Contextual features to generate personalized recommendations using Graph Neural Networks

Author: Pratik Aher

Problem Statement: Many recommendation algorithms have a tendency to exhibit popularity bias, where a small number of popular items are recommended to a large number of users. This can result in a lack of personalization in the recommendations provided. One solution to this problem is to incorporate rich contextual information from edge data and features from node data, which can enable more personalized recommendations for applications. To achieve this, the study will explore the use of Graph Neural Networks to generate personalized recommendations using contextual information.

---

1. Literature Review

Popularity bias is a drawback of many recommendation algorithms that favor a few popular items while ignoring the majority of less popular ones[1]. s. Typically, only a small percentage of items have a high volume of interactions, and this is referred to as the “head”. Most items are in the “long tail”, but they only make up a small percentage of interactions. Algorithms like PIXIE[2], that are commonly used tend to amplify the popularity of certain items because organizations seek to maximize their click-through rates, as people tend to engage with popular items more frequently.. There seems to be a need for users to discover niche items. Using the contextual information between users and an item interaction can prove helpful for a recommendation system.

Recent studies have shown how effective graph learning techniques are for recommendation tasks[3]. More recently, Uber Eats has achieved a 12% increase in clicks compared to the baseline model by incorporating the use of Graph Neural Networks (GNN)[4]. The Uber eats model did not use any other edge information except the number of times an item was ordered. There has been research in quantum chemistry to leverage edge contextual information in Graph Neural Networks to neural message passing for building graph classifiers[5].

Incorporating edge contextual information to train neural networks should help graph neural networks to come up with more nuanced recommendations.

2. Dataset

MovieLens is a widely-used dataset for recommendation research. It was established in 1997 for the purpose of collecting movie ratings data for research purposes. The MovieLens dataset has been critical for various research studies, including those related to personalized recommendation and social psychology. In this dataset, the most popular movies are blockbusters and classics. These movies are already well-known to most users, and recommendations of them may not offer a personalized experience or help users discover new, relevant movies.

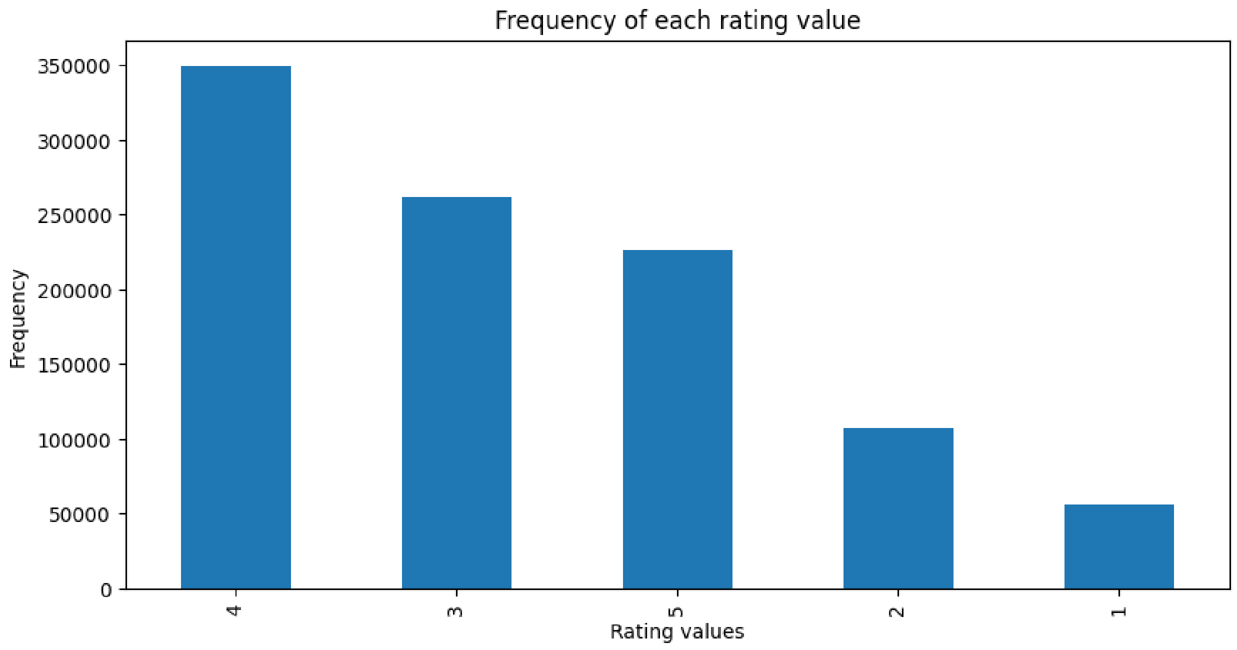
It contains 1000209 ratings and 6040 users across 3706 movies. Overall, there are three data frames that we use here : users dataframe, movie dataframe and ratings dataframe.

2.1. Exploratory data analysis

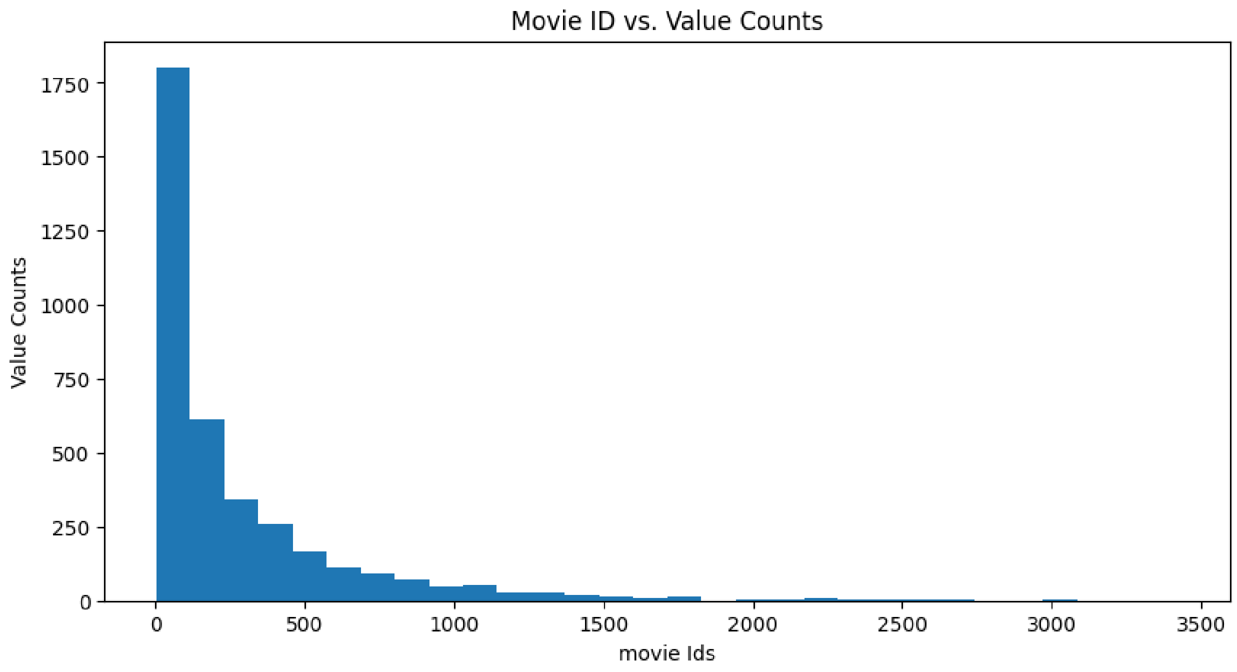
Here are the most important findings from the exploratory analysis of MovieLens dataset.

We can see that there are more highly rated movies than lowly rated movies.

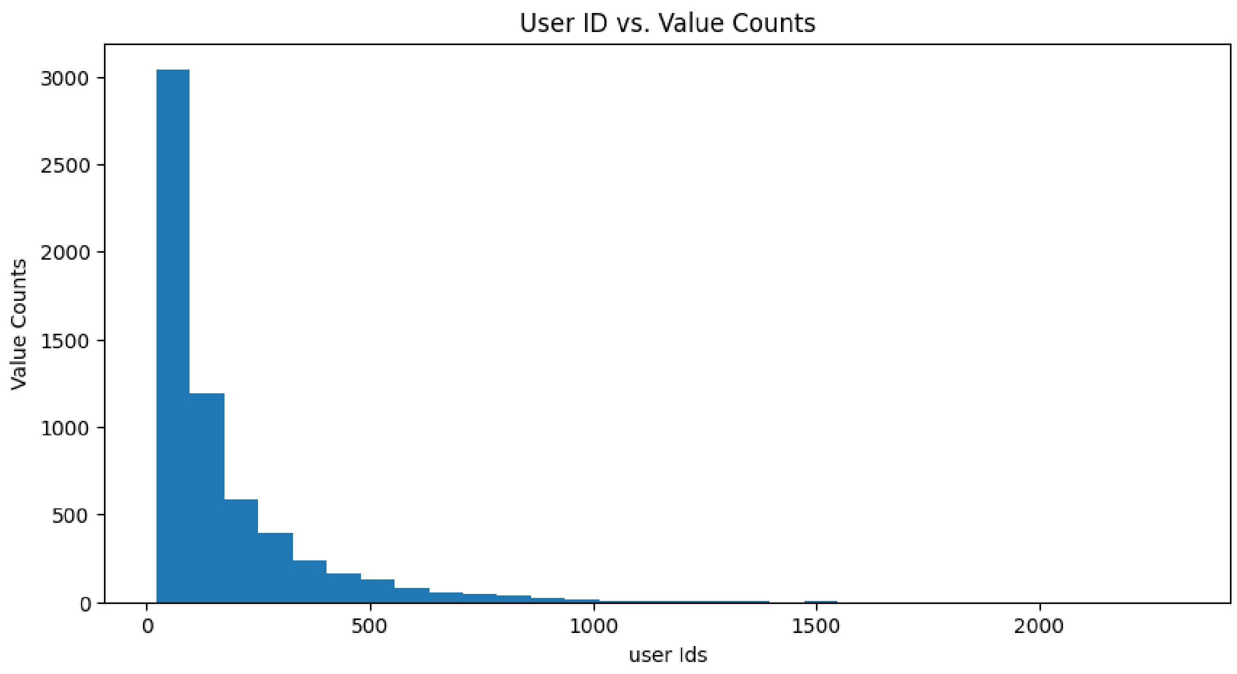
[1] Alejandro Bellogín, Pablo Castells ,Iván Cantador (2017). Statistical biases in Information Retrieval metrics for recommender systems. *Information Retrieval Journal* 20, 6 (2017), https://doi.org/10.1007/s10791-017-9312-z
[2] Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time : https://cs.stanford.edu/people/jure/pubs/pixie-www18.pdf
[3] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton and Jure Leskovec. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems.” *KDD* (2018)
[4] (2019, December 4). “Food Discovery with Uber Eats: Using Graph Learning to Power Recommendations.” *Uber Blog*. https://www.uber.com/blog/uber-eats-graph-learning/
[5] Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O. and Dahl, G.E. “Neural message passing for quantum chemistry.” *International conference on machine learning*, 1263-1272 (July 2017).



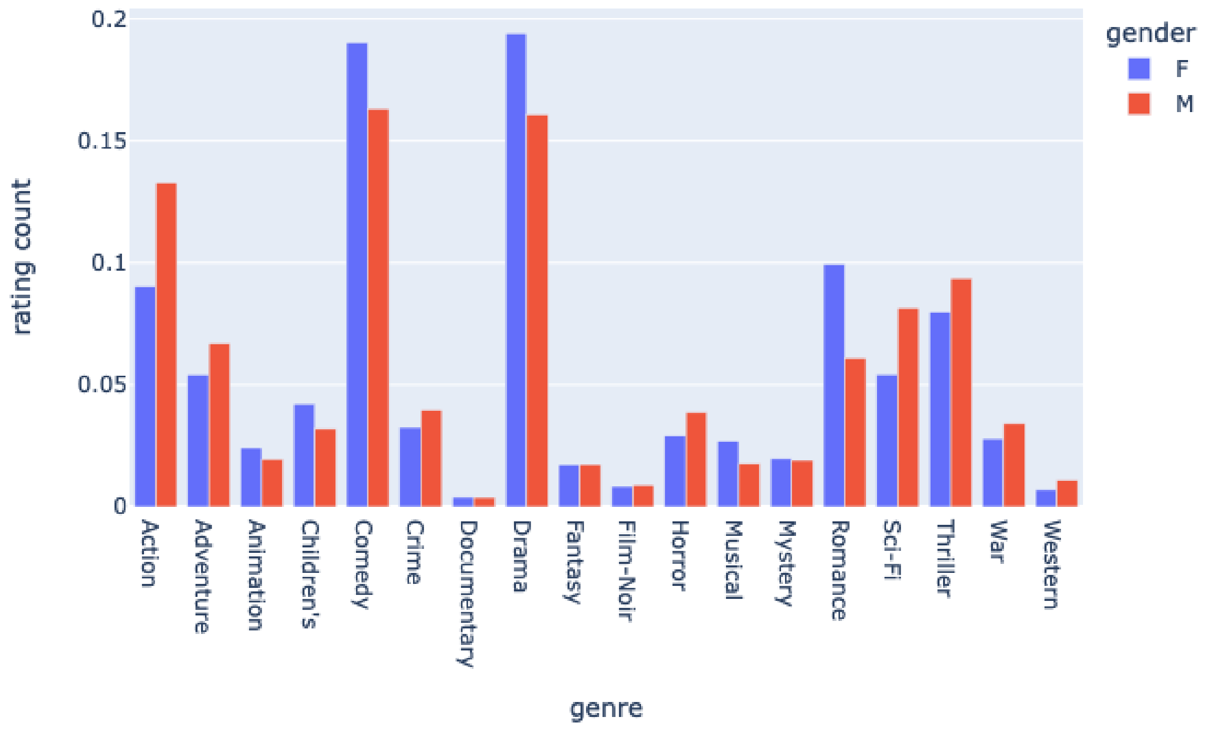
We can see that a small number of movies are requested a large number of times. We can see a long tail of movies, where a large number of movies are rated less number of times.



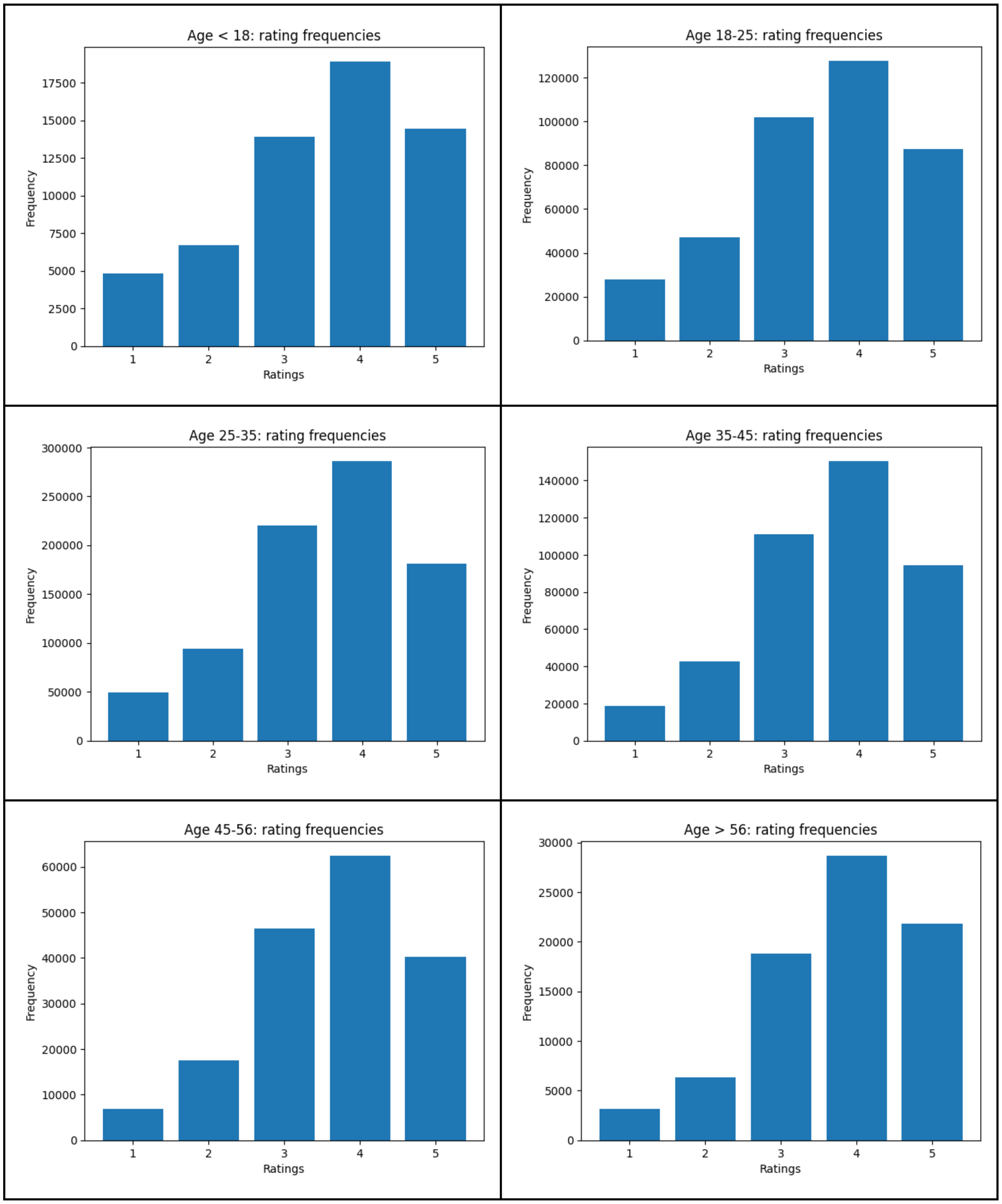
Similarly, we can see that a small number of users request a large number of times.



Again, we see some typical male/female differences in this dataset (men gave higher ratings to action movies than women and vice versa for romance movies).

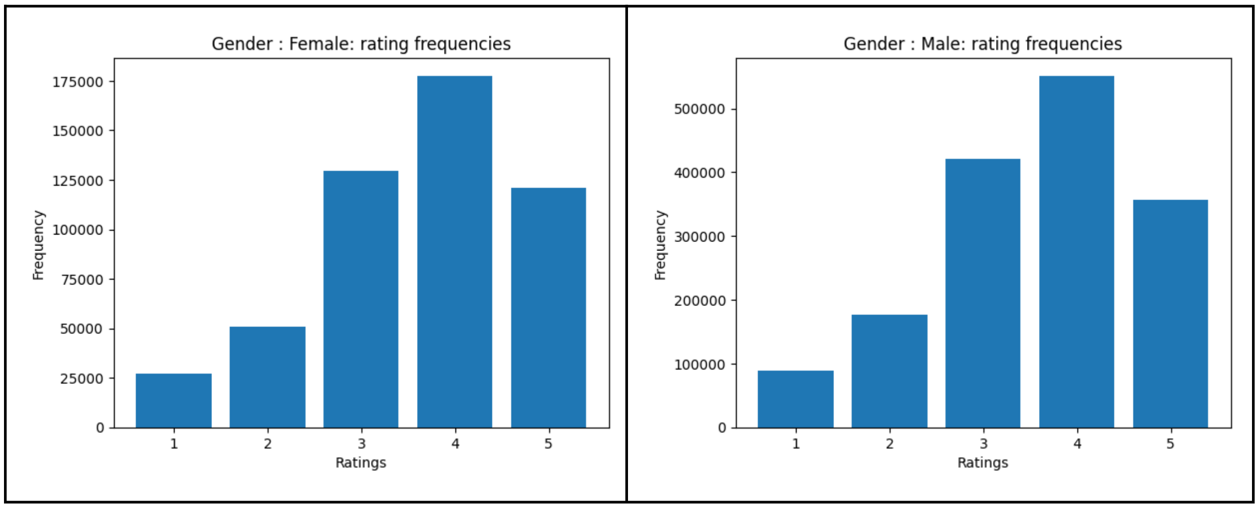


Below is the plot of age vs ratings :



Here, we can see that older people tend to rate things higher than younger people.

Below is the plot of gender vs ratings :



We can see that there aren't a lot of differences in the rating values according to gender. So it does not make sense to use this as a differentiating feature.

3. Graph Construction

In this study, we utilize the Deep Graph Library (DGL), a Python package for deep learning on graphs, to construct our graphs. DGL assigns a unique integer node ID to each node and a unique edge ID to each edge, based on the order in which it was added to the graph. The node and edge IDs both start from 0. All edges in DGL are directed, with an edge (u,v) indicating a direction from node u to node v. To create a DGLGraph, we can use the ***dgl.graph*** method and provide it with a set of edges. In our case, we use the ratings dataset, which contains edges representing the relationship between users and movies.

We construct a heterogeneous graph where the nodes are users and movies, and the edges between them are of two types: "rates" which go from user to movie and contain contextual information about the interaction between the user and movie, and "rev-rates" which go from movie to user. The nodes and edges of the ***DGLGraph*** can have several user-defined features to store properties specific to the graph.

3.1. Graph Cleaning

The DGL algorithm assigns nodes to the graph based on the maximum ID of the ratings IDs for both movies and nodes. This results in many isolated nodes in the graph. Additionally, there are many movies in the dataset that have not been rated, which would be represented as nodes in the graph without any connections to other nodes. To maintain the original IDs of the nodes, we assign each node an "original node ID." We then remove all of the isolated nodes from the graph.

3.2. Feature Selection

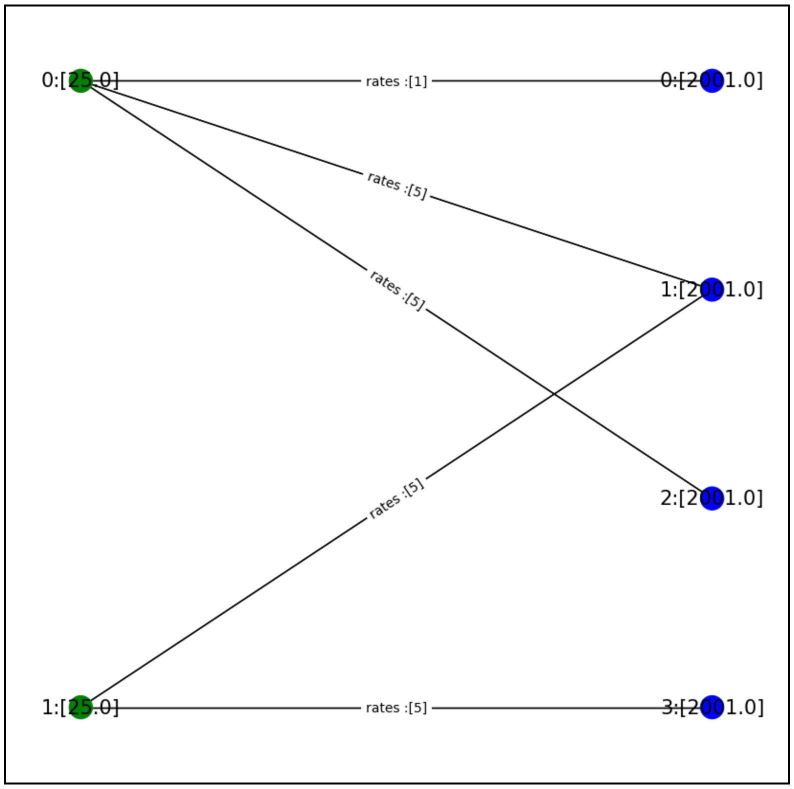
The user features we used in our model include age and gender. The movie features we used in our model include release year. Gender is encoded as a binary variable (0/1) to differentiate between male and female users. The timestamp of the user-movie rating interaction is converted to the corresponding date in Coordinated Universal Time (UTC). We then calculate the distance of the interaction date from the nearest holiday, and use this as an additional feature in our model. The "distance to holiday" feature can be represented using a one-hot encoding, where values closer to a specific holiday (e.g. Christmas) have a lower distance value compared to other holidays. This allows us to capture potential seasonal trends in movie viewing behavior.

3.3. Graph Testing

We wrote tests to fetch the node features from the graph and compare the values in the dataset. The test function fetched "original node ID"'s (node IDs before removing isolated nodes) and the features of nodes from the graph. It then compares the "original node IDs" to the movie IDs in the dataset.

3.4. Graph Plotting

DGL/NetworkX does not have a built-in plotting mechanism for plotting heterogeneous graphs and node/edge features. So, we wrote a functionality to take in a heterogeneous graph and the edge of interest and then plot the graph. This is how the plot looks like on a sample graph. The actual MovieLens dataset is too big to plot.



3.5. Negative Graph Construction

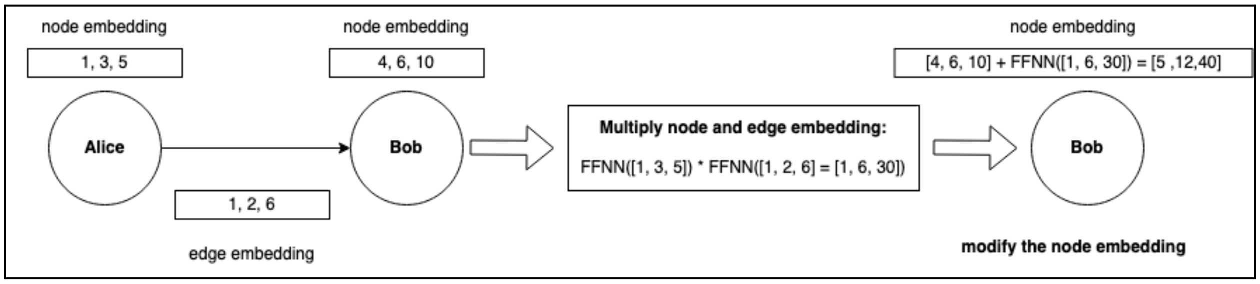
In the training phase, it is necessary to use a negative graph that includes edges that are not present in the actual graph. To create this negative graph, we constructed a graph that is similar to the original training graph but includes three times as many edges, some of which are not present in the original graph. This is done to help the neural network architecture learn to distinguish between correct and incorrect edges.

During training, the neural network produces two scores: a positive score indicating how well it performed on the positive graph (i.e., the original graph with the correct edges), and a negative score indicating how well it performed on the negative graph (i.e., the artificially constructed graph with incorrect edges). Ideally, we want the positive score to be higher and the negative score to be lower, as this indicates that the neural network is effective at predicting the correct edges in the graph.

3.6. Graph : Dataset Split

For the link prediction task, we want to randomly split *edges* into train, valid, and test data. We take all the edges from the DGL graph and shuffle them. Then, we take 70% of edges for the train graph, 20% for the test graph and 10% for the validation graph. We build subgraphs from these split edges and call these graphs as train graph, test graph, validation graph. Note that these graphs have the same number of nodes as the original graph. It is just the edges that are split.

4. Neural Network Architecture


node embedding
1, 3, 5
Alice
1, 2, 6
edge embedding
node embedding
4, 6, 10
Bob
Multiply node and edge embedding:
FFNN([1, 3, 5]) * FFNN([1, 2, 6] = [1, 6, 30])
node embedding
[4, 6, 10] + FFNN([1, 6, 30]) = [5 ,12,40]
Bob
modify the node embedding

Message passing paradigm :

The message passing paradigm is a widely-used approach for learning node representations in graph-structured data using neural networks. In this paradigm, each node in the graph is associated with a neural network that processes and updates the node's representation based on the representations of its neighboring nodes and edges. This process is repeated for multiple iterations, allowing the node representations to incorporate information from the entire graph.

The message passing on heterographs can be split into two parts:

1. Message computation and aggregation for each relation r.

2. Reduction that merges the aggregation results from all relations for each node type.

The key idea behind the message passing paradigm is that the structure of the graph can be used to propagate information between nodes. In a graph neural network, each node neural network receives information from its neighboring nodes and edges, and uses it to update its own representation. By iteratively updating the node representations in this way, the network can learn complex, abstract representations of the graph structure that are useful for a variety of tasks, such as node classification or link prediction.

In this specific case, we use the message passing paradigm to include edge features along with node features. The purpose is to use contextual information embedded in the edge features to modify the node features so that it contains the changes due to this information.

Node Function:

We pass the node information through a feed forward neural network. The feed forward neural network consists of a linear layer that converts the node features to an input embedding of size 8.

Edge Function :

We pass the edge information through a feed forward neural network as well. This feed forward neural network consists of : Two linear layers with 128 hidden units separated by ReLU activation function with dropout on the last layer.

4.1. Loss Function:

The cosine predictor for the node_features returns the distance of the node embeddings to the positive and negative graph. So, we get a list of two scores, positive score (distance vector for positive graph) and negative score (distance vector for negative graph).

The hinge loss is calculated as the maximum of 0 and the difference between the positive score and the negative score, plus a margin constant. This means that if the difference between the positive score and the negative score is greater than the margin constant, the hinge loss will be zero, indicating that the model has made a correct prediction. If the difference is less than the margin constant, the hinge loss will be equal to the difference minus the margin constant, indicating that the model has made an incorrect prediction.

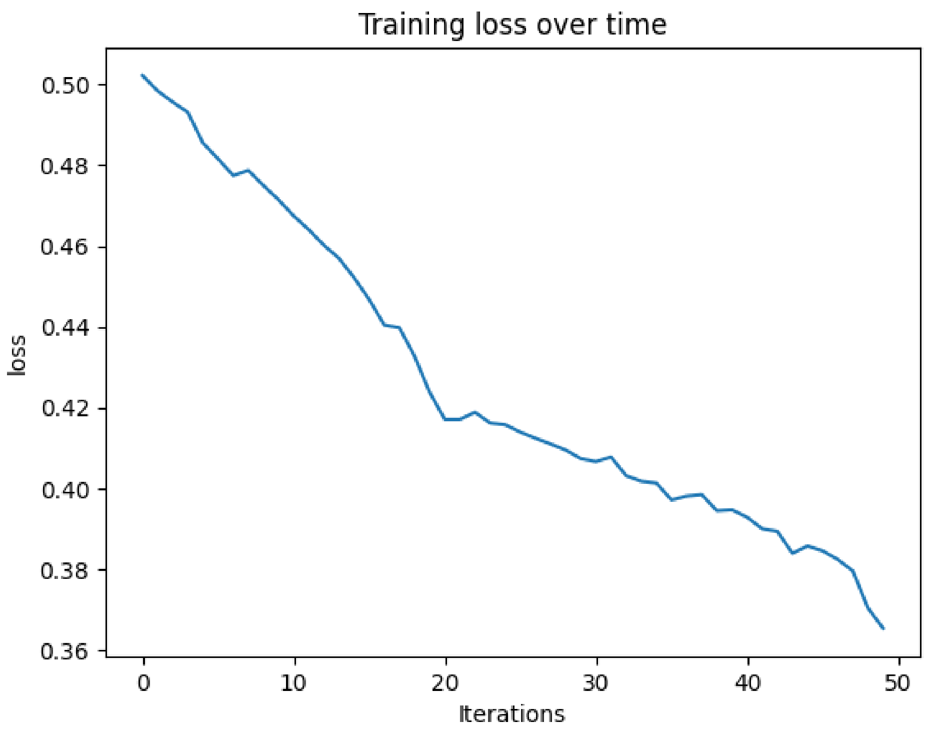
We calculate the score across all edge types and then take the mean of the scores as the total loss. We then backpropagate the loss to change the weights of the neural network.

4.2. Fetch recommendations from model:

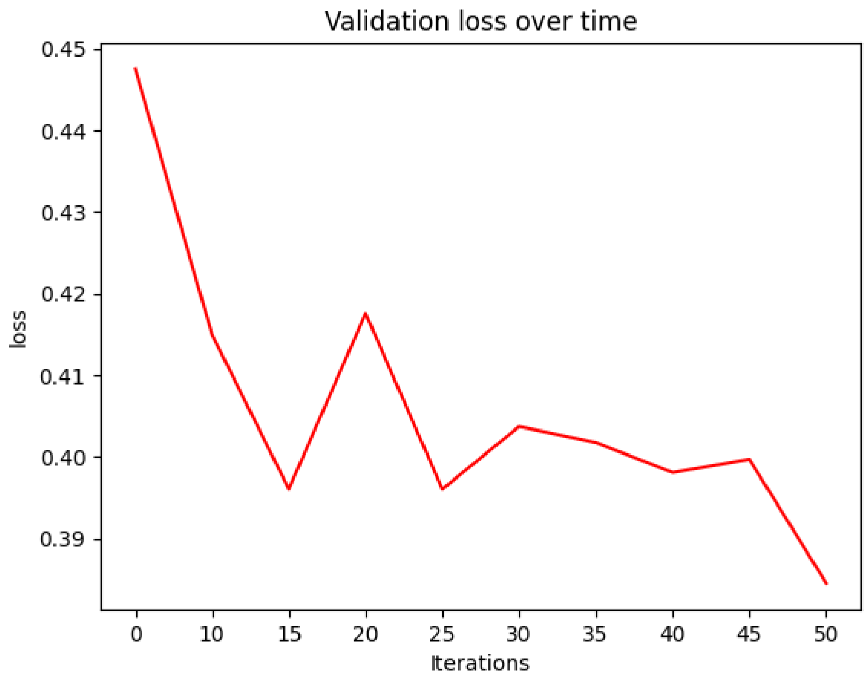
Once we have obtained the final user and item embeddings, we can calculate the distance between each user embedding and each item embedding. This can be done by repeating the user embedding a number of times equal to the number of movies, and then calculating the cosine distance between the repeated user embedding and the movie embeddings. From the movie recommendations, we remove movies that are already rated by the user. We can then sort the items according to their distance from the user embedding. We then select the top N items with the shortest distance as the recommendations for that user. These N values are the movie recommendations for every user.

5. Results:

Below is the plot for "**Training Loss**" :

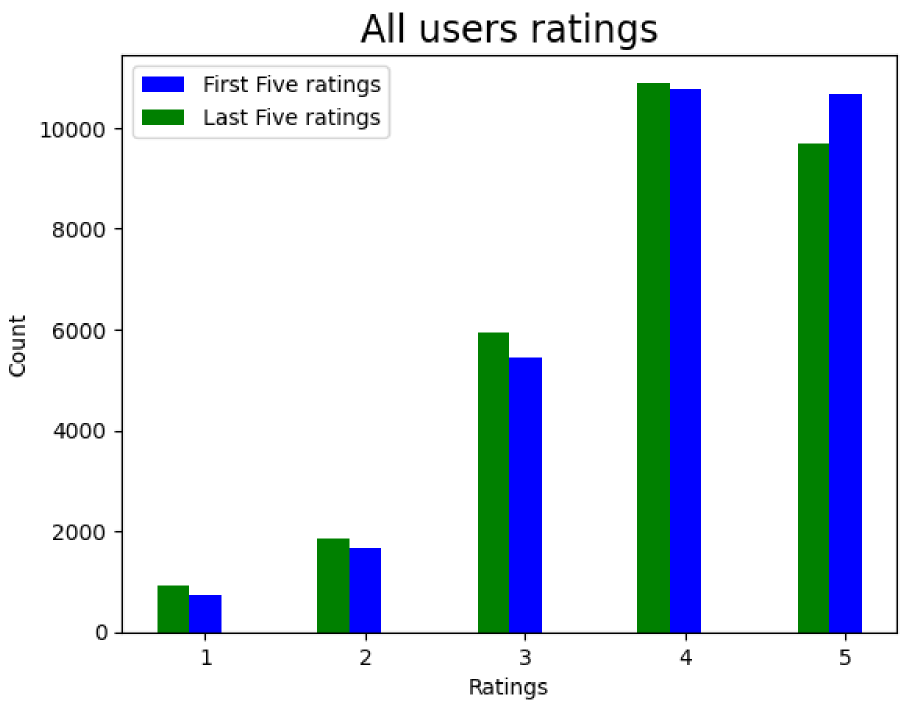


Below is the plot for "**Validation Loss**" :



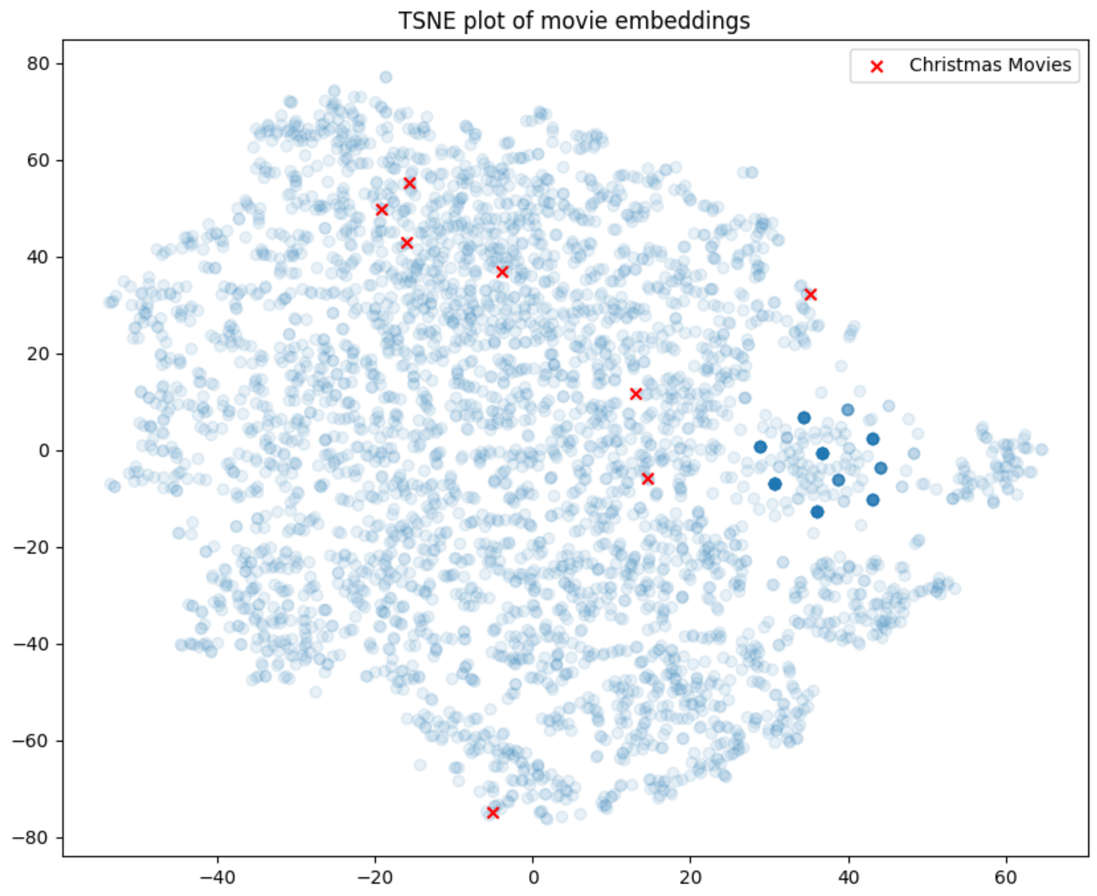
5.1. Evaluation

We tried to look at the ratings of the first five (closest to the user embedding) and last five (furthest to the user embedding) correctly predicted movies.



Our analysis of the model's performance showed that the first five correctly predicted movies had higher ratings than the last five correctly predicted movies for five-star ratings, while the ratings for four-star predictions were similar across both sets. Additionally, we observed that one- or two-star ratings were more prevalent among the last five correctly predicted movies, indicating that the model is able to recommend higher-rated movies more consistently.

To visualize the trained movie embeddings, we employed T-SNE (t-distributed stochastic neighbor embedding), a technique that maps high-dimensional data onto a lower-dimensional space while preserving as much of the original data's structure as possible. This allowed us to more easily analyze the relationships between the movies in the embedding space.



In the plot above, we plot Christmas movie embeddings. We can see that a few of the Christmas movies are really close to each other. Moreover, we can also see that some movie groups are clustered closer to each other.

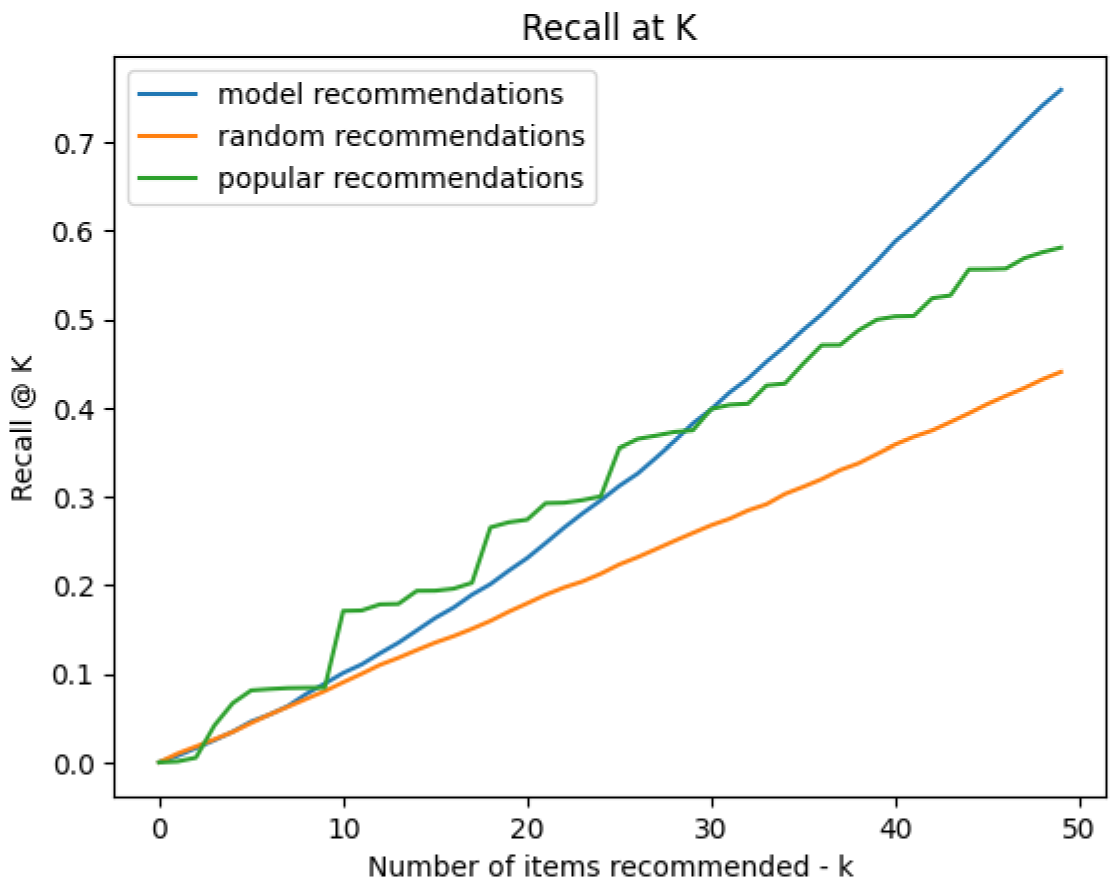
Three different recommender systems are tested and compared.

1. Random recommender (recommends random movies to each user)
2. Popularity recommender (recommends the top most popular movies to each user)
3. Recommendations from the model (Graph Neural Networks)

We calculate recall for each model when k items are being recommended to every user. Here recall for a specific user is defined as :

R = (# of top k recommendations that are correctly recommended)/(# of all relevant items)

We the average this recall value across al



Conclusion :

Limitations :

Due to computational limitations, we were only able to test output embedding sizes of 8 dimensions and hidden layer units of 128 units. It would be interesting to investigate the effect of larger output embedding sizes, such as 16 dimensions, on the model's performance. Similarly, we were unable to test the impact of different edge network sizes on the model's performance due to computational constraints.

In terms of message passing functions, we used u_mul_e, which multiplies node features with edge features. However, it would be interesting to explore the use of other message passing functions, such as u_div_e and u_add_r, to see if they yield different results. Additionally, we utilized mean pooling for message passing, but it would be worth investigating the use of alternative pooling techniques, such as LSTM or RNNs, to see if they result in improved performance.

Due to time constraints, we were unable to evaluate the model recommendations using a variety of metrics. It would also be interesting to evaluate the model on a different dataset that includes more contextual features to see if it performs differently in that context. Additionally, the timestamp feature of the dataset we used (Movielens) does not necessarily indicate that the user watched the movie immediately after rating it, so our assumption that the rating was made at the same time as the viewing may not be accurate.

Discussion :

In this project, we utilized graph neural networks (GNNs) to generate personalized recommendations by incorporating contextual edge features and node features. To facilitate our analysis, we utilized the MovieLens dataset and conducted extensive exploratory data analysis to identify various patterns and features within the dataset. Throughout the course of the project, we developed several techniques for constructing, cleaning, splitting, testing, and visualizing the graph.

We introduced a novel GNN architecture that utilizes message passing on a heterogeneous graph, a technique that has previously only been applied to quantum chemistry on homogenous graphs. Upon evaluating the results, we observed that our model recommended a higher proportion of movies that would receive a five-star rating from users at the top of the list, compared to the bottom. Additionally, we identified clusters of movie embeddings in the embedding space. We also found that the average recall of our recommendations outperformed that of other popular recommendation methods.

Overall, our work highlights the potential of GNNs for personalized recommendation systems and introduces a novel application of message passing on heterogeneous graphs.