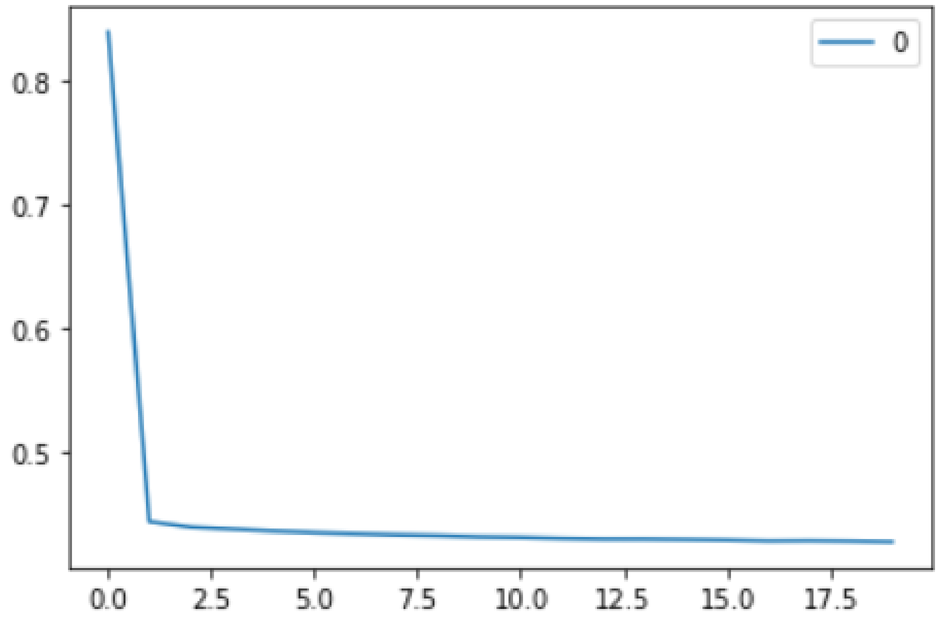


Figure 3

Root Mean Square Error

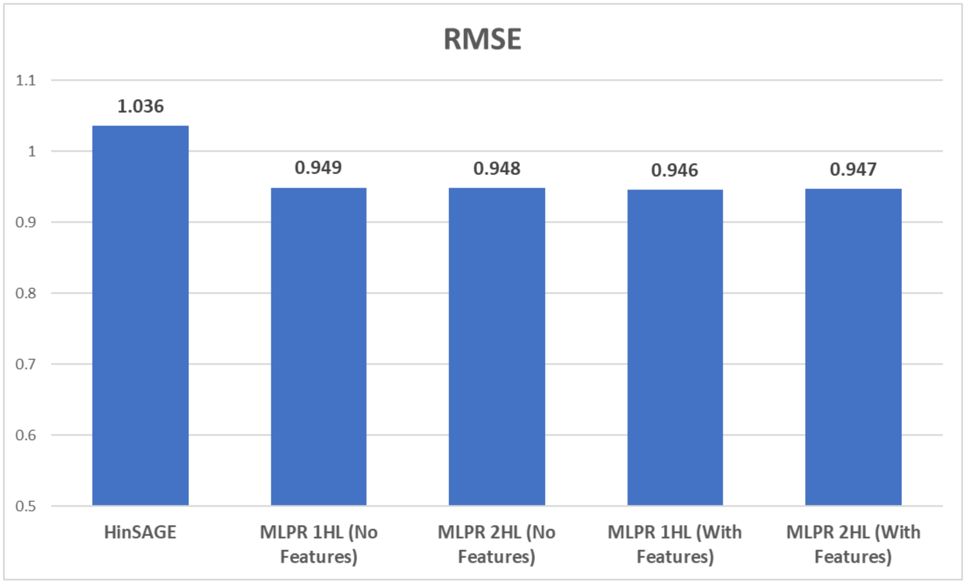


Figure 4

Mean Absolute Error

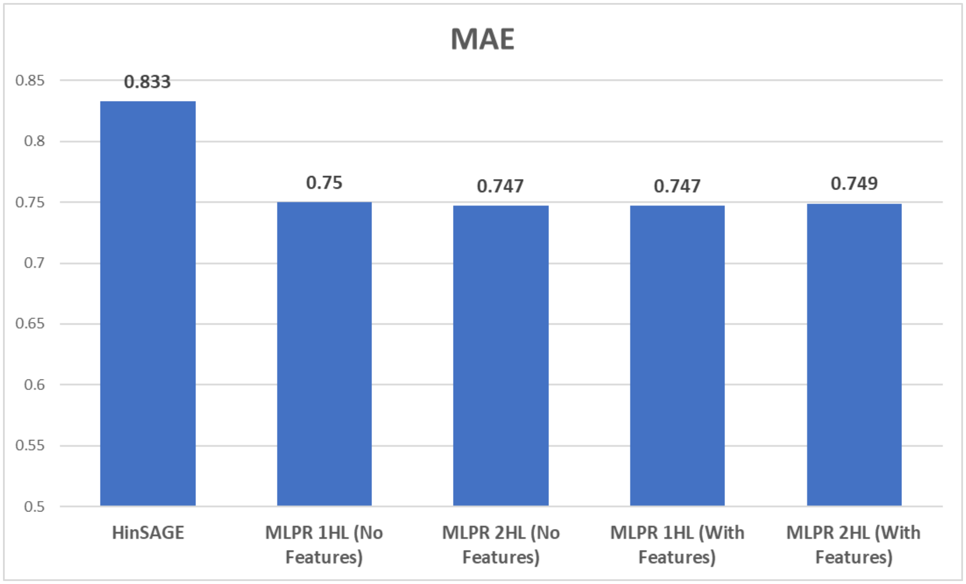


Figure 5

HinSAGE Histogram

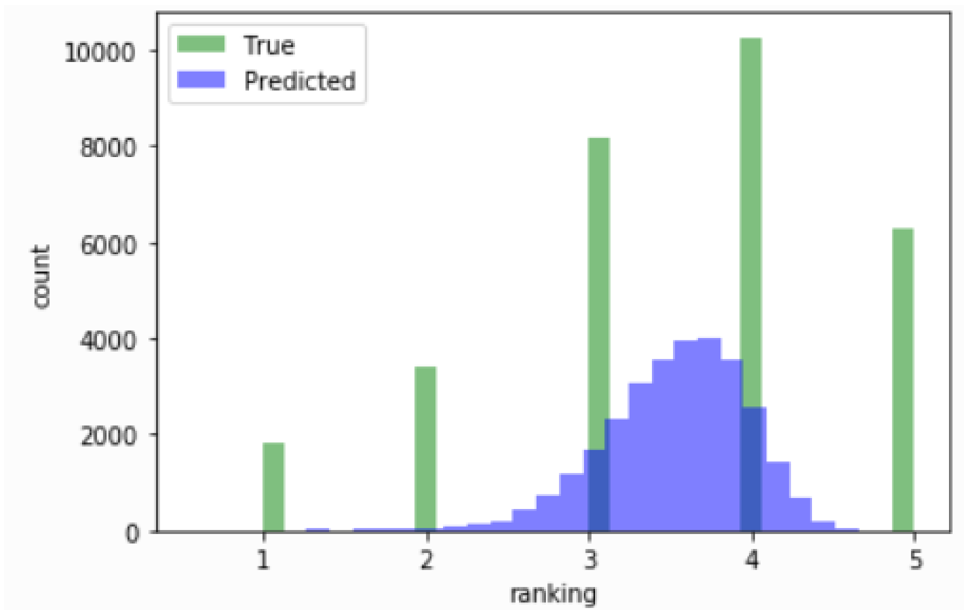
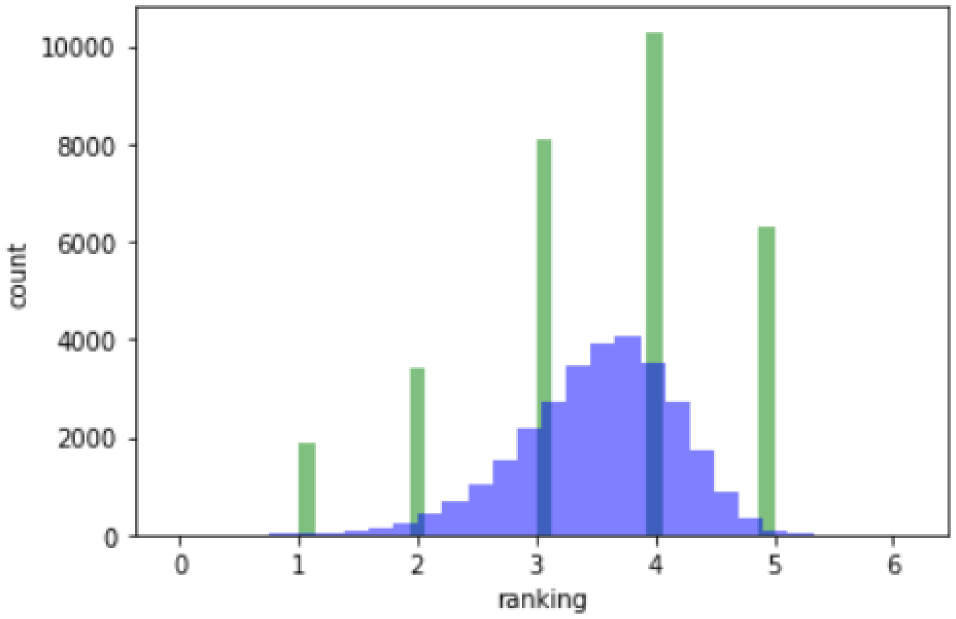


Figure 6

MLP Regressor



Appendix II

Multilayer Perceptron Regressor

The following illustrates the approach for one of the possible iterations of the MLPRegressor model, where the training partition and hidden layer depth are specified in the code section.

Section 1

```
# Import base libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn

%matplotlib inline


# Import task libraries

import sklearn as sk
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error


# Read in data

df = pd.read_csv('dataframe_modified.csv')


# View the data

df


# Describe the data

df.describe()
```

Section 2

Feature Engineering

```
# Convert numerical categories into strings

df_edit = df
df_edit['str_user'] = df_edit['userId']
df_edit['str_movie'] = df_edit['movieId']
df_edit['str_year'] = df_edit['year']
df_edit['str_user'] = df_edit['str_user'].astype(str)
df_edit['str_movie'] = df_edit['str_movie'].astype(str)
```

```
df_edit['str_year'] = df_edit['str_year'].astype(str)

# Check data types for converted features

df_edit['str_movie'].dtypes

# Generate one-hot encoded features

df_hot = pd.get_dummies(df_edit[['str_user','str_movie','gender','occupation','str_ye
ar']])

# View one hot encoded dataframe

df_hot

# Describe the one hot encoded dataframe

df_hot.describe()

# Get the names of one hot encoded dataframe

df_hot_columns = df_hot.columns.values.tolist()

df_features = pd.DataFrame(df_hot_columns,columns=['Features'])
```

Section 3

Scale the Numerical Features

```
# Scale the remaining features as necessary

df_scales = pd.concat([df['age']],axis=1)

df_scales

scaler = MinMaxScaler()

scaler.fit(df_scales)

df_scales['age'] = scaler.transform(df_scales)

df_scales.describe()
```

Section 4

Construct the Feature Matrix

```
# Index dataframe elements
```

```
df1_names = ['rating','event']

df2_names = ['age']

df3_names = ['gender_M']

df4_names = df_hot.iloc[:,1:943].columns.values

df5_names = df_hot.iloc[:,944:2625].columns.values

df6_names = df_hot.iloc[:,2628:2648].columns.values

df7_names = df_hot.iloc[:,2649:2719].columns.values

df8_names = ['action','adventure','animation','children','comedy','crime','documentar
y','drama','fantasy','film_noir','horror','musical','mystery','romance','sci_fi',
             'thriller','war','western']


# Create a feature dataframe

df_matrix = pd.concat([df[df1_names],df_hot[df4_names],df_hot[df5_names]],axis=1)


# View the feature dataframe

df_matrix


# Get the names of feature matrix

df_matrix_columns = df_matrix.columns.values.tolist()

df_matrix_features = pd.DataFrame(df_matrix_columns,columns=['Features'])


# Describe the full matrix

df_matrix.describe()
```

Section 5

Partition the Datasets

```
# Create the variable sets

y = df_matrix['rating']

x = df_matrix.iloc[:,2:]


# Parition the datasets
```

```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.7, random_state=0
)
```

Section 6

Build the Model

```
# Build the model

regr = MLPRegressor(hidden_layer_sizes=(32,32),
                    activation="identity",
                    solver="adam",
                    alpha=0.0001,
                    batch_size=200,
                    learning_rate_init=0.001,
                    max_iter=20,
                    random_state=0
                    ).fit(x_train, y_train)
```

Section 7

Evaluate the Model

```
# Plot the loss curve

pd.DataFrame(regr.loss_curve_).plot()


# Predict movie ratings

y_pred = regr.predict(x_val)


# Score the model

regr.score(x_val, y_val)


# Evaluate the model against the validation set

rmse = np.sqrt(mean_squared_error(y_val, y_pred))

mae = mean_absolute_error(y_val, y_pred)

print(rmse)
print(mae)


# Visualize the model performance

h_true = plt.hist(y_val, bins=30, facecolor="green", alpha=0.5)
```

```
h_pred = plt.hist(y_pred, bins=30, facecolor="blue", alpha=0.5)
plt.xlabel("ranking")
plt.ylabel("count")
#plt.legend("True", "Predicted")
plt.show()
```

Appendix III

HinSAGE

Section 1

Feature updates for homogeneous graphs

The feature update rule for homogeneous graphs is, for mean aggregator:

1. Aggregation of features from the neighbours of node vv:

   hkN(v)=1|N(v)|Dp[huk−1]hkN(v)=1|N(v)|Dp[huk−1]
2. Forward pass through layer kk:

   If `concat=True`:

   hvk=σ(concat[WkselfDp[hvk−1], WkneighhkN(v)]+bk)hvk=σ(concat[WkselfDp[hvk−1], WkneighhkN(v)]+bk)

   If ``` ``concat=False`` ```:

   hvk=σ(WkselfDp[hvk−1]+ WkneighhkN(v)+bk)hvk=σ(WkselfDp[hvk−1]+ WkneighhkN(v)+bk)

Where:

- hvkhvk is the output for node vv at layer kk
- WkselfWkself and WkneighWkneigh (both of size dk2×dk−1dk2×dk−1 if `concat=True`, or of size dk×dk−1dk×dk−1 if `concat=False`) are trainable parameters (shared for all nodes vv),
- bkbk is an optional bias,
- dkdk is node feature dimensionality at layer kk,
- σσ is the nonlinear activation,
- N(v)N(v) is the neighbourhood of node vv
- Dp[·]Dp[·]is a random dropout with probability pp applied to its argument vector.

The number of trainable parameters in layer kk for the mean aggregator is
dkdk−1+dkdkdk−1+dk if `concat=True`, or
2dkdk−1+dk2dkdk−1+dk if `concat=False`.

For the GCN aggregator, the feature update rule is:

1. Aggregation of features from the neighbours of node vv:

   hkN(v)=1|N(v)|+1(hvk−1+∑u∈N(v)huk−1)hkN(v)=1|N(v)|+1(hvk−1+∑u∈N(v)huk−1)
2. Forward pass through layer k:

hvk=σ(Wk·hkN(v)+bk)hvk=σ(Wk·hkN(v)+bk),

where WkWk (size dk×dk−1dk×dk−1) is a trainable weight matrix, shared between all nodes vv and other notation is as for the mean aggregator.

The number of trainable parameters in layer kk for the GCN aggregator is dkdk−1+dkdkdk−1+dk, i.e., this model has the same expressive power as the model with the mean aggregator and `concat=True`, or about half the expressive power of the model with the mean aggregator with `concat=False`.

Section 2

Feature updates for heterogeneous graphs (HINs)

The resulting feature update rules on heterogeneous graphs, for mean and GCN aggregators, are shown below (compare with the feature update rules for homogeneous graphs above).

HinSAGE with mean aggregator

1. Aggregation (mean) of features from the neighbours of node vv via edges of type rr:

   hkNr(v)=1|Nr(v)|∑u∈Nr(v)Dp[huk−1]hkNr(v)=1|Nr(v)|∑u∈Nr(v)Dp[huk−1]

2. Forward pass through layer k:

If `concat=Partial`:

hvk=σ(concat[Wktv,selfDp[hvk−1],Wkr,neighhkNr(v)]+bk)hvk=σ(concat[Wktv,selfDp[hvk−1],Wkr,neighhkNr(v)]+bk)

If `concat=Full`:

hvk=σ(concat[Wktv,selfDp[hvk−1],Wk1,neighhkN1(v),…,WkRe,neighhkNRe(v)]+bk)hvk=σ(concat[Wktv,selfDp[hvk−1],Wk1,neighhkN1(v),…,WkRe,neighhkNRe(v)]+bk)

If `concat=False`:

hvk=σ(Wktv,selfDp[hvk−1]+Wkr,neighhkNr(v)+bk)hvk=σ(Wktv,selfDp[hvk−1]+Wkr,neighhkNr(v)+bk)

Where:

- Wktv,selfWktv,self is the weight matrix for self-edges for node type tvtvand is of shape
  (dk2)×dk−1(dk2)×dk−1 if `concat=Partial` or
  dk×dk−1dk×dk−1 if `concat=False`, or
  dkRe+1×dk−1dkRe+1×dk−1if `concat=Full`.
- Wkr,neighWkr,neigh is the weight matrix for edges of type rr and is of shape
  dk2×dk−1(r)dk2×dk−1(r) if `concat=Partial` or
  dk2×dk−1(r)dk2×dk−1(r) if `concat=False`, or
  dkRe+1×dk−1(r)dkRe+1×dk−1(r) if `concat=Full`.
- rr denotes the edge type from node vv to node uu (rr is defined as unique tuple (tv,te,tu)(tv,te,tu)), where tvtv denotes type of node vv, and tete denotes the relation type.
- Nr(v)Nr(v) is a neighbourhood of node vv via edge type rr.
- dk−1(r)=dim(hkNr(v))dk−1(r)=dim(hkNr(v)) is the dimensionality of (k−1k−1)-th layer's features of node vv's neighbours via edge type rr.

The number of trainable parameters per layer kk for this model is

- If `concat=Partial`:

  Tv(dk2)dk−1+Re(dk2)dk−1+dk=Tv+Re2dkdk−1+dkTv(dk2)dk−1+Re(dk2)dk−1+dk=Tv+Re2dkdk−1+dk
- If `concat=False`:

  Tvdkdk−1+Redkdk−1+dk=(Tv+Re)dkdk−1+dkTvdkdk−1+Redkdk−1+dk=(Tv+Re)dkdk−1+dk
- If `concat=Full`

  TvdkRe+1dk−1+RedkRe+1dk−1+dk=dkdk−1Tv+ReRe+1+dkTvdkRe+1dk−1+RedkRe+1dk−1+dk=dkdk−1Tv+ReRe+1+dk.

assuming that the dimensionalities of all destination node features for all edge types rr are all equal, i.e. dk−1(r)=dk−1∀rdk−1(r)=dk−1∀r,

the number of all node types in the graph is TvTv, and the number of all edge types is ReRe.

HinSAGE with GCN aggregator

For GCN aggregator, the feature update rule is:

1. Aggregation of features from the neighbours of node vv, and node vv itself:

   hkNr(v)=1|Nr(v)|+1(hvk−1+∑u∈Nr(v)huk−1)hkNr(v)=1|Nr(v)|+1(hvk−1+∑u∈Nr(v)huk−1)
2. Forward pass through layer k:

hvk=σ(1ReWrk⋅hkNr(v)+bk)hvk=σ(1ReWrk⋅hkNr(v)+bk)

where WrkWrk are trainable weight matrices of size dk×dk−1dk×dk−1, one per edge type rr.

Note that in this model the dimensions of huk−1huk−1for u∈Nr(v)u∈Nr(v) for different edge types rr must be the same as the dimensionality of hvk−1hvk−1. This can be assumed to be true after the first layer as the bias vector bkbk in the update formula does not differ by node type.

However, if the dimensions of huk−1huk−1for u∈Nr(v)u∈Nr(v) differs from the size of hvk−1hvk−1 another weight matrix of size dk×dk−1(v)dk×dk−1(v) is required in front of the hvk−1hvk−1 term. This would be similar to the WkselfWkselfweight matrix of the mean aggregator.

The number of trainable parameters per layer kk for this model is:

Redkdk−1+dkRedkdk−1+dk

i.e., the model with GCN aggregator is less expressive (and hence less prone to overfitting in case of small datasets) than the model with mean aggregator.