Project Report
on

Developing a Movie Recommendation System Using Graph Neural Network

A Major Project Report

Submitted in fulfillment of
the requirements for the award of the degree of

Master of Computer Applications 2024

Submitted by

Khusbu Pradhan
Roll No: 22MCA006

Monisha Devi
Roll No: 22MCA010

Tenzin Deckyi Namgyal
Roll No: 22MCA015

Under the guidance of
Dr. Swarup Roy
Professor
Department of Computer Applications
Sikkim University,Gangtok-737102, India
July, 2024

Acknowledgements

It was our privilege and honour to work under the supervision of Dr. Swarup Roy, Professor,Department of Computer Applications, Sikkim University, Gangtok, India. We would like to express our sincere gratitude to our supervisor Dr. Swarup Roy for his excellent guidance, care, patience, and for providing us with an excellent atmosphere for doing research.

We would also like to extend our heartfelt appreciation to Dr. Mohan Pratap Pradhan, Head of Department of Computer Applications, Sikkim University for his unwavering support and encouragement.

We would like to thank all faculty members at the Department of Computer Applications, Sikkim University, Gangtok, India, for their valuable suggestions during our research work. I would like to thank the staffs of the Department of Department of Computer Applications, Sikkim University, Gangtok, India, who were always willing to help and give their best support in our project work journey.

We deeply acknowledge the love, cooperation, and the moral support extended by our parents, friends, relatives, and colleagues right from the beginning of project work.

Finally, We must thank the Almighty for blessing us in this journey.

Khusbu Pradhan	Monisha Devi	Tenzin Deckyi Namgyal
Roll No: 22MCA006	Roll No: 22MCA010	Roll No: 22MCA015

Declaration

I **Khusbu Pradhan**, Roll No. **22MCA006**, a registered candidate for Master of Computer Applications under Department of Computer Applications (MCA) of Sikkim University, Gangtok, India, declare that this is my own original work and does not contain material for which the copyright belongs to a third party and that it has not been presented and will not be presented to any other University/Institute for a similar or any other Degree award.

I further confirm that for all third-party copyright material in my project report (including any electronic attachments), I have "blanked out" third party material from the copies of the thesis fully referenced the deleted materials and where possible, provided links (url) to electronic sources of the material.

I hereby transfer exclusive copyright for this project report to Sikkim University. The following rights are resewed by the author:

a) The right to use, free of charge, all or part of this article in future work of their own, such as books and lectures, giving reference to the original place of publication and copyright holding.

b) The right to reproduce the article or thesis for their own purpose provided the copies are not offered for sale.

Khusbu Pradhan

Roll No: 22MCA006

Date:

Declaration

I **Monisha Devi**, Roll No. **22MCA010**, a registered candidate for Master of Computer Applications under Department of Computer Applications (MCA) of Sikkim University, Gangtok, India, declare that this is my own original work and does not contain material for which the copyright belongs to a third parry and that it has not been presented and will not be presented to any other University/Institute for a similar or any other Degree award.

I further confirm that for all third-party copyright material in my project report (including any electronic attachments), I have "blanked out" third party material from the copies of the thesis fully referenced the deleted materials and where possible, provided links (url) to electronic sources of the material.

I hereby transfer exclusive copyright for this project report to Sikkim University. The following rights are resewed by the author:

a) The right to use, free of charge, all or part of this article in future work of their own, such as books and lectures, giving reference to the original place of publication and copyright holding.

b) The right to reproduce the article or thesis for their own purpose provided the copies are not offered for sale.

Monisha Devi

Roll No: 22MCA010

Date:

Declaration

I **Tenzin Deckyi Namgyal**, Roll No. **22MCA015**, a registered candidate for Master of Computer Applications under Department of Computer Applications (MCA) of Sikkim University, Gangtok, India, declare that this is my own original work and does not contain material for which the copyright belongs to a third parry and that it has not been presented and will not be presented to any other University/Institute for a similar or any other Degree award.

I further confirm that for all third-party copyright material in my project report (including any electronic attachments), I have ”blanked out” third party material from the copies of the thesis fully referenced the deleted materials and where possible, provided links (url) to electronic sources of the material.

I hereby transfer exclusive copyright for this project report to Sikkim University. The following rights are resewed by the author:

a) The right to use, free of charge, all or part of this article in future work of their own, such as books and lectures, giving reference to the original place of publication and copyright holding.

b) The right to reproduce the article or thesis for their own purpose provided the copies are not offered for sale.

Tenzin Deckyi Namgyal

Roll No: 22MCA015

Date:

Certificate from the Supervisor

This is to certify that **Khusbu Pradhan**, Roll No. **22MCA006**, **Monisha Devi**, Roll No. **22MCA010** and **Tenzin Deckyi Namgyal**, Roll No. **22MCA015** are registered candidate for Master of Computer Applications Programme under the department of Computer Applications of Sikkim University, Gangtok, India.

The undersigned certifies that they have completed all other requirements for submission of the major project and hereby recommends for the acceptance of their project entitled **Developing a Movie Recommendation System Using Graph Neural Network** in the partial fulfillment of the requirements for the award of MCA Degree by Sikkim University, Gangtok, India.

Dr. Swarup Roy

Professor

Department Computer Applications

Sikkim University

Gangtok-737102, India

Date:

Certificate from the Head of Department

This is to certify that **Khusbu Pradhan**, Roll No. **22MCA006**, **Monisha Devi**, Roll No. **22MCA010** and **Tenzin Deckyi Namgyal**, Roll No. **22MCA015** are registered candidate for Master of Computer Applications Programme under the department of Computer Applications of Sikkim University, Gangtok, India.

The undersigned certifies that they have completed all other requirements for submission of the major project and hereby recommends for the acceptance of their project entitled **Developing a Movie Recommendation System Using Graph Neural Network** in the partial fulfillment of the requirements for the award of MCA Degree by Sikkim University, Gangtok, India.

Dr. Mohan Pratap Pradhan
Associate Professor
Head Of the Department of Computer Applications
Sikkim University
Gangtok-737102, India

Date:

Abstract

A recommendation system is a type of information filtering system that predicts or suggests items or content that a user may be interested in. Recommender systems have been extensively used to analyze user experience patterns and suggest potential targets to users and customers to provide a more personalized experience. Recently, Graph Neural Networks (GNNs) have emerged as the state-of-the-art approach for recommender systems. The basic idea behind GNNs involves using a neural network to obtain a representation for each node by aggregating the representations of neighboring nodes in a recursive manner, constrained to a certain network depth. These representations are then used to predict the probability of potential links between nodes in a graph. In this work, we employed GNNs to recommend movies to users. Here we have used the MovieLens 1M dataset which has three sets of data i.e movie features, item features and ratings(user-movie interaction). We experimented with various methods for calculating similarities between users and movies, as well as different loss functions to train the model and predict potential links between users and movies.

Keywords: Recommendation System, Graph Neural Network

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Recommendation System is part of our daily life where people rely on knowledge for making decisions of their personal interest. Recommendation systems are designed to suggest items that users might like or find useful. They work by analysing data about users' preferences, behaviours, and similarities with other users or items.

Movie recommendation systems address the challenge of information overload faced by viewers when choosing from a vast array of available films. By analyzing user behavior, preferences, and viewing history, these systems curate personalized lists of movie suggestions. This not only enhances user satisfaction but also increases engagement and retention on streaming platforms. esearch into movie recommendation systems has evolved significantly over the years, driven by advances in artificial intelligence and big data analytics. Early systems relied on basic collaborative filtering techniques, while modern approaches integrate complex algorithms that consider diverse factors such as genre preferences, user demographics, and social interactions.

Platforms like Netflix, Hulu, and Amazon Prime Video employ sophisticated recommendation systems to cater to millions of users worldwide. These systems leverage vast amounts of data to predict user preferences accurately, thereby improving the likelihood that users will discover and enjoy new movies aligned with their tastes.

Movie recommendation systems are indispensable tools that enhance the movie-watching experience by providing personalized suggestions tailored to individual preferences. As technology continues to evolve, these systems will play a crucial role in shaping how users discover, engage with, and enjoy movies in the digital age.

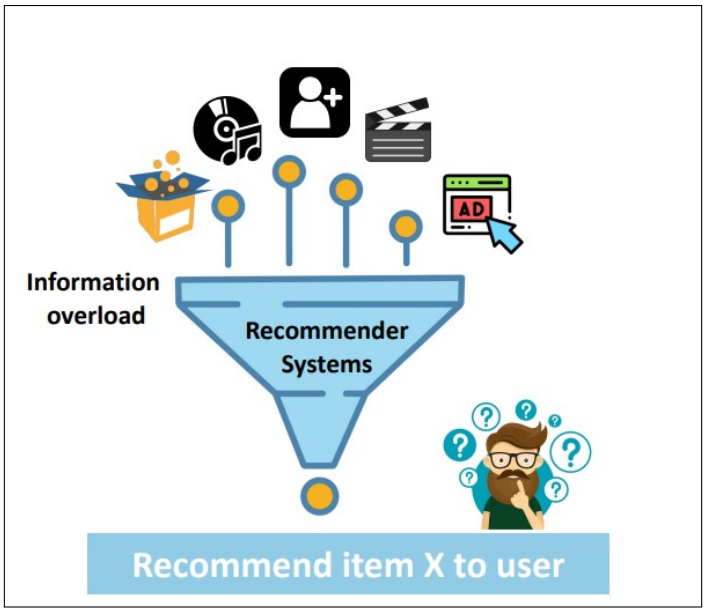

Figure 1.1: Recommendation System

1.1 Types of Recommendation System

There are several types of recommendation systems, but two common approaches are **collaborative filtering** and **content-based filtering**. Recommendation systems can also use hybrid methods that combine collaborative and content-based filtering, as well as other techniques such as matrix factorization and deep learning.

1.1.1 Content-based Filtering:

Content-based filtering recommends items to users based on individual item features. It makes recommendations to users based on their consumption history and generally becomes more accurate the more actions (inputs) the user takes. Instead of relying solely on user behaviour, content-based filtering analyses the attributes and keywords associated with items in a database.Content-based recommender systems use machine learning algorithms and data science techniques.

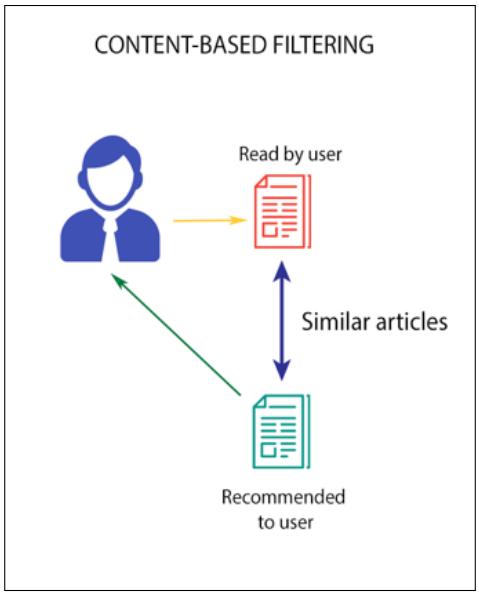

Figure 1.2: Content-Based Filtering

Content-based systems focus on properties of items. The idea is to find important characteristics of the items that are easily identifiable, which then serve to make up a ”profile”. Measuring the similarity between items. In this type of system is synonymous with measuring the similarities between their profiles. Another essential part of a content-based system is user profiles. Vectors must be created that describe the user's preferences in the same dimensionality as the item profiles. Once they are represented in the same dimensions, a utility matrix indicates the connection between users and items. The items which the users like are some aggregation of the profiles of those items. With profile vectors for both user and items, estimation can be made of the degree in which a user prefers an item, which is normally achieved by computing the cosine distance between the user's and item's vectors.[8]

1.1.2 Collaborative Filtering:

Collaborative filtering is a method of making predictions about the interests of a user based on information about the preferences of other users. The underlying assumption is that if person A has the same opinion as person B on some issues, then A is more likely to have similar opinions as B on different issues. There are many different forms of collaborative filtering systems. The two most popular ones are user-based collaborative filtering and item-based collaborative filtering.[8]

There are two primary types of collaborative filtering

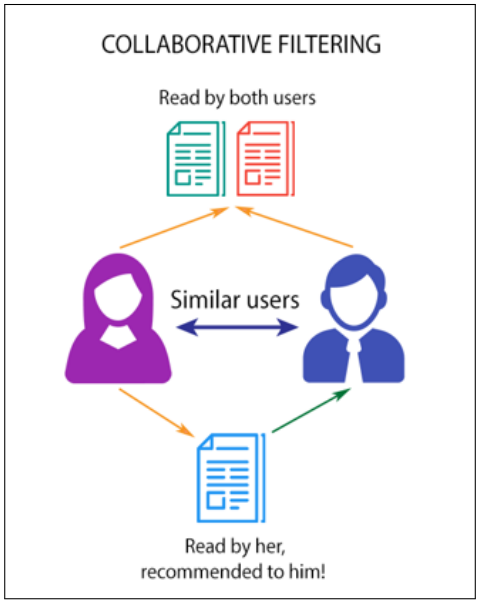

Figure 1.3: Collaborative Filtering

1. **User-based collaborative filtering:** Recommends items based on the behaviour of similar users.

2. **Item-based collaborative filtering:** Recommends items based on the similarity between items themselves.

1.2 Challenges of Recommendation System

Building a recommendation system using content based filtering techniques is simple, transparent and independent of other users. However there are some limitations using this method such as:

1.2.1 Cold start problem:

the system cannot produce efficient recommendations for new users who have not interacted with any items. It generally arises when a new user enters the system or new items are inserted.

1.2.2 Over-specialisation:

Where the system may recommend only items similar to those a user has already interacted with, potentially leading to a lack of diversity in recommendations.
Similarly, while building a recommendation system using collaborative filtering, it does overcome some of the problems presented by the content based filtering, but it still cannot solve the cold start problem.

1.3 Motivation

A graph-theoretic approach, combined with machine learning, may offer robust solutions to several challenges inherent in traditional recommendation systems. It effectively addresses the cold start problem by examining interactions between similar users and items, enabling recommendations even when new users have minimal input. By incorporating features, additional information can be attached to nodes, enriching the recommendation process. This method excels in understanding and managing complex relationships by analyzing the intricate connections between nodes, making it versatile for both structured and unstructured data. Consequently, the integration of graph theory and machine learning results in more accurate and dynamic recommendation systems.

1.4 Problem Definition

Design and development of effective graph based Movie Recommendation System using Graph Neural Network

1.5 Objective

1. **Design and implementing GNN based Movie Recommendation system:** To develop a movie recommendation system using Graph Neural Networks (GNNs) that accurately predicts user preferences and enhances the overall recommendation quality by effectively capturing the complex relationships between users and movies, as well as among the movies themselves.

2. **Performance of the model with variant features:** Initially we have only used ”Gender” as a user feature. But To improve the model's performance and provide more personalized recommendations, we expanded the user features to include additional demographic information such as ”Age” and ”Occupation”. This enhancement enables the model to better capture user preferences and tailor recommendations more effectively.

3. **Performance analysis with various similarity functions:** In the development of a GNN-based movie recommendation system, We employed a range of similarity measures, including Cosine similarity, Dot product similarity, Pairwise similarity(Euclidean distance) and Neural Network based similarity measure to calculate scores that capture the complex relationships between nodes in the graph.

Chapter 2

Literature Review

2.1 Introduction

There are various recommendation approaches like content-based filtering, collaborative filtering, and demographic filtering. These methods can be combined with different machine learning techniques to enhance the recommendation system's performance for movies. Each recommendation approach has its own advantages and disadvantages that affect the precision and effectiveness of the system.

Table 2.1: Research papers on Content Based Filtering

Paper	**Methodology**	**Dataset**	**Issues**
Movies Recommendation system using Cosine Similarity Shubham Pawar, Pritesh Patne et al.,2022, (IJISRT) [1]	Cosine similarity	TMDB (The Movies Database)	Cold start problem
Movies Recommendation System using Cosine Similarity and KNN, Ramni Harbir, et al., 2020, (IJEAT) [2]	Cosine similarity,KNN(k Nearest Neighbour)	TMDB (The Movies Database)	• Cold start problem • Overspecialization

Table 2.2: Research paper on Collaborative Based Filtering

Paper	Methodology	Dataset	Issues
Recommendation System using Machine Learning Techniques, Kalkar, Shailesh D., and Pramila M. Chawan, et al., 2022, (IRJEAT) [3]	• Cosine Similarity • KNN(K Nearest Neighbour) • SVD(Singular Value Decomposition). • Co Clustering • ALS (Alternating Least Square)	TMDB (The Movies Database)	• Cold Start Problem • Scalability

Table 2.3: Research paper on CNN

Paper	Methodology	Dataset	Issues
Convolutional Neural Network-Based Personalized Program Recommendation System for Smart Television Users, Dudekula, K.V., Syed, H, et al., 2023, [4]	CNN(Convolutional Neural Network)	• CelebA dataset • LFW-people dataset	• Cold Start Problem • Cannot handle complex dataset

2.2 Discussion

The literature identifies several key challenges in recommendation systems, including the cold-start problem, which arises when new users or items enter the system and there is insufficient data to make accurate recommendations. Overspecialization occurs when the system becomes too focused on specific attributes or features, limiting its ability to recommend diverse items. The lack of diversity problem results in users being presented with too many similar recommendations, failing to cater to different tastes and preferences. Additionally, incorporating features beyond the query or item ID is challenging, and traditional collaborative filtering models struggle to handle complex datasets with non-linear relationships and noise.

Graph-based approaches may offer effective solutions to the challenges in recommendation systems. By leveraging relationships between users and items, and incorporating additional metadata and side information into the graph, the cold-start problem can be mitigated. Graph-based methods can also address overspecialization by connecting items based on similarities in user interactions, leading to more diverse and personalized recommendations. Furthermore, graph-based approaches can solve the problem of recommending too many similar items by using connections between items and their characteristics, ensuring a variety of choices that cater to different tastes and preferences. Additionally, graph-based methods can seamlessly integrate side features into the recommendation process and effectively handle complex datasets with non-linear relationships and noise.

In the next chapter, we delve into the methodology, where we have graph theoretic approach to tackle the aforementioned challenges.

Chapter 3

Methodology

In this chapter, we delve into the Graph Neural Networks (GNNs), leveraging their capabilities to enhance recommendation systems. We will also discuss our proposed framework that uses GNNs to generate recommendations.

3.1 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural networks designed to perform inference on data described by graphs. They are particularly useful for tasks where the data is structured in a non-Euclidean space, such as social networks, molecule structures, or recommendation systems. GNNs leverage the relationships and interactions between entities (nodes) to learn and make predictions. The entities are represented as nodes and the relationship between them as edges.
In GNNs, message passing is a crucial process that enables nodes to exchange information and learn from each other. During each layer of the GNN, nodes send and receive messages to and from their neighbors, updating their representations based on the information received. The message passing process can be represented by the following formula:

$$h_u^{(k)} = \sigma\left(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_{\text{neighbor}}^{(k)} \sum_{v \in \mathcal{N}(u)} h_v^{(k-1)} + b^{(k)}\right) \tag{3.1}$$

Where: $h_u^{(k)}$ is the hidden state (feature representation) of node u at the k-th layer. σ is a non-

linear activation function, such as ReLU. $W_{\text{self}}^{(k)}$ is the weight matrix applied to the features of the node itself at the k-th layer. $W_{\text{neighbor}}^{(k)}$ is the weight matrix applied to the aggregated features of the neighbors of node u at the k-th layer. $\sum_{v \in \mathcal{N}(u)} h_v^{(k-1)}$ denotes the sum of the feature vectors of the neighbors of node u from the previous layer. $b^{(k)}$ is the bias term at the k-th layer.[7]

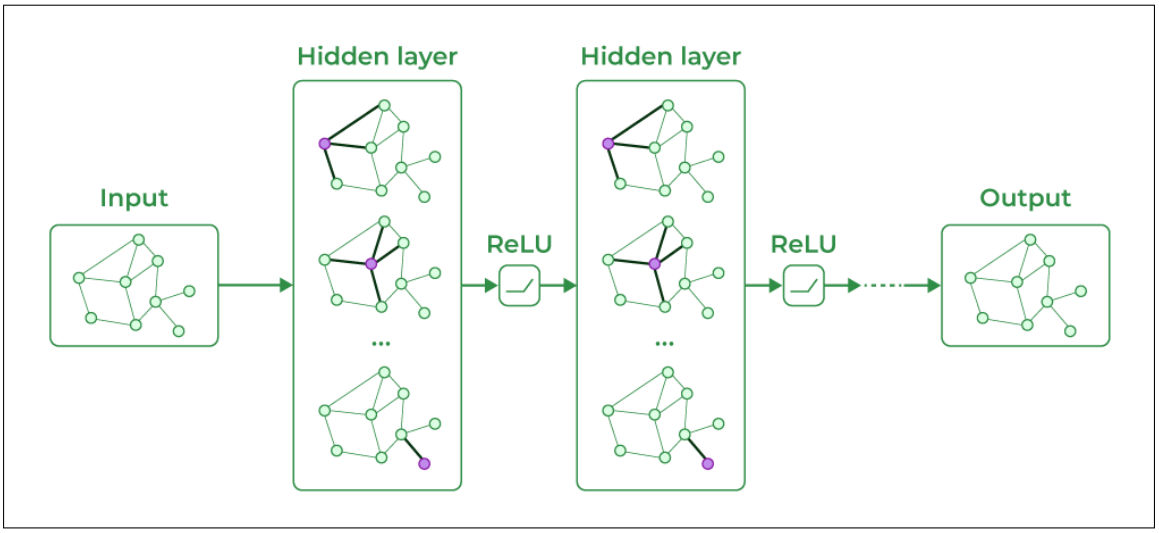

Figure 3.1: General representation of GNN Model

The goal is to develop a movie recommendation system that leverages the advanced capabilities of Graph Neural Networks (GNNs) to improve recommendation and user satisfaction.

3.2 Proposed Framework

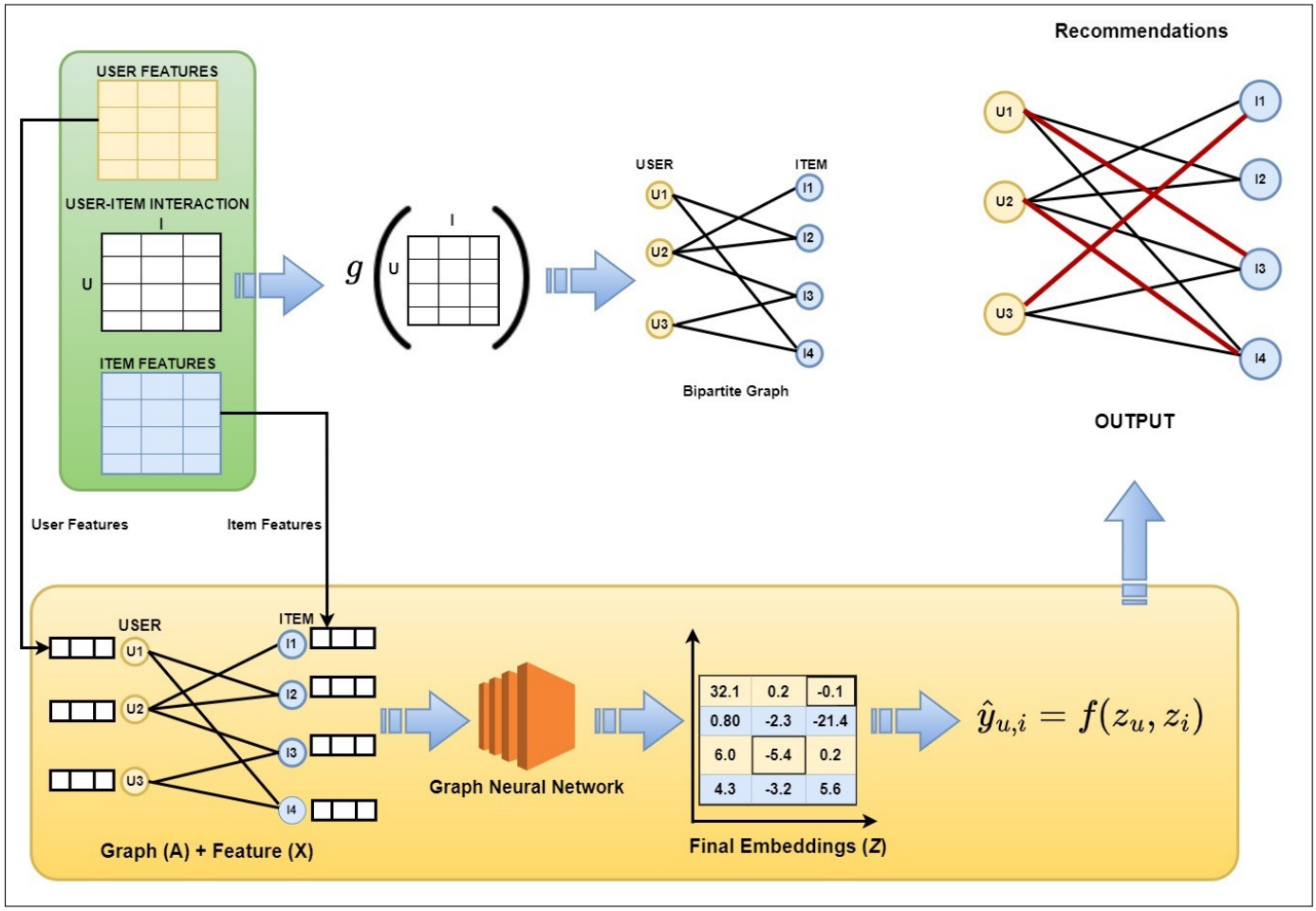

Figure 3.2: Proposed GNN based Movie Recommendation System

3.2.1 Graph Construction

Our recommendation system utilizes three key datasets: user features, movie features, and user-movie interactions. The interaction data is transformed into a bipartite graph, where users and movies are represented as nodes with distinct features as shown in figure 3.2. The user nodes embody user attributes, while the movie nodes encompass movie attributes, facilitating a comprehensive representation of user preferences and movie properties. The graph and the features are passed as inputs to the GNN model.[9]

3.2.2 GNN model

The GNN model processes the inputs using the HeteroGraphConv layer from the DGL library, which is designed for heterogeneous graphs with multiple types of nodes and edges[10]. This layer performs message passing across the graph, allowing for different types of interactions between user nodes and item nodes. It learns representations (embeddings) for each user and item node by aggregating information from neighboring nodes based on the specific types of edges connecting them.

3.2.2.1 Training the model

The model is trained to minimize the loss function, which is optimized using an optimizer such as Adam(Adaptive Moment Estimation) [11]. The training process involves iteratively feeding edge data batches, computing the loss, and updating the model's parameters to improve performance. The loss is calculated using a specified loss function.

3.2.2.2 Loss Function

The loss function is a measure of how well the model is performing. We have used different loss functions such as Bayesian personalized Ranking, Max margin loss.
The loss functions used in our recommendation system are as follows:

1. Bayesian Personalized Ranking (BPR) Loss:BPR loss is a pairwise loss function used in the area of recommender systems. BPR assumes that observed interactions indicating user preferences should be assigned higher prediction values than unobserved interactions. Through this loss function, BPR essentially attempts to discriminate between positive and negative samples[13]

$$LCF = \sum_{(u,i,j) \in O} - \ln \sigma(\hat{y}(u,i) - \hat{y_1}(u,j)) \tag{3.2}$$

where, $O = \{(u,i,j)|(u,i) \in R^+, (u,j) \in R^-\}$ denotes the training set, $\hat{y}$ represents the actual outcomes, $\hat{y_1}$ represents the predicted outcomes, R^+ represents the observed positive interactions between user u and item i, and R^- indicates the sampled unobserved (negative) interaction set. The function $\sigma(.)$ is the sigmoid function.

2. Max Margin Loss:The basic idea of max-margin based loss function is to maximize the similarity score of positive examples and minimize the similarity score of negative examples [14]

$$J_G(z_q z_i) = \mathbb{E}_{n_k \sim P_n(q)} \max\{0, z_q \cdot z_{n_k} - z_q \cdot z_i + \Delta\} \tag{3.3}$$

where, $P_n(q)$ represents the distribution of negative examples for item q, and Δ denotes the margin hyperparameter. If Δ is too high, the task would be very difficult for the model to learn, and if it is too low, the learning performance would diminish due to low scores given to negative pairs.

3.2.3 Embeddings

The Graph Neural Network (GNN) generates final embeddings for both users and items, denoted as z_u and z_i, respectively, after k rounds of message passing and aggregation. Specifically, the embeddings are produced by iteratively applying the update formula mentioned in the equation 3.1. After k rounds, the resulting embeddings capture the essential characteristics of users and items in a lower-dimensional vector space, where semantically similar users and items are mapped to nearby points. These embeddings are then utilized to compute the score for recommendation.

3.2.4 Similarity measure

The learned embeddings z_u for user u and z_i for item i are used to predict the score ($\hat{y}_{u,i}$) between users and items using a prediction function $f(\mathbf{z}_u, \mathbf{z}_i)$. The similarity function is a measure of the similarity or dissimilarity between two data points. We have experimented with different similarity functions such as follows:

1. Cosine similarity (cos): It is a measure of similarity between two non-zero vectors of an inner product space. It is defined as the cosine of the angle between the two vectors. It is essentially the dot product scaled by magnitude, and because of scaling it is normalized between 0 and 1. The source and destination node embeddings are normalised and a function

is applied between them to calculate the similarity score.1

$$\hat{y}_{ui} = \frac{\mathbf{z}_u \cdot \mathbf{z}_i}{\|\mathbf{z}_u\| \|\mathbf{z}_i\|} \tag{3.4}$$

2. Dot Product Similarity (dotprod): Dot product similarity, also known as the inner product, is a measure of similarity between two vectors in an inner product space. It measures the scalar product of two vectors. Scalar value is a single number or quantity without direction or magnitude. A function is applied on the source and destination node embeddings to calculate the similarity score.1

$$\hat{y}_{ui} = \mathbf{z}_u \cdot \mathbf{z}_i \tag{3.5}$$

3. Neural Network Similarity (NN): It is a type of similarity function that uses the concept of neural network to calculate the similarity score. In this two hidden layers are created and an activation function (ReLu) is applied after each layer. An output layer is applied to the result of the second layer after which a Sigmoid function is used to produce the similarity score between 0-1.

$$\hat{y}_{ui} = NN(\mathbf{z}_u \cdot \mathbf{z}_i) \tag{3.6}$$

4. Pairwise Distance Similarity Function (PW): It measures the Euclidean distance between two points in a vector space or the distance between the source node embeddings and destination node embeddings for each edge in the graph.1

$$\hat{y}_{ui} = -\|\mathbf{z}_u \cdot \mathbf{z}_i\| \tag{3.7}$$

$$\textbf{where, } \|\mathbf{z}_u \cdot \mathbf{z}_i\| = \sqrt{\sum_j (z_{ij} \cdot z_{ij})^2}$$

[1].

[1] https://developers.google.com/machine-learning/clustering/similarity/measuring-similarity

3.2.5 Final recommmendations

The items are sorted based on their similarity scores. The higher the score, the more similar the item is to the user's preferences. Based on this the top k items are recommended to the user.

In the next chapter, we will present the experimental results and performance evaluation of our proposed framework.

Chapter 4

Experimental Result

The performance of the GNN model was assessed by varying several parameters to observe its efficacy in generating accurate recommendations. This assessment measured the model's ability to distinguish between positive edges (interaction between the user and item) and negative edges (no interaction between the user and item) and how closely the predicted outcomes align with the actual outcomes.

4.1 Dataset

We utilised the MovieLens 1M dataset retreived from GroupLens which is a research lab in the Department of Computer Science and Engineering at the University of Minnesota, it is a widely-used benchmark dataset for recommendation system research. The dataset contains information about users, movies, and their interactions, including ratings, genres, and timestamps.
The user and movie features were converted from categorical features to binary variables using the one hot encoding. We constructed a user-item interaction graph where users and movies are nodes, and edges represent the interactions (ratings) between them.

Table 4.1: Dataset Statistics

Dataset	Users	Movies	Ratings
MovieLens	6040	3,900	1,000,209

4.2 Experimental setup

For the development of this project, several software and hardware tools were utilized. The software tools included Python 3.8 as the programming language, with frameworks such as PyTorch, TensorFlow, and the Deep Graph Library (DGL). Development work was conducted using Jupyter Notebook. On the hardware side, the project was supported by a system equipped with 25.6 GiB of memory, an Intel Xeon(R) processor, and a 1 TiB hard disk.

4.2.1 Hyperparameter Tuning

Hyperparameters are configuration settings used to structure the model and govern its training process and play a crucial role in the model's ability to learn from data and generalize to new, unseen examples. We performed hyperparameter tuning to optimize the performance of our machine learning model by carefully selecting the best set of hyperparameters. By altering the values of each hyperparameter, we aim to enhance the model's accuracy, minimize overfitting, and ensure robust and reliable predictions.
The following parameters were used to train the model:

- **Epoch:** The number of times the model gets trained on the entire dataset.
- **Dropout:** It is a regularization technique where a fraction of the neurons (in the hidden layer) are randomly ignored or set to zero to prevent overfitting.
- **No. of layers:** The number of hidden layers in the neural network.
- **Hidden dimension (Hid Dim):**When processing information in a GNN, each node typically maintains a hidden state that encodes information about the node and its neighborhood. The hidden dimension determines the size or the number of features in this hidden state vector.
- **Output dimension (Out Dim):** The output dimension refers to the size of the vector produced as the final output of the GNN for each node or for the entire graph.
- **Patience:** It is the early stopping mechanism. The model will stop training if there is no improvement on the loss for 'n' consecutive epochs.

- **Weight decay:** It is a small penalty added to the training process to keep the weights from getting larger. The ”weights” generally refer to the parameters of the neural network.
- **Prediction Function (Pred):** The similarity functions used like cosine, dot product, etc.
- **Loss Function (Loss):** It is a way to measure how good or bad a model's predictions are compared to the actual outcomes.
- **Learning rate:** The learning rate in a GNN is a small number that determines how much the model's parameters (weights) are adjusted during each step of training. It controls the size of the steps the model takes as it learns from the data to minimize errors and improve its predictions.
- **k:** The number of top recommendations out of total.

4.3 Result Discussion

We experimented with different similarity functions including cosine similarity, dot product, NN similarity and Pairwise similarity function the similarity score obtained is used for ranking the recommended movies. Our experiments showed that using Cosine similarity function gave the best results. Loss calculation is another important field we focused on. We experimented with Max Margin Loss and BPR (Bayesian Personalised Ranking). Based on the results of our experiments BPR was better suited to our problem.
The results for the following metrics 4.3.1 were observed after performing hyperparameter tuning. These metrics help us to determine the accuracy of the model in making better recommendations.

4.3.1 Assessment parameters

1. **AUC:** The type of AUC used is ROC AUC which is the area under the ROC curve that plots the true positive rate against the false positive rate. In our case the AUC score determines whether the model is able to distinguish between positive edges and negative edges.

Training AUC: This metric measures how well the model distinguishes between different classes (e.g., liked vs. not liked movies) during training. A higher AUC indicates better performance, with 1 being perfect and 0.5 indicating random guessing.

Test AUC: This metric evaluates the model's ability to distinguish between different classes on the test or unseen dataset. Like Training AUC, it measures the area under the ROC curve. A higher Test AUC indicates that the model generalizes well to new data and is effective at making accurate predictions on the test set.

2. **Loss:** It quantifies the difference between the predicted movie ratings and the actual ratings given by the users.

Training Loss: This is a measure of how well the model is performing on the training data. The training loss is minimized during the training process, and a lower training loss indicates better performance on the training set.

Validation Loss: This measures the model's performance on the test dataset, which is not used during training. It helps to evaluate how well the model generalizes to new data. A lower validation loss indicates better performance. If the validation loss is significantly higher than the training loss, it may suggest that the model is overfitting.

Table 4.2: Optimal result for each metric

Metric	Optimal result
AUC	>0.5 or 1
Loss	<0.5 or 0

Table 4.3: Range of output

Metric	Range
AUC	0 to1
Loss	0 to 1

4.4 Results Obtained

The values of the hyperparameters were altered with respect to the **similarity or prediction function** to determine the best results for the assessment parameters (4.3.1). We discuss the results obtained for each similarity function by altering the values of certain hyperparameters.

4.4.1 Interim Results

The model was trained for the following user feature. The best results for the respective user feature are discussed below.

Table 4.4: User Feature

User Feature	
UserID	Gender

4.4.1.1 Cosine Similarity Function

The model was trained for epochs 30 and 60 and loss function BPR and MaxMargin.

- **Epoch=60 and Loss Function=BPR**

  The Table: 4.5 shows the results for AUC and loss on both training and test dataset. The prediction function taken is cos and the loss function used is BPR (Bayesian Personalised Ranking). The model was trained for 60 epochs and patience was set to 30.
  The Fig:4.1 and Fig:4.2 shows the graphical representation of the Table:4.5. We can see that the model has given fairly good results for AUC on both training and test dataset. However, we can see that the validation loss is greater than the training loss which means that the model was overfitting.

Table 4.5: Performance result of Cosine function with respect to BPR

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=cos	0.86	0.77	0.5466	0.6075
Epoch=60				
Patience=30				
Loss=BPR				

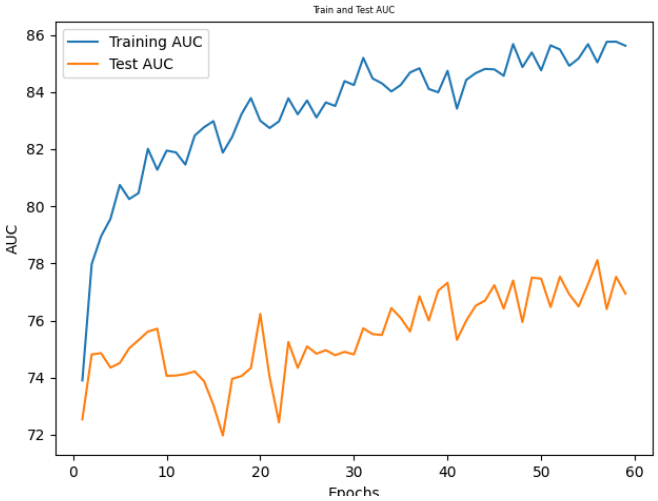

Figure 4.1: Training and Test Performance with Epoch=60

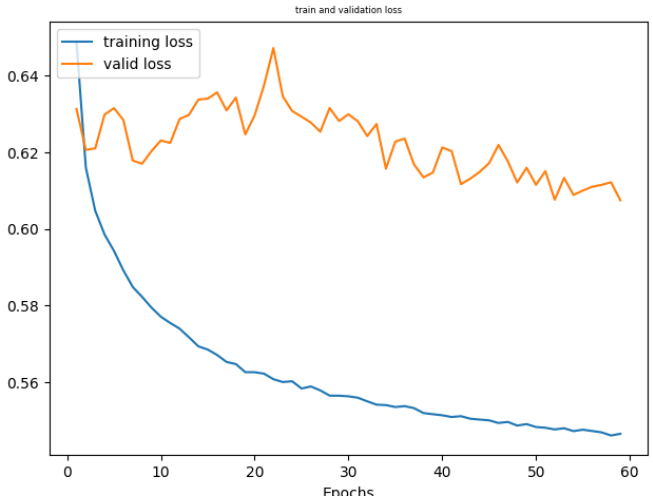

Figure 4.2: Training and Validation Performance with Epoch=60

- **Epoch=30 and Loss Function=MaxMargin**

  The Table:4.6 shows the results for AUC and loss on both training and test dataset. The prediction function taken is cos and the loss function used is Max Margin. The model was trained for 30 epochs and patience was set to 20.

  The Fig:4.3 and Fig:4.4 shows the graphical representation of the Table:4.6. We can see that for prediction function=cos the model has given comparatively less better results than the previous (Fig:4.1) for AUC on both training and test dataset. Although the loss has improved than before still the validation loss is much greater than the training loss which means that the model was overfitting.

Table 4.6: Performance result of Cosine function with respect to MaxMargin

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=cos	0.85	0.72	0.1922	0.3405
Epoch=30				
Patience=20				
Loss=MaxMargin				

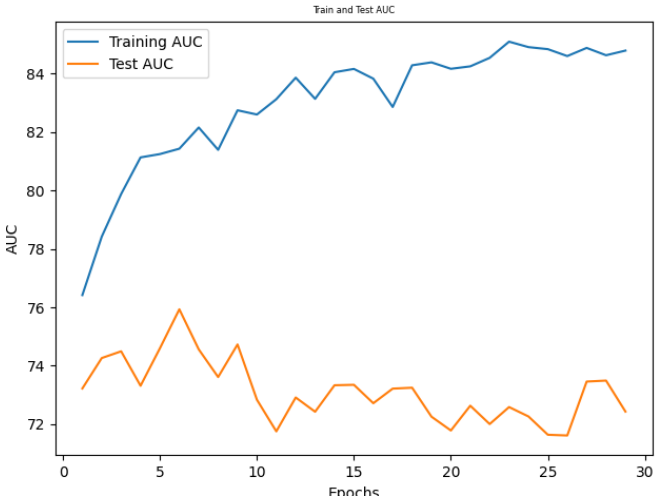

Figure 4.3: Training and Test Performance with Epoch=30

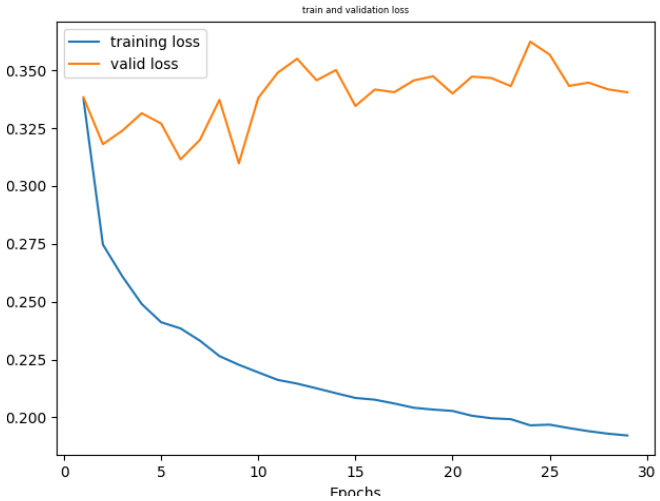

Figure 4.4: Training and Validation Performance with Epoch=30

4.4.1.2 Neural Network Similarity Function

The model was trained for epochs 30 and 60 and loss function BPR and MaxMargin.

- **Epoch=60 and Loss Function=BPR** The Table:4.7 shows the results for AUC and loss on both training and test dataset. The prediction function taken is NN(Neural Network) and the loss function used is BPR. The model was trained for 60 epochs and patience was set to 30. The Fig:4.5 and Fig:4.6 shows the graphical representation of the Table:4.7. The training AUC is equal to cos (Fig:4.3) but the test AUC is greater than cos (Fig:4.3). The model is not able to properly distinguish between predicted and actual data as in prediction function=cos and loss function=maxmargin (Fig:4.4) since the loss has increased also the validation loss is greater than the training loss which means that the model was overfitting.

Table 4.7: Performance result of NN function with respect to BPR

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=NN	0.85	0.76	0.5536	0.6298
Epoch=60				
Patience=30				
Loss=BPR				

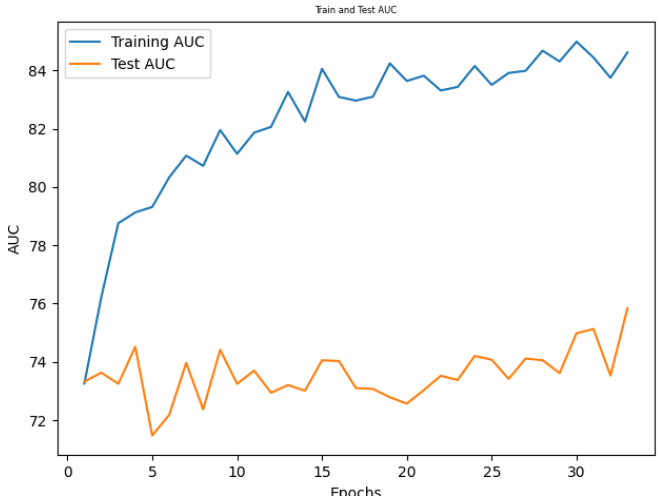

Figure 4.5: Training and Test Performance with Epoch=60

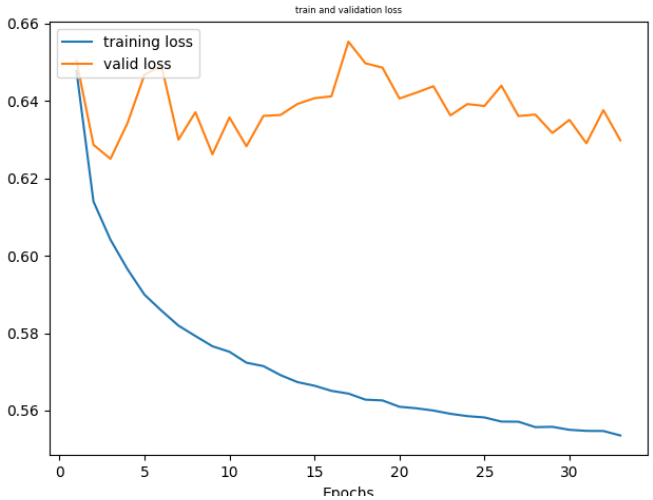

Figure 4.6: Training and Validation Performance with Epoch=60

- **Epoch=30 and Loss Function=MaxMargin**

  The Table:4.8 shows the results for AUC and loss on both training and test dataset. The prediction function taken is NN(Neural Network) and the loss function used is MaxMargin. The model was trained for 30 epochs and patience was set to 20.

  The Fig:4.7 and Fig:4.8 shows the graphical representation of the Table:4.8. The training and test AUC is equal to the previous (Fig:4.5). Also the training and validation loss has improved as the values are comparatively less than before (Fig:4.6).

Table 4.8: Performance result of NN function with respect to MaxMargin

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=nn	0.85	0.76	0.1951	0.3291
Epoch=30				
Patience=20				
Loss=MaxMargin				

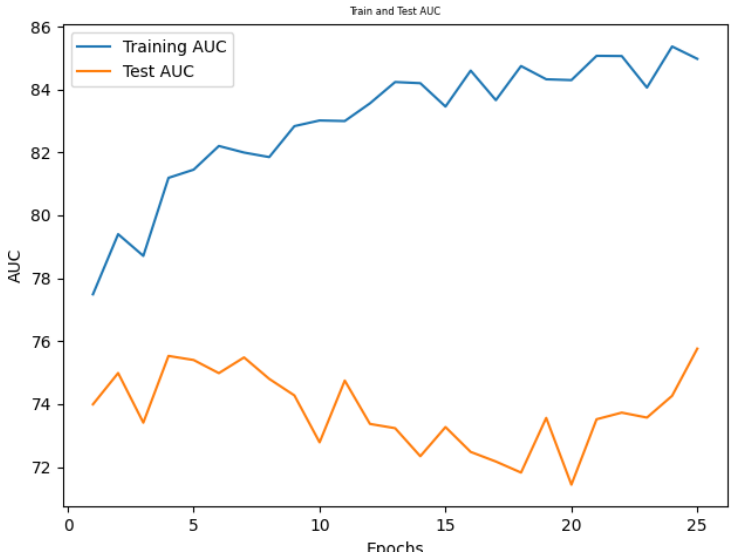

Figure 4.7: Training and Test Performance with Epoch=30

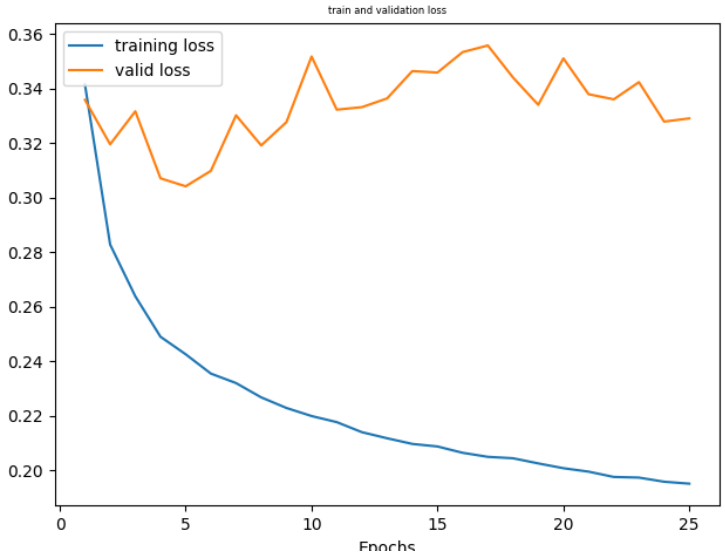

Figure 4.8: Training and Validation Performance with Epoch=30

The above results Table:4.5, Table:4.6, Table:4.7, Table:4.8 are interim and were evaluated with respect to only one user's feature (gender). However, we have added one more feature of the user whose results are discussed later. The overall result of the interim evaluation is given below. The best values for both AUC and Loss were observed for Cos with loss function=bpr for epoch=60.

Table 4.9: Optimized model configuration for interim results

	Training AUC	**Test AUC**	**Training Loss**	**Validation Loss**
Prediction Function=Cos				
Epoch=60				
Patience=30				
Loss Function=bpr				
Dropout=0.3				
Hidden Dimension=256	**0.86**	**0.77**	**0.5466**	**0.6075**
Output Dimension=64				
Learning rate=0.001				
No. of recommendation=5				
No. of layers=3				
Weight decay=1e-5				

4.4.2 Final Results

The model was trained for the following user feature. The best results for the respective user feature are discussed below.

Table 4.10: User Features

User Feature		
User ID	Gender	Age

4.4.2.1 Cosine Similarity Function

The model was trained for epochs 30 and 60 and loss function BPR and MaxMargin.

- **Epoch=60 and Loss Function=BPR**

  The Table:4.11 shows the results for AUC and loss on both training and test dataset. The prediction function taken is cos and the loss function used is BPR (Bayesian Personalised Ranking). The model was trained for 60 epochs and patience was set to 30.

  The Fig:4.9 and Fig:4.10 shows the graphical representation of the Table:4.11. We can see that the model has given good results after increasing the features for AUC on both training and test dataset. However, the validation loss is greater than the training loss so the model was overfitting.

Table 4.11: Performance result of Cosine function with respect to BPR

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=cos	0.84	0.78	0.5518	0.6075
Epoch=60				
Patience=30				
Loss=BPR				

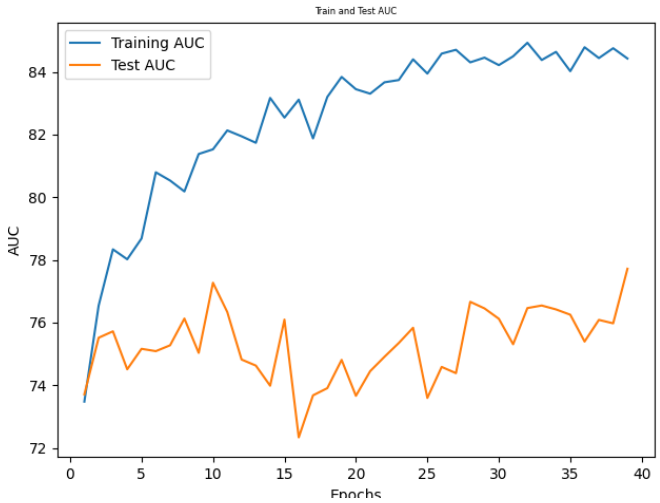

Figure 4.9: Training and Test Performance with Epoch=60

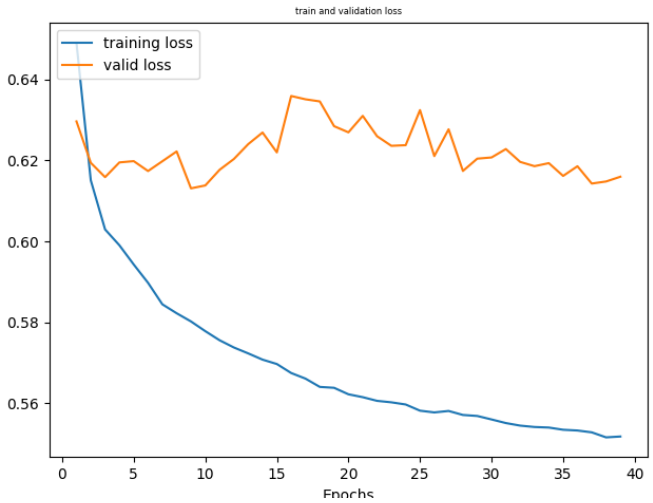

Figure 4.10: Training and Validation Performance with Epoch=60

- **Epoch=60 and Loss Function=MaxMargin**

  The Table:4.12 shows the results for AUC and loss on both training and test dataset. The prediction function taken is cos and the loss function used is MaxMargin. The model was trained for 60 epochs and patience was set to 30.

  The Fig:4.11 and Fig:4.12 shows the graphical representation of the Table:4.12. We can see that the train and test AUC has decreased on both training and test dataset. However, the training and validation loss is fairly good than previous results but the validation loss is greater than the training loss so the model was overfitting.

Table 4.12: Performance result of Cosine function with respect to other parameters

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=cos				
Epoch=60				
Patience=30				
Loss=MaxMargin	0.83	0.73	0.2141	0.3565
Dropout=0.5				
Learning rate=0.01				
No. of layers=3				
Hid Dim=128				
Out Dim=32				

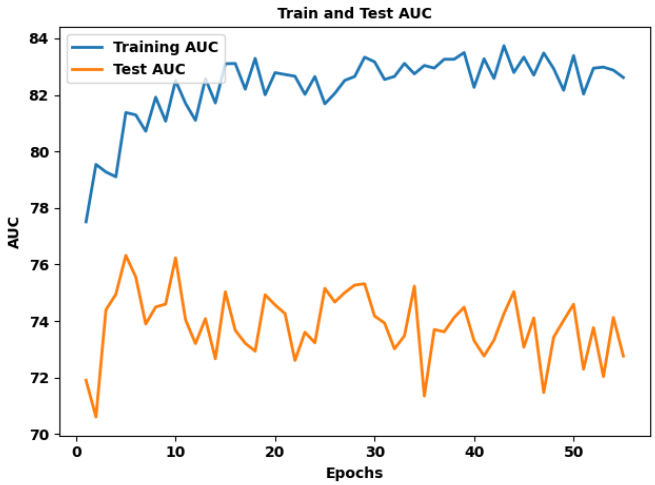

Figure 4.11: Training and Test Performance with Epoch=60

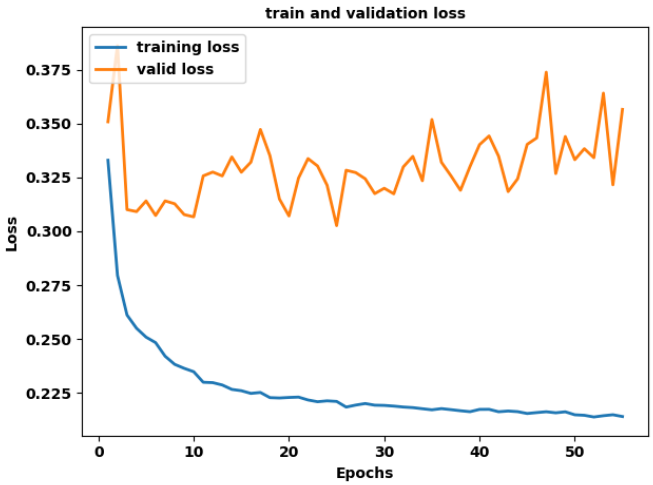

Figure 4.12: Training and Validation Performance with Epoch=60

4.4.2.2 Neural Network Similiarity Function

The model was trained for epochs 30 and 60 and loss function BPR and MaxMargin.

- **Epoch=30 and Loss Function=BPR**

  The Table:4.13 shows the results for AUC and loss on both training and test dataset. The prediction function taken is NN and the loss function used is BPR. The model was trained for 30 epochs and patience was set to 20.

  The Fig:4.13 and Fig:4.14 shows the graphical representation of the Table:4.13. We can see that the train and test AUC is moderately good as compared to loss on both training and test dataset. However, the validation loss is greater than the training loss so the model was overfitting.

Table 4.13: Performance result of NN function with respect to other parameters

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=nn	0.82	0.75	0.5754	0.6298
Epoch=30				
Patience=20				
Loss=BPR				
Dropout=0.5				
Learning rate=0.01				
No. of layers=3				
Hid Dim=128				
Out Dim=32				

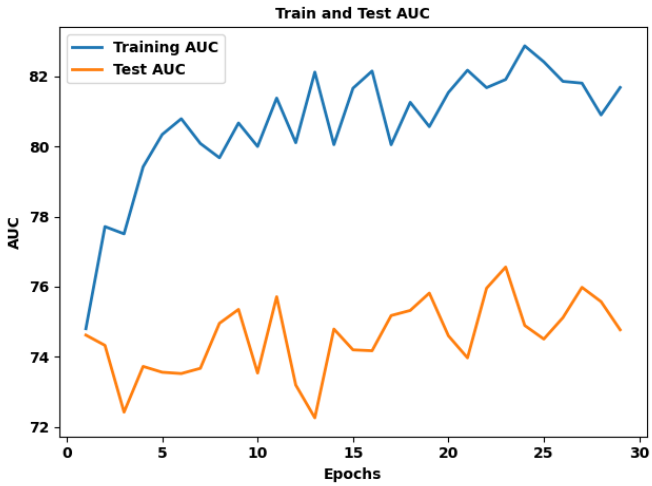

Figure 4.13: Training and Test Performance with Epoch=30

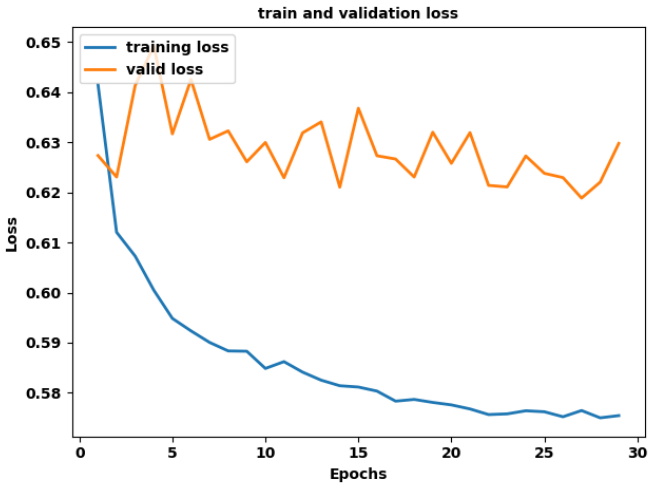

Figure 4.14: Training and Validation Performance with Epoch=30

- **Epoch=30 and Loss Function=MaxMargin**

  The Table:4.14 shows the results for AUC and loss on both training and test dataset. The prediction function taken is NN and the loss function used is MaxMargin. The model was trained for 30 epochs and patience was set to 20.

  The Fig:4.15 and Fig:4.16 shows the graphical representation of the Table:4.14. We can see that the train AUC has decreased whereas the test AUC remain the same. However, the training and validation loss is fairly good than previous results (Table:4.13) but the validation loss is greater than the training loss so the model was overfitting.

Table 4.14: Performance result of NN function with respect to other parameters

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=nn	0.83	0.75	0.2175	0.3142
Epoch=30				
Patience=20				
Loss=MaxMargin				
Dropout=0.5				
Learning rate=0.01				
No. of layers=3				
Hid Dim=128				
Out Dim=32				

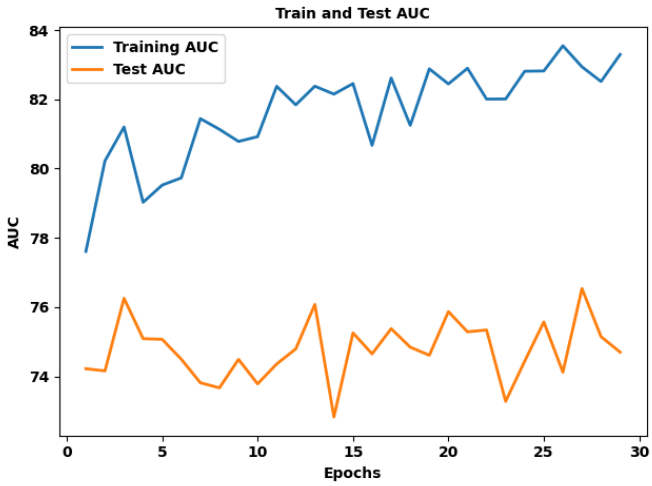

Figure 4.15: Training and Test Performance with Epoch=30

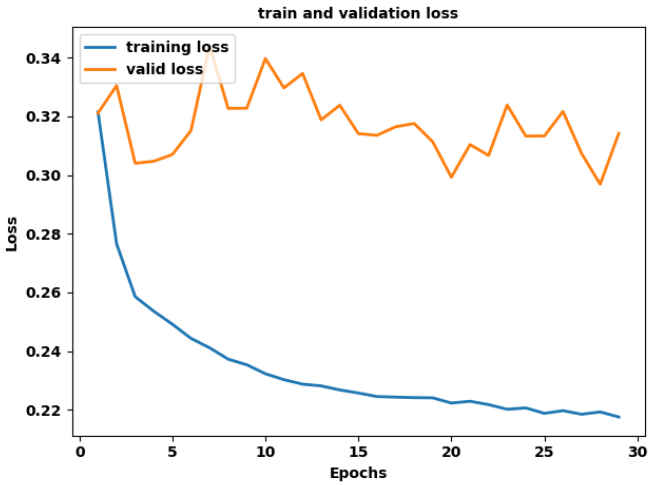

Figure 4.16: Training and Validation Performance with Epoch=30

4.4.2.3 Dot Product Similarity Function

The model was trained for epochs 30 and loss function BPR.

- **Epoch=30 and Loss Function=BPR**

  The Table:4.15 shows the results for AUC and loss on both training and test dataset. The prediction function taken is Dot Product and the loss function used is BPR. The model was trained for 30 epochs and patience was set to 20.

  The Fig:4.17 and Fig:4.18 shows the graphical representation of the Table:4.15. We can see that the test AUC is better than NN and PW but the validation loss is greater than the training loss so the model was overfitting.

Table 4.15: Performance result of Dot Product function with respect to other parameters

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=dotprod	0.82	0.76	0.5751	0.6227
Epoch=30				
Patience=20				
Loss=BPR				
Dropout=0.5				
Learning rate=0.01				
No. of layers=3				
Hid Dim=128				
Out Dim=32				

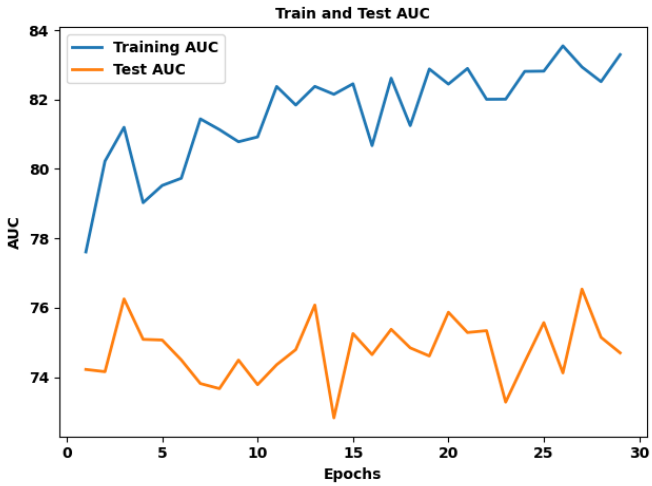

Figure 4.17: Training and Test Performance with Epoch=30

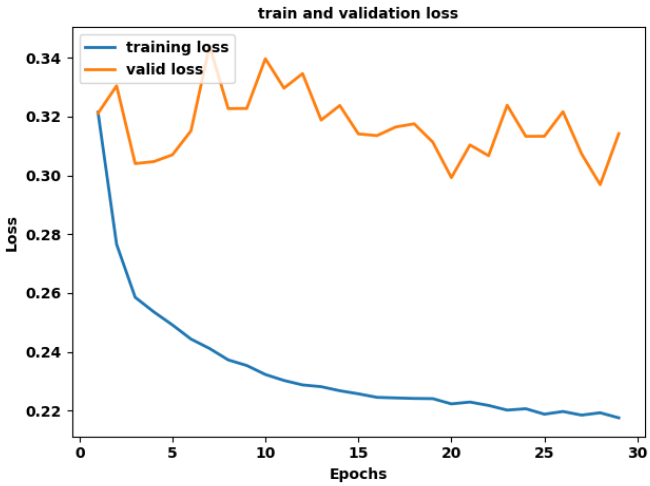

Figure 4.18: Training and Validation Performance with Epoch=30

4.4.2.4 Pairwise Distance Similarity Function

The model was trained for epochs 30 and loss function BPR.

- **Epoch=30 and Loss Function=BPR**

  The Table:4.16 shows the results for AUC and loss on both training and test dataset. The prediction function taken is PairWise and the loss function used is BPR. The model was trained for 30 epochs and patience was set to 20.

  The Fig:4.19 and Fig:4.20 shows the graphical representation of the Table:4.16. We can see that the test AUC is the lowest for PW as compared to other similarity functions. The results for loss is not good enough and the validation loss is greater than the training loss so the model was overfitting.

Table 4.16: Performance result of PairWise function with respect to other parameters

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=pw	0.82	0.72	0.5799	0.6721
Epoch=30				
Patience=20				
Loss=BPR				
Dropout=0.5				
Learning rate=0.01				
No. of layers=3				
Hid Dim=128				
Out Dim=32				

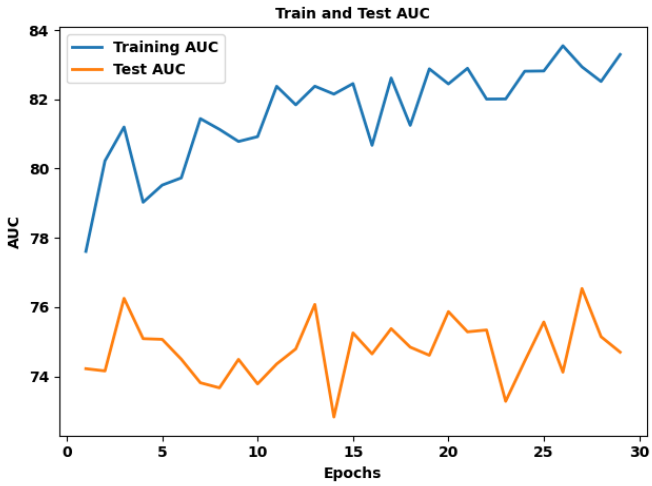

Figure 4.19: Training and Test Performance with Epoch=30

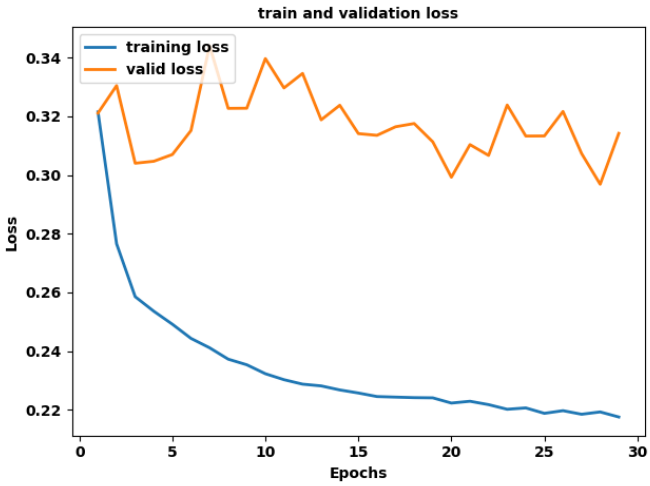

Figure 4.20: Training and Validation Performance with Epoch=30

4.4.2.5 Cosine Similarity Function

The model was trained for the following features only for cosine similarity function since we did not get satisfactory results.

Table 4.17: User Features

User Feature			
User ID	Gender	Age	Occupation

- **Epoch=60 and Loss Function=BPR**

  The Table:4.18 shows the results for AUC and loss on both training and test dataset. The prediction function taken is cos and the loss function used is BPR (Bayesian Personalised Ranking). The model was trained for 60 epochs and patience was set to 30.

  The Fig:4.21 and Fig:4.22 shows the graphical representation of the Table:4.18. We can see that after adding one more feature the test AUC has decreased for cosine function for loss=bpr which shows that the model's performance on the test dataset has effected negatively after adding the new feature. Also the validation loss is greater than the training loss so the model was overfitting.

Table 4.18: Performance result of Cosine function with respect to BPR

	Training AUC	Test AUC	Training Loss	Validation Loss
Pred=cos	0.84	0.74	0.5528	0.6346
Epoch=60				
Patience=30				
Loss=BPR				
Dropout=0.5				
Learning rate=0.01				
No. of layers=3				
Hid Dim=128				
Out Dim=32				

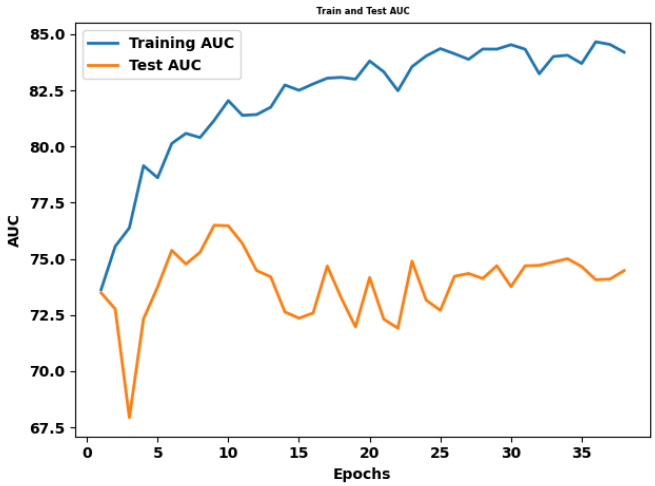

Figure 4.21: Training and Test Performance with Epoch=60

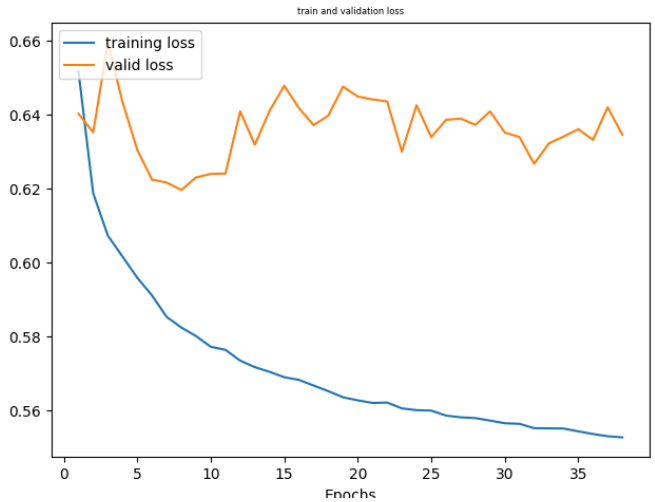

Figure 4.22: Training and Validation Performance with Epoch=60

The overall result of the final evaluation is given below. The best values for AUC and Loss were observed for Cos with loss function=bpr for epoch=60.

Table 4.19: Optimized model configuration for final results

	Training AUC	Test AUC	Training Loss	Validation Loss
Dropout=0.3	0.84	0.78	0.5518	0.6075
Epochs=60				
Prediction Function=cos				
Loss Function=bpr				
Number of layers=3				
Learning rate=0.001				
Hidden_dim=256				
Out_dim=64				

4.4.3 Comparison with different number of features

The graph (4.4.3) shows the relationship between the AUC (Area Under the Curve) of a model and the number of features used in the model. The trend of the graph shows that adding Age to Gender improves the model's performance. However, when Occupation is also added, the performance of the model decreases.

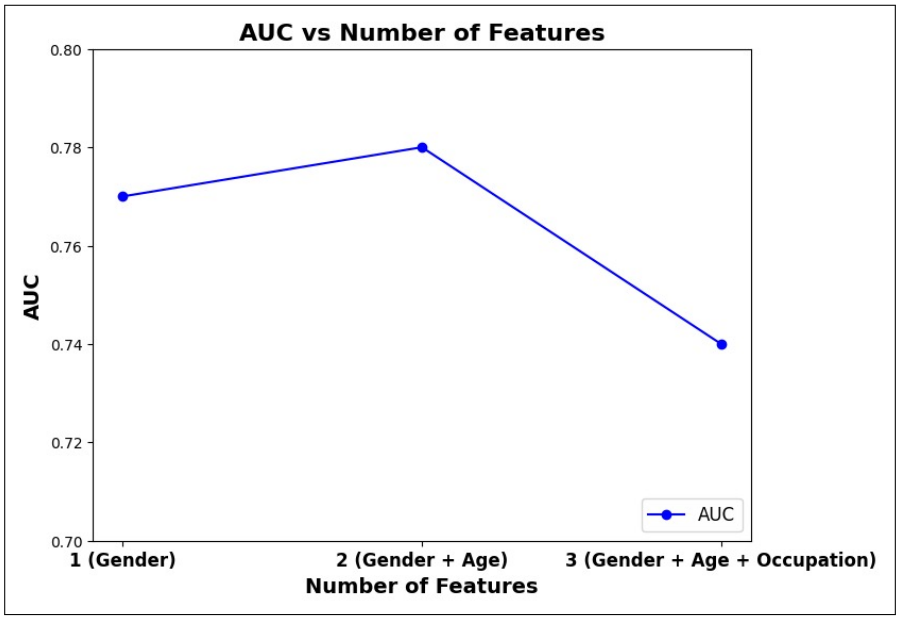

Figure 4.23: Comparison of AUC with respect to the number of features

The trend shows that when only the gender feature is used, the AUC is around 0.77. When the age feature is added to the model, the AUC increases to 0.78, indicating an improvement in the model's performance. However, when the occupation feature is also included, the AUC decreases to about 0.74, suggesting a decline in performance. This indicates that while the addition of age to gender enhances the model's accuracy, the inclusion of occupation introduces noise or irrelevant information, negatively impacting the model's performance. .

Chapter 5

Conclusion

Recommendation systems are used in almost every application to provide recommendations to the user catered to his/her preferences. The recommended items can include movies, music, products, etc. There are various methods through which we can build a recommendation system. In our study, we tried to build a movie recommendation system using Graph Neural Networks (GNNs), specifically employing the *HeteroGraphConv* architecture which effectively leveraged the graph structure of user-item interactions for recommendations. The experimental results were used to validate the effectiveness of our proposed methodology and its performance across various metrics like epochs, dropout rate, prediction function, etc. Despite the promising aspects of this project, the limitations must be acknowledged. One significant challenge is data sparsity, where the lack of sufficient user-item interactions can hinder the accuracy of recommendations. Scalability is also a concern, as the computational demands increase with the growth of users and items, potentially affecting the system's performance. Furthermore, there is a trade-off between accuracy and diversity in recommendations, where focusing too much on one can negatively impact the other. Additionally, users' dynamic preferences require the system to continuously adapt, which can be complex to implement effectively.

5.1 Further Work

For future work, we propose applying the proposed model to different recommendation systems to evaluate its generalizability and effectiveness across various domains. Specifically, we aim to

replace the current message passing layer with Graph Auto-Encoder (GAE) and Graph Attention Network (GAT) layers. This modification is expected to enhance the system's ability to capture and model complex relationships within the data. Additionally, we plan to develop a recommendation system that utilizes a combination of user-user graphs, item-item graphs, and user-interaction graphs. By integrating these different graph structures, the system will benefit from a more comprehensive understanding of user preferences and item characteristics, leading to improved recommendation accuracy and personalization.

Bibliography

[1] Shubham Pawar; Pritesh Patne; Priya Ratanghayra; Simran Dadhich; Shree Jaswal. ”Movies Recommendation System using Cosine Similarity.” Volume. Volume. 7 Issue. 4, April - 2022 , International Journal of Innovative Science and Research Technology (IJISRT)

[2] Singh, Ramni Harbir, et al. ”Movie recommendation system using cosine similarity and KNN.”International Journal of Engineering and Advanced Technology9.5 (2020): 556-559.

[3] Kalkar, Shailesh D., and Pramila M. Chawan. ”Recommendation System using Machine Learning Techniques.”Certified Journal(2022): 3-5.

[4] Dudekula, K.V.; Syed, H.; Basha, M.I.M.; Swamykan, S.I.; Kasaraneni, P.P.; Kumar, Y.V.P.; Flah, A.; Azar, A.T. Convolutional Neural Network-Based Personalized Program Recommendation System for Smart Television Users.Sustainability 2023,15, 2206.

[5] Lee, CheonSol, et al. ”Improving graph-based movie recommender system using cinematic experience.” Applied Sciences 12.3 (2022): 1493.

[6] Yang, Liangwei, et al. ”Consisrec: Enhancing gnn for social recommendation via consistent neighbor aggregation.” Proceedings of the 44th international ACM SIGIR conference on Research and development in information retrieval. 2021.

[7] Ankur Sarda Ankit Jain, Isaac Liu and Piero Molino. ”Food discovery with uber eats: Using graph learning to power recommendations”, 2019.

[8] Ajay Kaushik, Shubham Gupta and Manan Bhatia .”A Movie Recommendation System using Neural Networks”. IJARIIT, 2018

[9] Shi, Jinghan, et al. ”Heterogeneous graph neural network for recommendation.” arXiv preprint arXiv:2009.00799 (2020).

[10] TehraniJamsaz, Ali, et al. ”Perfograph: A numerical aware program graph representation for performance optimization and program analysis.” Advances in Neural Information Processing Systems 36 (2024).

[11] Ciampiconi, Lorenzo, et al. ”A survey and taxonomy of loss functions in machine learning.” arXiv preprint arXiv:2301.05579 (2023).

[12] Kingma, Diederik P., and Jimmy Ba. ”Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980 (2014).

[13] Rendle, Steffen. ”Factorization machines.” 2010 IEEE International conference on data mining. IEEE, 2010.

[14] Ciampiconi, Lorenzo, et al. ”A survey and taxonomy of loss functions in machine learning.” arXiv preprint arXiv:2301.05579 (2023).

Appendix

5.2 Create Adjacency Dict

```
adjacency_dict = {}

adjacency_dict['user_movie_rating'] = user_movie_train.Rating.values
adjacency_dict['user_movie_src'] = user_movie_train.user_new_id.values
adjacency_dict['user_movie_dst'] = user_movie_train.movie_new_id.values
```

5.3 Create Graph Schema based on Adjacency Dict

```
graph_schema = {('user', 'buys', 'movie'): list(zip(adjacency_dict['
    user_movie_src'], adjacency_dict['user_movie_dst'])),
                ('movie', 'watched-by', 'user'): list(zip(adjacency_dict['
                    user_movie_dst'], adjacency_dict['user_movie_src']))
                }
```

5.4 Message passing layer

```
class MessagePassing(nn.Module):

def __init__(self, input_features, output_features, dropout,):
    super().__init__()
    self._in_neigh_feats, self._in_self_feats = input_features
    self._output_features = output_features
    self.dropout = nn.Dropout(dropout)
    self.fc_self = nn.Linear(self._in_self_feats, output_features, bias=False)
    self.fc_neighbour = nn.Linear(self._in_neigh_feats, output_features, bias=
        False)
    self.fc_pre_agg = nn.Linear(self._in_neigh_feats, self._in_neigh_feats,
        bias=False)

def forward(self, graph, x):

    h_neighbour, h_self = x
    h_self = self.dropout(h_self)
    h_neighbour = self.dropout(h_neighbour)



    graph.srcdata['h'] = torch_nn_func.relu(self.fc_pre_agg(h_neighbour))
    graph.update_all(dgl_func.copy_src('h', 'm'), dgl_func.mean('m', 'neigh'))
    h_neighbour = graph.dstdata['neigh']

    #message passing
    z = self.fc_self(h_self) + self.fc_neighbour(h_neighbour)
    z = torch_nn_func.relu(z)

    z_normalization = z.norm(2, 1, keepdim=True)
    z_normalization = torch.where(z_normalization == 0, torch.tensor(1.).to(
        z_normalization), z_normalization)
    z = z / z_normalization

return z
```

5.5 GNN Model

```
class GNNModel(nn.Module):
def __init__(self, g, n_layers: int, dim_dict, dropout, pred,
    aggregator_hetero, embedding_layer,):

    super().__init__()
    self.embedding_layer = embedding_layer

    if embedding_layer:
        self.user_embed = NodeEmbedding(dim_dict['user'], dim_dict['hidden'])
        self.item_embed = NodeEmbedding(dim_dict['movie'], dim_dict['hidden'])

    self.layers = nn.ModuleList()

    # input layer
    if not embedding_layer:
        self.layers.append(
            dglnn.HeteroGraphConv(
                {etype[1]: MessagePassing((dim_dict[etype[0]], dim_dict[etype
                    [2]]), dim_dict['hidden'], dropout) for etype in g.
                    canonical_etypes},
                aggregate=aggregator_hetero)
                )

    # hidden layers
    for i in range(n_layers - 2):
        self.layers.append(
            dglnn.HeteroGraphConv(
                {etype[1]: MessagePassing((dim_dict['hidden'], dim_dict['
                    hidden']), dim_dict['hidden'], dropout) for etype in g.
                    canonical_etypes},
                aggregate=aggregator_hetero)
                )

    # output layer
```

```
        self.layers.append(
            dglnn.HeteroGraphConv(
                {etype[1]: MessagePassing((dim_dict['hidden'], dim_dict['hidden'])
                    , dim_dict['out'], dropout) for etype in g.canonical_etypes},
                aggregate=aggregator_hetero)
                 )

        if pred == 'cos':
            self.pred_fn = CosinePrediction()
        elif pred == 'nn':
            self.pred_fn = NnPredictingModule(NnSimilarityPredictingLayer,
                dim_dict['out'])
        elif pred == 'dotprod':
            self.pred_fn = DotProductPredictor()
        elif pred == 'pw':
            self.pred_fn = PairWiseDistancePredictor()
        else:
            raise KeyError('Prediction function does not exist')
            sys.exit(1)

    def get_repr(self, blocks, h):
        for i in range(len(blocks)):
            layer = self.layers[i]
            h = layer(blocks[i], h)
        return h

    def forward(self, blocks, h, pos_g, neg_g, embedding_layer: bool=True, ):
        if embedding_layer:
            h['user'] = self.user_embed(h['user'])
            h['movie'] = self.item_embed(h['movie'])

        h = self.get_repr(blocks, h)
        pos_score = self.pred_fn(pos_g, h)
        neg_score = self.pred_fn(neg_g, h)
    return h, pos_score, neg_score
```

5.6 Train the Model

```
def train_model(model,
                num_epochs,
                num_batches_train_loss,
                num_batches_validation_loss,
                num_batches_val_metrics,
                num_batches_train_metric,
                edgeLoad_train,
                edgeLoad_validation,
                nodeLoad_validation,
                nodeLoad_train,
                k,
                loss_function,
                delta,
                negative_sample_size,
                cuda=False,
                device=None,
                optimizer=torch.optim.Adam,
                lr=0.001,
                train_graph=None,
                validation_graph=None,
                out_dim=None,
                ground_truth_train=None,
                ground_truth_validation=None,
                result_filepath=None,
                patience=5,
                prediction=None,
                embedding_layer=True,
                ):

    model.train_loss_list, model.train_precision_list, model.
        train_coverage_list, model.train_auc_list, model.validation_auc_list,
        model.validation_loss_list, model.validation_precision_list, model.
        validation_coverage_list = [], [], [], [], [], [], [], []
```

```
best_metrics = {}

max_metric = - 0.1
patience_counter = 0
min_loss = 1.1

opt = optimizer(model.parameters(), lr=lr, weight_decay=weight_decay)

start = time.time()
print('Start training for '+ str(num_epochs)+ ' epochs')

for epoch in range(1, num_epochs):

    if epoch == 1:
        mode = 'w'
    else:
        mode = 'a'

    model.train()
    i = 0
    total_loss = 0
    for _, pos_g, neg_g, blocks in edgeLoad_train:
        opt.zero_grad()

        # Negative mask
        negative_mask = {}

        nids = neg_g.ndata[dgl.NID]

        for etype in pos_g.canonical_etypes:
            neg_src, neg_dst = neg_g.edges(etype=etype)
            neg_src = nids[etype[0]][neg_src]
            neg_dst = nids[etype[2]][neg_dst]
            negative_mask_tensor = train_graph.has_edges_between(neg_src,
                neg_dst, etype=etype)
            negative_mask[etype] = negative_mask_tensor.type(torch.float)
```

```
            if cuda:
                negative_mask[etype] = negative_mask[etype].to(device)

        if cuda:
            blocks = [b.to(device) for b in blocks]
            pos_g = pos_g.to(device)
            neg_g = neg_g.to(device)

        i += 1

        if i % 10 == 0:
            print("Edge batch " + str(i) + " out of ", str(
                num_batches_train_loss))

        input_features = blocks[0].srcdata['features']

        _, pos_score, neg_score = model(blocks, input_features, pos_g,
            neg_g, embedding_layer,)
        train_score = (pos_score, neg_score)
        loss = loss_function(pos_score, neg_score, delta, neg_sample_size,
            negative_mask=negative_mask, cuda=cuda, device=device,)

        loss.backward()
        opt.step()

        total_loss += loss.item()

    train_avg_loss = total_loss / i
    model.train_loss_list.append(train_avg_loss)

    model.eval()

    with torch.no_grad():
        total_loss = 0
        i = 0
```

```
for _, pos_g, neg_g, blocks in edgeLoad_validation:
    i += 1

    if i % 10 == 0:
        print("Edge batch {} out of {}".format(i,
            num_batches_validation_loss))

    # Negative mask
    negative_mask = {}

    nids = neg_g.ndata[dgl.NID]

    for etype in pos_g.canonical_etypes:
        neg_src, neg_dst = neg_g.edges(etype=etype)
        neg_src = nids[etype[0]][neg_src]
        neg_dst = nids[etype[2]][neg_dst]
        negative_mask_tensor = validation_graph.has_edges_between(
            neg_src, neg_dst, etype=etype)
        negative_mask[etype] = negative_mask_tensor.type(torch.
            float)

        if cuda:
            negative_mask[etype] = negative_mask[etype].to(device)

    if cuda:
        blocks = [b.to(device) for b in blocks]
        pos_g = pos_g.to(device)
        neg_g = neg_g.to(device)

    input_features = blocks[0].srcdata['features']
    _, pos_score, neg_score = model(blocks, input_features, pos_g,
        neg_g, embedding_layer,)

    validation_score = (pos_score, neg_score)
```

```
            validation_loss = loss_function(pos_score, neg_score, delta,
                neg_sample_size, negative_mask=negative_mask,
                                cuda=cuda, device=device,)
            total_loss += validation_loss.item()

        validation_avg_loss = total_loss / i
        model.validation_loss_list.append(validation_avg_loss)

    model.eval()

    with torch.no_grad():
        y = get_embeddings(train_graph, out_dim, model, nodeLoad_train,
            num_batches_train_metrics, cuda, device, embedding_layer,)
        train_precision, train_coverage = get_metrics(y, train_graph,
            model, out_dim, ground_truth_train, k, cuda, device,
            prediction, epoch)

        # validation metrics
        y = get_embeddings(validation_graph,
                           out_dim,
                           model,
                           nodeLoad_validation,
                           num_batches_val_metrics,
                           cuda,
                           device,
                           embedding_layer,
                           )

        validation_precision, validation_coverage = get_metrics(y,
            validation_graph, model, out_dim, ground_truth_validation,  k,
        cuda, device, prediction, epoch)

        AUC_val = compute_auc(validation_score)
        AUC_train = compute_auc(train_score)
        results = "Epoch " + str(epoch) +" | Training loss " + str(round(
            train_avg_loss,4)) +" | Training Precision " + str(round(
```

```
            train_precision * 100,2)) + " | Training AUC " + str(round(
            AUC_train,2))+ ' | Validation loss '+ str(round(
            validation_avg_loss,4)) + ' | Validation Precision '+str(round
            (validation_precision * 100,2)) + ' | AUC '+str(round(AUC_val
            ,2))


        print(results)
        print('process time: ' +str(round(float(time.time()-start)/60, 2))
            + ' minutes')

        save_txt(results, result_filepath, mode=mode)

        model.train_precision_list.append(train_precision * 100)
        model.validation_precision_list.append(validation_precision * 100)
        model.train_auc_list.append(AUC_train * 100)
        model.validation_auc_list.append(AUC_val * 100)

        if validation_precision > max_metric:
            max_metric = validation_precision
            best_metrics = { 'precision': validation_precision}


    if validation_avg_loss < min_loss:
        min_loss = validation_avg_loss
        patience_counter = 0
    else:
        patience_counter += 1

    if patience_counter == patience:
        break

viz_dict = {'train_loss_list': model.train_loss_list,
       'train_precision_list': model.train_precision_list,
       'train_auc_list': model.train_auc_list,
       'val_loss_list': model.validation_loss_list,
```

```
            'validation_precision_list': model.validation_precision_list,
            'validation_auc_list': model.validation_precision_list,
            }

    print('end of training!!')
    print ('process time: ' +str(round(float(time.time()-start)/60, 2))+ '
        minutes')
return model, viz_dict, validation_score, train_score, best_metrics
```

5.7 Model Inference

Apply the Trained Model to the Test Data

```
def model_inference(valid_graph, out_dim: int, trained_model, nodeLoad_test,
    num_batches_test: int, cuda: bool, device, embedding_layer:bool,
    ground_truth_test, all_eids_dict):

    trained_model.eval()

    with torch.no_grad():
            embeddings = get_embeddings(valid_graph, out_dim, trained_model,
                nodeLoad_test, num_batches_test, cuda, device, embedding_layer,)
            precision,_ = get_metrics(embeddings, valid_graph, trained_model,
                out_dim, ground_truth_test, k, cuda, device, prediction, epoch =
                 4)
            sentence = " TEST Precision {:.3f}% ".format(precision * 100)
            print(sentence)
            save_txt(sentence, result_filepath, mode= 'a')
```