Group Information

- **Group:** 89
- **Module:** 2
- **Name:** Xinyan Liu, No Personal Number, Bilateral Exchange Agreements Program
- **Email:** `liuxinya@chalmers.se`
- **Allowed to be finished by one person:** Yes

Declaration

I hereby declare that I have actively participated in solving every exercise. All solutions are entirely my own work, without having copied or referred to other solutions.

Time Spent

- **Hours Spent:** 17 hours

1 Reading and reflection

The Netflix Prize and its associated challenges represent landmark efforts in recommendation systems, primarily aiming to improve Netflix's movie recommendation algorithm, *Cinematch*. By releasing an anonymized dataset of over 100 million movie ratings, Netflix catalyzed advancements in collaborative filtering (CF), attracting global participation. The competition demonstrated that even marginal enhancements (e.g., a 10% improvement in RMSE) had substantial business implications for user engagement.

1.1 Key Design Features

Two critical aspects emerged from the competition:

1. Data Design and Challenges

- Dataset: User-movie rating interactions with inherent sparsity
- Challenges: Only 1-2% of possible user-movie pairs were rated
- Solutions: Matrix factorization and imputation techniques
- Contrast with e-commerce: Platforms like *Amazon* leverage richer contextual features (timestamps, purchase history, metadata), enabling hybrid CF + content-based models

2. Algorithmic Innovations

- Dominant approach: Latent factor models (matrix factorization)
- Critical techniques: Regularization to prevent overfitting
- Domain contrasts: Music platforms like *Spotify* prioritize sequence-aware models (RNNs, transformers) for temporal preference evolution
- Use of baselines:
  - Baselines, such as global average ratings or user/item biases, were critical in the Netflix Prize to establish a foundation for comparison.
  - They help isolate the incremental improvements of more complex models (e.g., matrix factorization, hybrid approaches).
  - Baselines improve accuracy by accounting for systematic biases (e.g., some users consistently rate higher or lower than others).

1.2 Conclusion

The Netflix Prize highlighted:

- The role of data/algorithm co-design in recommendation systems
- Domain-specific adaptation requirements:
  - CF (Collaborative Filtering) dominates movie recommendations
  - Temporal/contextual dynamics are crucial in domains like music streaming
- The importance of baselines:
  - Baselines provide a starting point for evaluating more sophisticated models.
  - They ensure that improvements are measured against a reasonable benchmark, avoiding overestimation of model performance.
  - Baselines are particularly useful in real-world systems where simplicity and interpretability are often as important as accuracy.

2 Implementation

Based on the course lectures and the provided articles, I selected three methods and obtained different results. The first two methods utilize the baseline and similarity calculations from the provided article "Lessons from the Netflix Prize Challenge", employing **item-item** and **user-user** approaches to predict customers' preferences for unknown movies, thereby facilitating recommendations. The third method incorporates insights from the lecture notes. Given the large scale of the database with many sparse entries, simple linear regression matrix multiplication using gradient descent struggles to learn the parameters effectively. Therefore, we introduced **a hidden layer and combined it with a neural network** to better estimate user preferences (θ).

2.1 Method 1: User-User Collaborative filtering

The recommendation system implemented in this project is based on **user-user collaborative filtering**. This approach predicts a user's preference for an item by leveraging the preferences of similar users. The system operates in three main steps:

1. **Baseline Calulation** The average rating is calculated for each user, ignoring unrated items (ratings of 0). The baseline b_u is defined as:

$$b_u = \frac{\sum_{i \in I_u} r_{ui}}{|I_u|},$$

   where I_u is the set of items rated by user u.r_{ui} is the rating given by user u to item i.$|I_u|$ is the number of items rated by user u.

   This formula computes the average rating for each user, excluding unrated items (which are typically represented by a rating of 0). The baseline b_u represents the central tendency of a user's ratings and is used to adjust for individual rating biases.

2. **Similarity Calculation**: Compute the cosine similarity between users based on their rating patterns. The cosine similarity between two users u and v is calculated as:

$$\text{sim}(u, v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_{uv}} (r_{ui})^2} \cdot \sqrt{\sum_{i \in I_{uv}} (r_{vi})^2}},$$

   where I_{uv} is the set of items rated by both users u and v, and r_{ui} and r_{vi} are the ratings given by users u and v to item i, respectively.

3. **Rating Prediction**: Predict the rating $\hat{r}_{ui}$ that user u would give to item i using the weighted average of ratings from similar users (We use the rating from all users), adjusted by the baseline:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_u} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum_{v \in N_u} \text{sim}(u, v)},$$

where N_u is the set of users similar to user u who have rated item i.

4. **Recommendation Generation**: For each user, recommend the top 5 movies with the highest predicted ratings that they have not yet rated. And here are the results:

```
User-Based Recommendations:
Vincent: ['AVP: Alien vs. Predator', 'Highlander: Endgame', 'The Phantom', 'The Informers', 'Neighbors']
Edgar: ['Edtv', 'Torque', 'Clockwatchers', 'Flyboys', 'Wicked Blood']
Addilyn: ['Space: Above and Beyond', 'Amadeus', 'Sex and the City 2', 'The Tempest', 'Highlander: Endgame']
Marlee: ['The Death and Life of Bobby Z', 'The Ballad of Cable Hogue', 'The Puffy Chair', 'Martha Marcy May Marlene', 'Beloved']
Javier: ['Bad Company', 'The Phantom', 'The Tempest', "The Astronaut's Wife", 'Thirteen']
```

Figure 1: Five movies are recommended to the five users using User-User CF.

2.2 Motivation for Choices

The user-user collaborative filtering approach, combined with baseline estimation, was chosen for the following reasons:

- **Improved Accuracy**: The baseline accounts for systematic biases (e.g., some users tend to rate higher or lower than others), leading to more accurate predictions.
- **Interpretability**: The method is intuitive and easy to explain, as it relies on the idea that users with similar preferences in the past will have similar preferences in the future.
- **No Requirement for Item Features**: Unlike content-based methods, collaborative filtering does not require metadata about items (e.g., genre, director), making it applicable to domains where such information is unavailable.
- **Proven Effectiveness**: Collaborative filtering has been widely used in recommendation systems, including in the Netflix Prize, demonstrating its practical utility.

2.3 Strengths and Weaknesses

2.3.1 Strengths

- **Personalization**: The system provides highly personalized recommendations by leveraging the preferences of similar users.
- **Serendipity**: It can recommend unexpected but relevant items that a user might not have discovered otherwise.
- **Baseline Integration**: The inclusion of a baseline improves prediction accuracy by accounting for user and item biases.
- **Scalability with Sparse Data**: The method performs well even when the user-item interaction matrix is sparse, as it focuses on similarities between users rather than requiring dense data.

2.3.2 Weaknesses

- **Cold Start Problem**: The system struggles to make accurate recommendations for new users or items with few ratings.
- **Sensitivity to Data Sparsity**: If the overlap in rated items between users is minimal, similarity calculations become unreliable.
- **Computational Complexity**: As the number of users grows, the computation of pairwise similarities becomes increasingly expensive.
- **Bias Amplification**: The system may reinforce existing biases in the data, as it relies on historical preferences.

2.4 Method 2: Content-Content Collaborative filtering

The content-based method is fundamentally similar to the user-user approach, utilizing the same similarity and baseline calculations, as well as the rating methods. The only difference is that the content-based method calculates the similarity between movies. It then combines the similarity of a target movie with the ratings of similar movies to infer the user's rating for that movie. Finally, it selects the top five recommendations based on the highest predicted ratings. Here are the results:

```
Item-Based Recommendations:
Vincent: ['The Alamo', 'The Broadway Melody', 'Vessel', 'Indiana Jones and the Temple of Doom', 'Ip Man 3']
Edgar: ['The Game', 'Metropolitan', 'A Passage to India', 'The Nativity Story', 'The Final Destination']
Addilyn: ['Molière', 'Enemy of the State', 'Flash of Genius', 'Top Spin', 'Shooting Fish']
Marlee: ['Superman', 'The Ninth Gate', 'Pitch Black', "Valley of the Heart's Delight", 'Intolerable Cruelty']
Javier: ["Pirates of the Caribbean: At World's End", 'Rebecca', 'The House of the Devil', 'The Juror', 'Boom Town']
```

Figure 2: Five movies are recommended to the five users using Item-Item CF.

2.5 Motivation for Choices

The item-item collaborative filtering approach, combined with baseline estimation, was chosen for the following reasons, in addition to the reasons shared with user-user collaborative filtering (e.g., improved accuracy and interpretability):

- **No Requirement for User Features**: Unlike user-user collaborative filtering, this method does not require detailed user profiles or metadata, making it applicable to domains where user information is limited or unavailable.
- **Proven Effectiveness**: Item-item collaborative filtering has been widely used in recommendation systems, including platforms like Amazon, demonstrating its practical utility and scalability in real-world applications.

2.6 Strengths and Weaknesses

2.6.1 Strengths

- **Personalization**: The system provides highly personalized recommendations by leveraging the similarities between items that a user has rated and those they have not.
- **Serendipity**: It can recommend unexpected but relevant items that a user might not have discovered otherwise, based on the preferences of similar items.
- **Baseline Integration**: The inclusion of a baseline improves prediction accuracy by accounting for user biases, ensuring that predictions are adjusted for individual rating tendencies.

2.6.2 Weaknesses

- **Cold Start Problem**: The system struggles to make accurate recommendations for new items with few or no ratings, as similarity calculations require sufficient rating data.
- **Sensitivity to Data Sparsity**: If the overlap in users who have rated pairs of items is minimal, similarity calculations become unreliable, leading to poor recommendations.
- **Computational Complexity**: As the number of items grows, the computation of pairwise similarities becomes increasingly expensive, especially for large datasets.
- **Bias Amplification**: The system may reinforce existing biases in the data, as it relies on historical preferences and may fail to recommend diverse or novel items.

2.7 Method 3: Content filtering with Neural Network

The recommendation system implemented in this project also includes a neural network-based approach to address the limitations of linear filtering methods. This method leverages the power of neural networks to model complex, non-linear relationships between users and items, which is particularly useful for large and sparse datasets.

2.7.1 Methodology

The neural network-based recommendation system operates as follows:

1. **Neural Network Architecture** : The neural network consists of two layers, activation function is not used in this case:

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1,$$

$$\hat{\mathbf{y}} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2,$$

   where:

   - $\mathbf{x}$ is the input feature vector (flattened movie-genre matrix).
   - $\mathbf{W}_1$ and $\mathbf{b}_1$ are the weights and biases of the first hidden layer.
   - $\mathbf{h}$ is the output of the hidden layer after applying the ReLU activation function.
   - $\mathbf{W}_2$ and $\mathbf{b}_2$ are the weights and biases of the output layer.
   - $\hat{\mathbf{y}}$ is the predicted user-movie ratings.

2. **Loss Function**: The mean squared error (MSE) loss is used to train the model, focusing only on observed ratings:

$$\text{Loss} = \sum_{(u,i)\in\text{Observed}} (r_{ui} - \hat{r}_{ui})^2,$$

   where r_{ui} is the actual rating and $\hat{r}_{ui}$ is the predicted rating.

3. **Training**: The model is trained using the Adam optimizer with L2 regularization to prevent overfitting. The training loop iteratively updates the weights and biases to minimize the loss.

4. **Recommendation Generation**: After training, the model predicts ratings for all user-movie pairs. The top 5 movies with the highest predicted ratings for each user are recommended. Here are the results:

```
Vincent: ['Sugar Hill', 'Dinner for Schmucks', 'Elmer Gantry', 'Metropolitan', 'Dumb and Dumber To']
Edgar: ['Rollerball', 'Pet Sematary', 'Pretty Woman', 'Blade II', 'Multiplicity']
Addilyn: ['Dumb & Dumber', 'Firewall', 'Flushed Away', 'Mi America', 'Jack Reacher']
Marlee: ['Middle of Nowhere', 'Ted 2', 'The Story of Us', 'Pootie Tang', 'The Heart of Me']
Javier: ['Just My Luck', 'Machete', 'House of 1000 Corpses', 'Dear John', 'Once in a Lifetime: The Extraordinary Story of the New York Cosmos']
```

Figure 3: Five movies are recommended to the five users using neural network .

2.8 Motivation for Choices

The method discussed in the lecture employs linear filtering. However, for this large and sparse dataset, **linear filtering struggles to converge**, often resulting in a **high loss** (around several hundreds). In linear filtering, we represent the relationship as $Y = X \cdot \theta$, where Y is the user rating, X represents movie features, and θ denotes user preferences. To improve convergence and performance, we utilize neural networks to model this relationship. By introducing activation functions and hidden layers, we enhance the training of user preferences, allowing for a more effective representation of complex user-item interactions.

2.9 Strengths and Weaknesses

2.9.1 Strengths

- **High Accuracy**: Neural networks can capture complex patterns in the data, leading to more accurate predictions compared to linear methods.
- **Scalability**: The model can handle large datasets efficiently, especially when implemented with modern deep learning frameworks.
- **Flexibility**: The architecture can be easily extended to incorporate additional features or more complex structures (e.g., deep neural networks).

- **Non-Linearity**: The use of activation functions allows the model to learn non-linear relationships, which are often present in real-world data.

2.9.2 Weaknesses

- **Computational Complexity**: Training neural networks is computationally expensive, especially for large datasets and complex architectures.
- **Interpretability**: Neural networks are often considered "black boxes," making it difficult to interpret the learned features and predictions.
- **Overfitting**: Without proper regularization, neural networks can overfit to the training data, leading to poor generalization on unseen data.
- **Hyperparameter Sensitivity**: The performance of neural networks is highly dependent on the choice of hyperparameters (e.g., learning rate, number of layers), which require careful tuning.

3 Discussion

Assessing the quality of a recommendation system before deploying it to users is a challenging task due to several fundamental issues. These challenges stem from the inherent complexity of user behavior, the limitations of evaluation metrics, and the dynamic nature of real-world data. Below, we discuss these challenges and their potential impact on the practical deployment of recommendation systems.

3.1 Fundamental Challenges in Evaluation

- **Lack of Ground Truth**: In recommendation systems, there is no definitive "correct" recommendation for a user. Unlike supervised learning tasks, where the ground truth is explicitly known, user preferences are subjective and can vary over time. This makes it difficult to objectively measure the accuracy of recommendations.
- **Sparse and Implicit Feedback**: User-item interaction data is often sparse, with most users rating only a small fraction of available items. Additionally, feedback is often implicit (e.g., clicks, views) rather than explicit (e.g., ratings), making it harder to infer true user preferences. This sparsity and ambiguity complicate the evaluation process.
- **Dynamic Nature of Data**: User preferences and item popularity are not static; they evolve over time. A model that performs well on historical data may not generalize to future interactions. This temporal shift requires continuous evaluation and model updates, which can be resource-intensive.
- **Evaluation Metrics**: Common evaluation metrics, such as Root Mean Squared Error (RMSE) or precision/recall, may not fully capture user satisfaction or business goals. For example, RMSE focuses on rating prediction accuracy but does not account for the diversity or novelty of recommendations, which are critical for user engagement.
- **Bias and Fairness**: Recommendation systems can inadvertently amplify biases present in the training data, leading to unfair or discriminatory outcomes. Evaluating and mitigating these biases is challenging, as it requires careful consideration of both algorithmic and societal factors.

3.2 Practical Implications

These challenges can lead to several problems in practice:

- **Poor User Experience**: If a recommendation system fails to account for the dynamic nature of user preferences or the sparsity of data, it may provide irrelevant or outdated recommendations, leading to user dissatisfaction.
- **Reduced Engagement**: Over-reliance on historical data or biased recommendations can result in a lack of diversity in suggestions, causing users to disengage from the platform.
- **Resource Intensive Maintenance**: The need for continuous evaluation and model updates to address temporal shifts and evolving user preferences can strain computational and human resources.

- **Ethical Concerns**: Unintended biases in recommendations can have ethical implications, such as reinforcing stereotypes or excluding certain groups of users. Addressing these concerns requires ongoing monitoring and intervention.

4 Code for Implementation

4.1 User-User CF

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Load data
user_reviews = pd.read_csv('user_reviews.csv')
movie_genres = pd.read_csv('movie_genres.csv')
header = pd.read_csv('user_reviews.csv', nrows=0).columns.tolist()[2:]
first_column = user_reviews.iloc[:, 1]
name = first_column.to_numpy()

user_reviews_tensor = user_reviews.iloc[:, 2:].values
movie_genres_cleaned = movie_genres.iloc[:, 2:].values

# User-Based Collaborative Filtering

def user_based_recommendation(user_reviews_tensor, num_recommendations=5):
    # Compute cosine similarity between users
    user_similarity = cosine_similarity(user_reviews_tensor) # Shape: (num_users,
        num_users)

    # Calculate user averages, ignoring zeros
    user_means = np.array([np.mean(user_reviews_tensor[user_idx][user_reviews_tensor[
        user_idx] > 0])
                           for user_idx in range(user_reviews_tensor.shape[0])])

    # Generate recommendations for the first 5 users
    recommendations = {}
    for user_idx in range(5): # First 5 users
        # Find similar users (excluding the user themselves)
        similar_users = np.argsort(-user_similarity[user_idx]) # Sort in descending
            order of similarity
        similar_users = similar_users[similar_users != user_idx] # Exclude self

        # Aggregate ratings from similar users
        user_ratings = user_reviews_tensor[user_idx]
        predicted_ratings = np.zeros_like(user_ratings)
        similarity_sums = np.zeros_like(user_ratings) # To store the sum of
            similarities for normalization

        for similar_user in similar_users:
            predicted_ratings += user_similarity[user_idx, similar_user] *
                user_reviews_tensor[similar_user]
            similarity_sums += user_similarity[user_idx, similar_user] * (
                user_reviews_tensor[similar_user] != 0)

        predicted_ratings = np.divide(
            predicted_ratings, similarity_sums, where=similarity_sums != 0
        ) # Avoid division by zero
```

```
        # Apply baseline adjustment
        predicted_ratings += user_means[user_idx]

        # Mask already rated movies
        unrated_mask = user_ratings == 0
        predicted_ratings = predicted_ratings * unrated_mask # Only consider unrated
            movies

        # Recommend top N movies
        recommended_movie_indices = np.argsort(-predicted_ratings)[:num_recommendations
            ]
        recommendations[name[user_idx]] = [header[idx] for idx in
            recommended_movie_indices]

    return recommendations

user_based_recommendations = user_based_recommendation(user_reviews_tensor)

# Print recommendations
print("User-Based␣Recommendations:")
for user, movies in user_based_recommendations.items():
    print(f"{user}:{movies}")
```

4.2 Item-Item CF

```
#item based:baseline
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Load data
user_reviews = pd.read_csv('user_reviews.csv')
movie_genres = pd.read_csv('movie_genres.csv')
header = user_reviews.columns.tolist()[2:]
first_column = user_reviews.iloc[:, 1]
name = first_column.to_numpy()

user_reviews_cleaned = user_reviews.iloc[:, 2:].values # User-movie ratings
user_reviews_cleaned = user_reviews_cleaned / np.max(user_reviews_cleaned) # Normalize
    ratings
movie_genres_cleaned = movie_genres.iloc[:, 2:].values # Movie-genre matrix

def item_based_recommendation(user_reviews_tensor, num_recommendations=5):
    item_similarity = cosine_similarity(user_reviews_tensor.T)
    recommendations = {}

    # Calculate user averages, ignoring zeros
    user_means = np.array([np.mean(user_reviews_tensor[user_idx][user_reviews_tensor[
        user_idx] > 0])
                           for user_idx in range(user_reviews_tensor.shape[0])])

    for user_idx in range(5):
        user_ratings = user_reviews_tensor[user_idx]

        predicted_ratings = np.zeros_like(user_ratings)
        for movie_idx in range(len(user_ratings)):
            if user_ratings[movie_idx] > 0:
                continue
```

```
            weighted_sum = 0
            sim_sum = 0
            for other_movie_idx in range(len(user_ratings)):
                if user_ratings[other_movie_idx] > 0:
                    # Adjust ratings by subtracting the user's average
                    adjusted_rating = user_ratings[other_movie_idx] - user_means[
                        user_idx]
                    weighted_sum += item_similarity[movie_idx, other_movie_idx] *
                        adjusted_rating
                    sim_sum += item_similarity[movie_idx, other_movie_idx]

            # Apply the baseline adjustment
            if sim_sum > 0:
                predicted_ratings[movie_idx] = weighted_sum / sim_sum + user_means[
                    user_idx]

        unrated_mask = user_ratings == 0
        predicted_ratings = predicted_ratings * unrated_mask # Only consider unrated
            movies

        recommended_movie_indices = np.argsort(-predicted_ratings)[:num_recommendations
            ]
        recommendations[name[user_idx]] = [header[idx] for idx in
            recommended_movie_indices]

    return recommendations

item_based_rec = item_based_recommendation(user_reviews_cleaned)

print("Item-Based␣Recommendations:")
for user, movies in item_based_rec.items():
    print(f"{user}:{movies}")
```

4.3 Neural Network Content Filtering

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim

# Load data
user_reviews = pd.read_csv('user_reviews.csv')
movie_genres = pd.read_csv('movie_genres.csv')

# Preprocess data
user_reviews_cleaned = user_reviews.iloc[:, 2:].values # User-movie ratings
user_reviews_cleaned = user_reviews_cleaned / np.max(user_reviews_cleaned) # Normalize
    ratings

movie_genres_cleaned = movie_genres.iloc[:, 2:].values # Movie-genre matrix

# Convert data to PyTorch tensors
user_reviews_tensor = torch.tensor(user_reviews_cleaned, dtype=torch.float32) # Shape:
    (num_users, num_movies)
movie_genres_tensor = torch.tensor(movie_genres_cleaned, dtype=torch.float32) # Shape:
    (num_movies, num_genres)
```

```
# Dimensions
num_users = user_reviews_tensor.shape[0] # Number of users
num_movies = user_reviews_tensor.shape[1] # Number of movies
num_genres = movie_genres_tensor.shape[1] # Number of genres

# Mask for observed ratings
mask = (user_reviews_tensor > 0).float() # 1 for observed ratings, 0 for missing

# Define the neural network with two hidden layers
class RecommendationNN(nn.Module):
    def __init__(self, input_size, hidden_size1,output_size):
        super(RecommendationNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1) # First hidden layer
        self.fc2 = nn.Linear(hidden_size1, output_size) # Output layer (predict ratings

    def forward(self, x):
        x = self.fc1(x) # Apply ReLU activation for the first hidden layer
        x = self.fc2(x) # Output predicted ratings
        return x

# Hyperparameters
hidden_size1 = 32 # Number of neurons in the first hidden layer

learning_rate = 0.05
reg_param = 0.001 # Regularization parameter
num_epochs = 5000

# Initialize the model
input_size = num_genres * num_movies
output_size = num_users * num_movies
model = RecommendationNN(input_size, hidden_size1,output_size)

# Define the loss function and optimizer
criterion = nn.MSELoss(reduction='sum') # Mean squared error loss
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=reg_param) #
     L2 regularization via weight_decay

# Training loop
for epoch in range(num_epochs):
    # Flatten movie_genres_tensor to match input size
    flattened_input = movie_genres_tensor.reshape(1, -1) # Shape: (1, num_genres *
        num_movies)

    # Forward pass: predict ratings
    predictions = model(flattened_input) # Shape: (1, num_users * num_movies)

    # Reshape predictions to match the shape of user_reviews_tensor
    predictions_reshaped = predictions.view(num_users, num_movies) # Shape: (num_users
        , num_movies)

    # Compute the error only for observed ratings
    error = (user_reviews_tensor - predictions_reshaped) * mask # Shape: (num_users,
        num_movies)

    # Compute the loss (MSE for observed ratings)
    loss = torch.sum(error**2)
```

```
    # Backward pass and weight update
    optimizer.zero_grad() # Clear the gradients
    loss.backward() # Compute gradients
    optimizer.step() # Update weights

    # Print the loss every 50 epochs
    if (epoch + 1) % 50 == 0:
        print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item():.4f}")

    if loss<0.1:
      break
model_path = "recommendation_model.pth"
torch.save(model.state_dict(), model_path)
print(f"Model saved to {model_path}")

# Set the model to evaluation mode
model.eval()
header = user_reviews.columns.tolist()[2:]

first_column = user_reviews.iloc[:, 1]
name = first_column.to_numpy()

# Predict ratings for all users and movies
with torch.no_grad():
    flattened_input = movie_genres_tensor.reshape(1, -1) # Shape: (1, num_genres *
        num_movies)
    predictions = model(flattened_input) # Shape: (1, num_users * num_movies)
    predictions_reshaped = predictions.view(num_users, num_movies) # Shape: (num_users
        , num_movies)


# Recommendations for the first 5 users
recommendations = {}
for user_idx in range(5): # First 5 users
    user_ratings = user_reviews_tensor[user_idx] # Actual ratings by the user
    predicted_ratings = predictions_reshaped[user_idx] # Predicted ratings by the
        model

    # Mask already rated movies
    unrated_mask = user_ratings == 0 # True for movies not rated by the user
    predicted_ratings = predicted_ratings * unrated_mask # Set scores for rated movies
         to 0

    # Get the indices of the top 5 movies
    recommended_movie_indices = torch.argsort(predicted_ratings, descending=True)[:5]

    # Map indices to movie names
    recommended_movies = [header[idx] for idx in recommended_movie_indices]
    recommendations[user_idx] = recommended_movies

# Print recommendations
for user_idx, movie_list in recommendations.items():
    print(f"{name[user_idx]}:{movie_list}")
```