Final Report for Assignment 2

By Nagim Isyanbaev B21-DS-02

Introduction

The primary objective of the assignment was to design a model with the ability to suggest movies to users by considering their demographic details and their previously enjoyed movies. This challenge is closely linked to content-based filtering and has many solutions in the research domain. In this assignment, my main focus was on harnessing the power of Graph Neural Networks (GNN) to solve this problem.

Data analysis

During the data analysis, I made several interesting observations. First of all, the MovieLens 100k dataset is segmented into six edge-disjoint subsets *(1, 2, 3, 4, 5, a, b)*. Moreover, both the training and testing sets within the one subset are also edge-disjoint. Such dataset structure is valuable as it allows us to evaluate our model across different subsets, leading to more precise metrics.

The next observation is that the majority of users fall within the 20 to 30 age range (Figure 1.). Additionally, our dataset exhibits a gender skew, with more male users than female (Figure 2.). Furthermore, I've found that the major genre among the movies in our dataset is drama (Figure 3.). Lastly, it's worth noting that the most popular rating among users is 4 starts (Figure 4.).

In terms of preprocessing, I decided to remove all features except genre from movies and one hot encode such features as gender, occupation and genre.

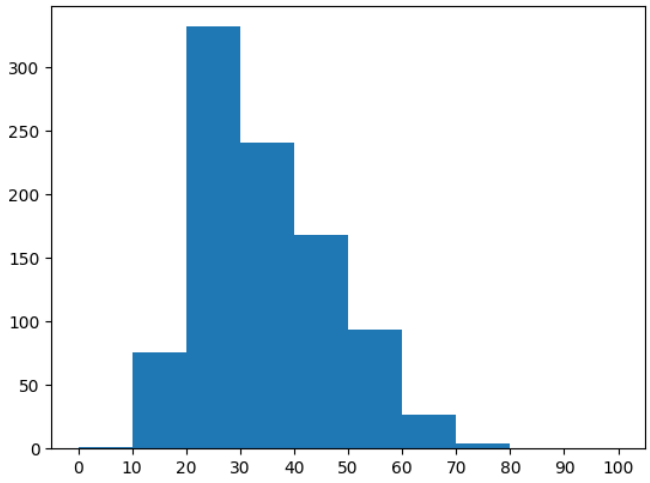


Figure 1.

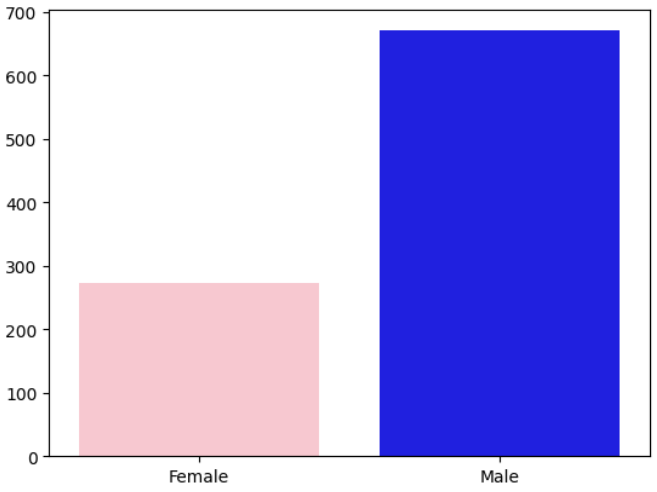


Figure 2.

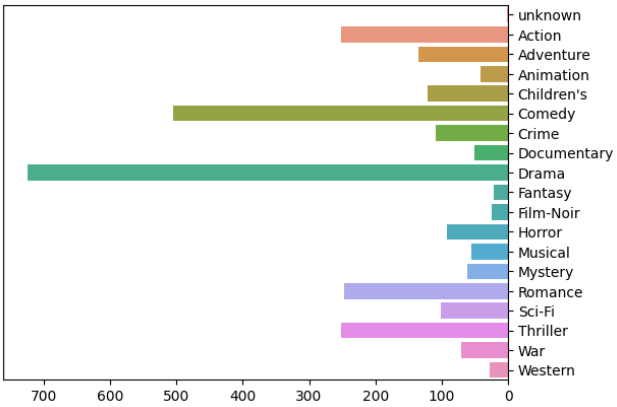


Figure 3.

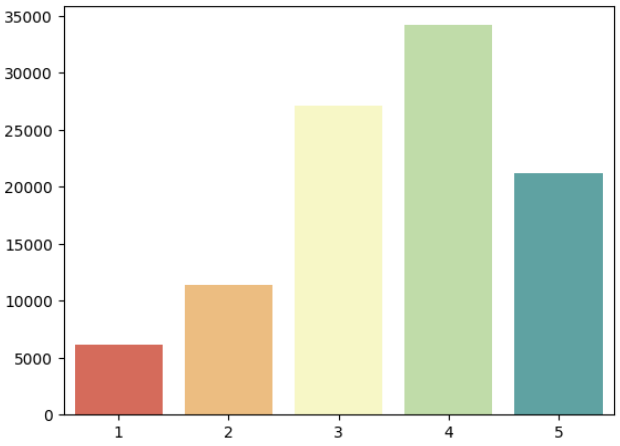


Figure 4.

To be able to use GNN on the dataset, it was transformed into a Bipartite graph, where one set of nodes are users and the other one are movies (Figure 5.).

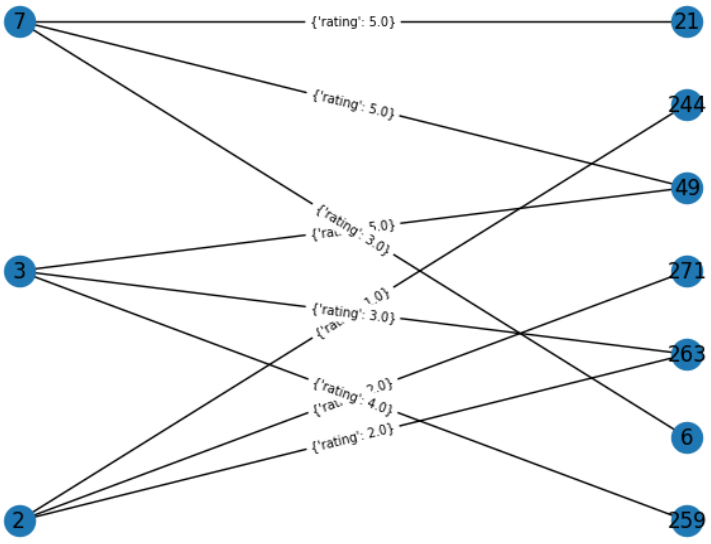


Figure 5.

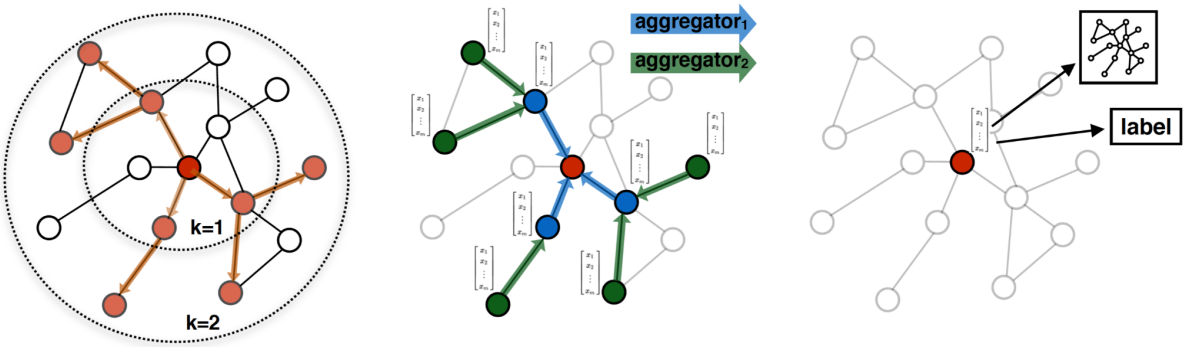
Model implementation

I designed the following pipeline for the recommendation system:

1. Build node embeddings: Create node embeddings in a high-dimensional space using the GraphSAGE model.
2. Predict movie rating for users: Estimate movie ratings for users based on cosine similarity between movie and user embedding vectors.
3. Recommendation: Arrange movies according to predicted ratings and provide recommendations.

GraphSAGE

GraphSAGE, a type of Graph Convolutional Networks, operates through message passing in a graph. The message passing algorithm involves collecting features from neighbors, aggregating them, multiplying the result by a weight matrix, and then updating the current node embedding. A key distinction of GraphSAGE from the default GCN is its feature collection solely from a subsample of neighbors, aiming to reduce execution time, especially on large graphs.



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

The chosen aggregation method is averaging. The formula for message passing is as follows:

$$x_j' = W_1 x_i + W_2 mean_{j \in N(i)}(x_j)$$

Here x_j' represent the update node i feature vector, W_1,W_2 are weight matrices, x_j feature vector of node j, N(i) is the sampled neighborhood of node i

Node embeddings

Node embeddings are N-dimensional vectors in a designated space. Pairs of vectors for users and movies highly rated by those users are close in this space. The distance between vectors is determined by the angle between them, indicating cosine similarity. The rating of movie A for user B is calculated as follows: **RATING(A,B)=(cosine_similarity(A,B)+1)×2.5**

Architecture

Architecture consists of two SAGE convolution layers with 128 embedding dimensions. The link rating prediction relies on cosine similarity, as illustrated. Afterward, recommendations are generated based on the predicted ratings, with movies sorted in descending order of rating.

Model Advantages and Disadvantages

Advantages:

1. Model is simple to implement and fast to train.
2. Obtained embeddings can be used to search similar users or movies.

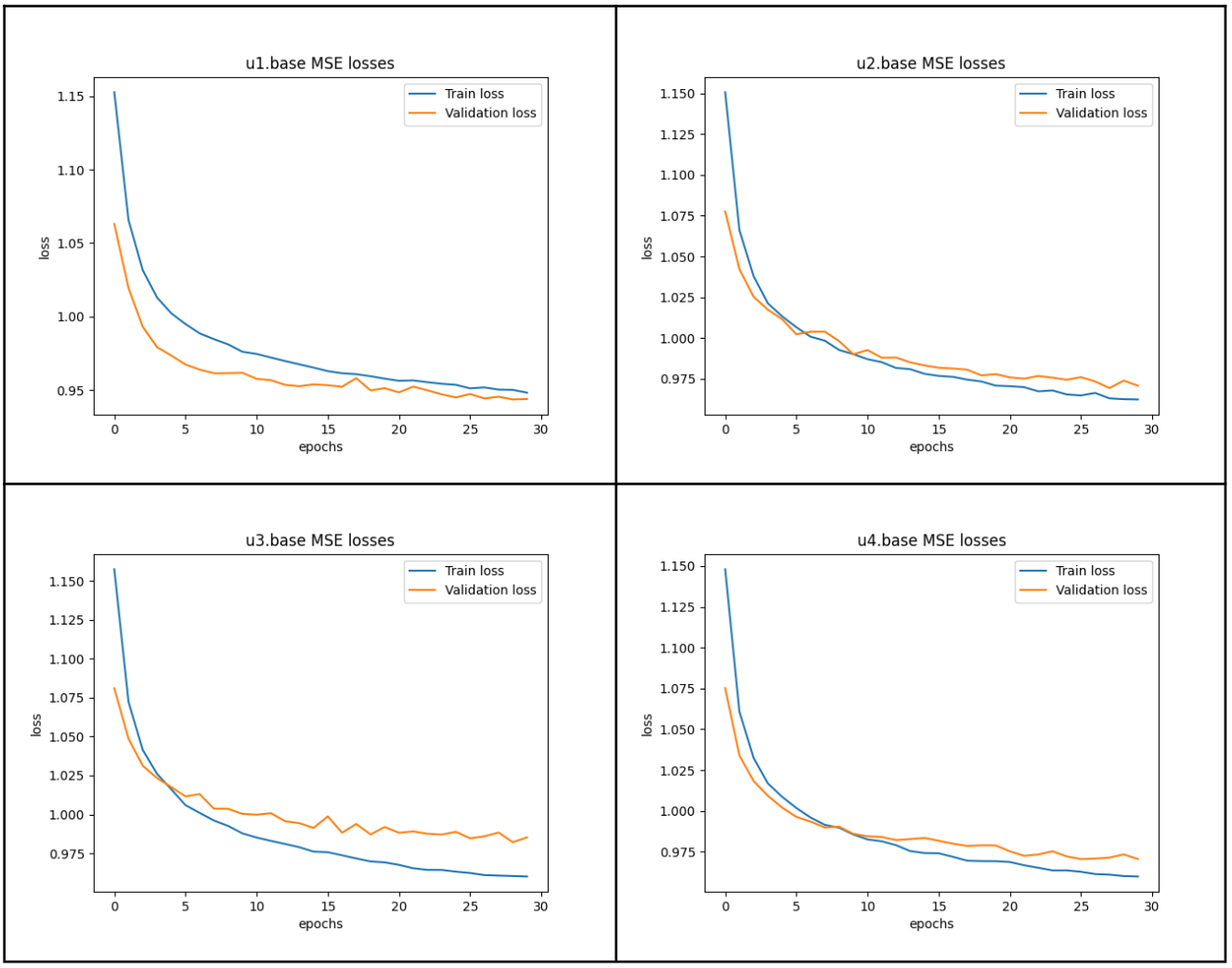
Disadvantages:

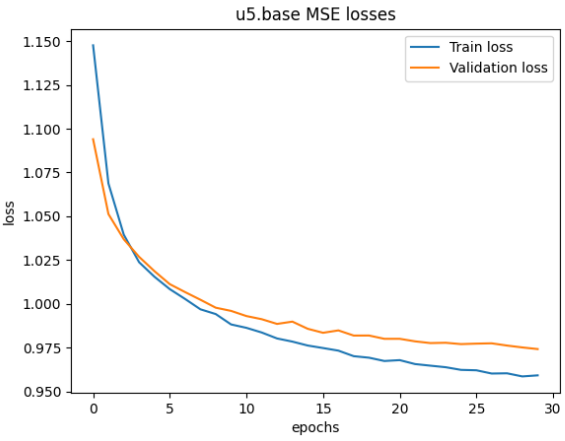
1. High RMSE, in comparison to other methods my solution has an RMSE value about 1.18 which is not so perfect.
2. Can perform very poorly if there are a lot of edges on a preprocess bipartite graph.

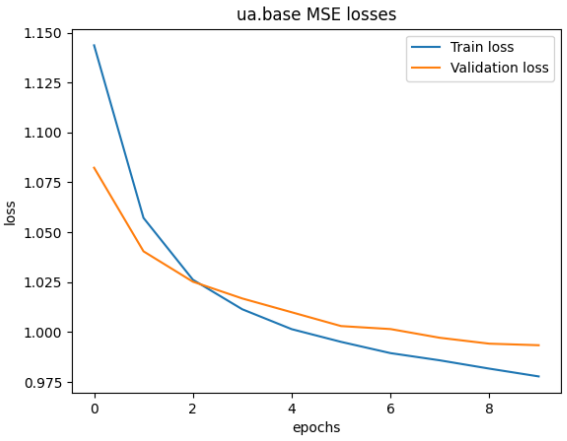
Training Process

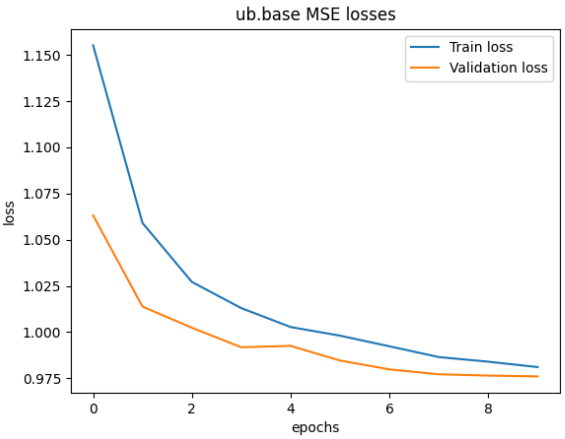
To train the model, I utilized the training sets from the MovieLens 100k dataset, specifically (u1.base, u2.base, u3.base, u4.base, u5.base, ua.base, ub.base). For each train set, I divided it into smaller sets, creating both train and validation sets. Subsequently, I randomly selected edges from the train set using the LinkNeighborhoodLoader from the PyG library, training the model on mini-batches.

The loss plots for each train set (Mean Squared Error):







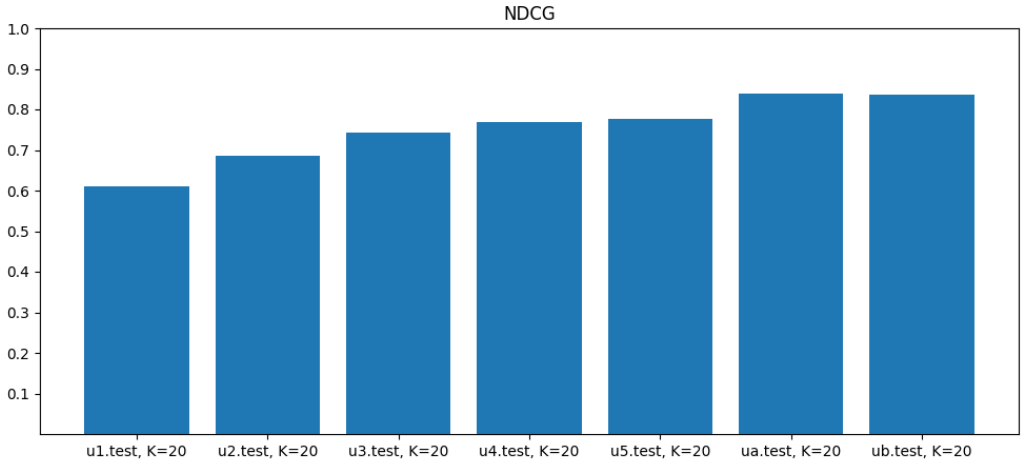


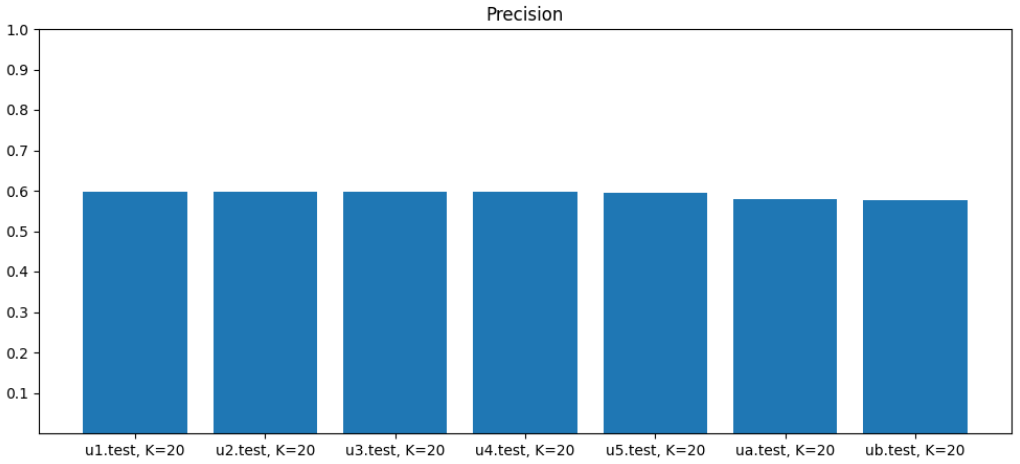
Evaluation

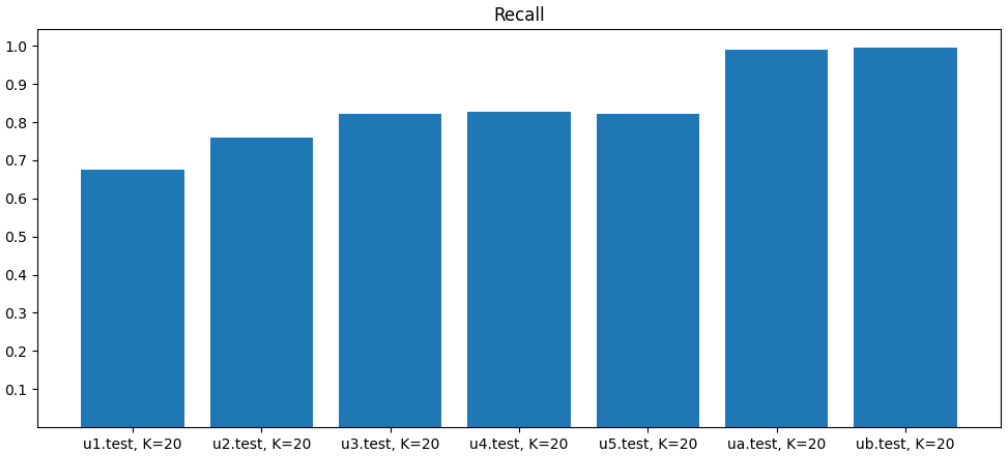
For evaluation, I employed the following metrics: Precision@20, Recall@20, NDCG@20, and RMSE. The test sets utilized were extracted from the MovieLens 100k dataset, specifically (u1.test, u2.test, u3.test, u4.test, u5.test, ua.test, ub.test). To enhance the clarity of metrics, I excluded all edges not present in the test sets, as they significantly negatively impact metric values.

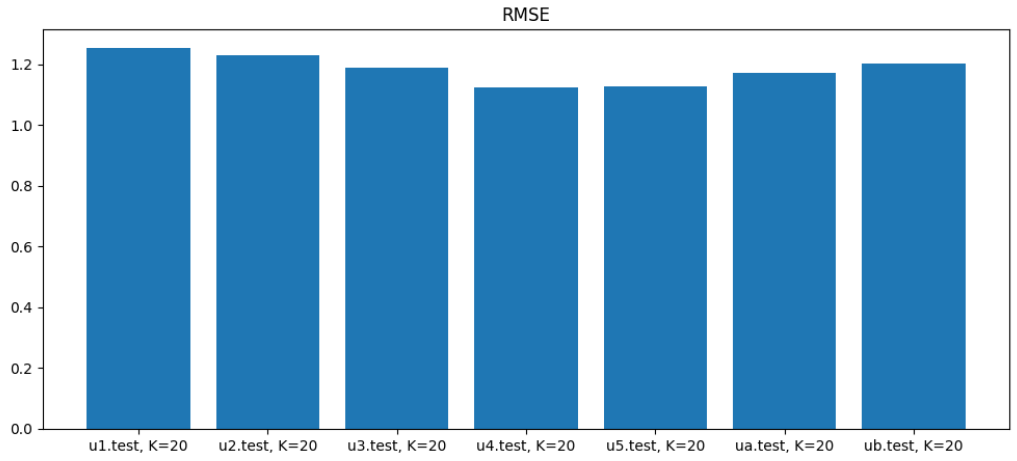
Results

Metric values:
Average RMSE: 1.1856
Average Precision@20: 0.5919
Average Recall@20: 0.8411
Average NDCG@20: 0.7518


NDCG
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
u1.test, K=20
u2.test, K=20
u3.test, K=20
u4.test, K=20
u5.test, K=20
ua.test, K=20
ub.test, K=20


Precision
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
u1.test, K=20
u2.test, K=20
u3.test, K=20
u4.test, K=20
u5.test, K=20
ua.test, K=20
ub.test, K=20


Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
u1.test, K=20
u2.test, K=20
u3.test, K=20
u4.test, K=20
u5.test, K=20
ua.test, K=20
ub.test, K=20


RMSE
1.2
1.0
0.8
0.6
0.4
0.2
0.0
u1.test, K=20
u2.test, K=20
u3.test, K=20
u4.test, K=20
u5.test, K=20
ua.test, K=20
ub.test, K=20