

**University of Swat**

Department of Computer and Software Technology

**Major Assignment**  
  
**Movie Recommendation System**

**Submitted by:**

Muhammad Haseeb, Zaheer Khan, Fazal Umar, Junaid khan, Fawad Ali, Azhar Mehmood

**Submitted To:**

Dr. Muzzamil Khan Saib

**Date:**

24-Dec-2024

**Table of Contents**

[I. Introduction 3](#_Toc185830197)

[A. Objective of the project 3](#_Toc185830198)

[B. Scope of the project 3](#_Toc185830199)

[C. Overview of Recommendation System 3](#_Toc185830200)

[II. Dataset Selection and Preprocessing 5](#_Toc185830201)

[A. Dataset Description 5](#_Toc185830202)

[B. Data Loading and Cleaning 5](#_Toc185830203)

[III. Exploratory Data Analysis 6](#_Toc185830204)

[A. Insights 6](#_Toc185830205)

[i. Insights from rating.csv 6](#_Toc185830206)

[ii. Insights From movies.csv: 7](#_Toc185830207)

[IV. Model Development and Evaluation 9](#_Toc185830208)

[A. Cosine Similarity 9](#_Toc185830209)

[B. Singular Value Decomposition (SVD) in Collaborative Filtering 11](#_Toc185830210)

[C. Neural Collaborative filtering (NCF) 13](#_Toc185830211)

[D. Recurrent Neural Network (RNN): 15](#_Toc185830212)

[V. Challenges encounter 17](#_Toc185830213)

[VI. Conclusion 17](#_Toc185830214)

# Introduction

## Objective of the project

The primary objective of this project was to develop a robust ***movie recommendation system*** that delivers accurate and personalized suggestions by analyzing user preferences and movie attributes. Throughout the process, we applied various machine learning techniques, gaining hands-on experience with tasks such as data preprocessing, exploratory data analysis (EDA), and the implementation of recommendation algorithms.

Our focus was not only on building the system but also on fine-tuning and evaluating its performance. Metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Precision, Recall, and F1-score were used to ensure accuracy and optimize the result of the recommendation system.

## Scope of the project

This project involves various steps such as:

* Selecting and preprocessing a dataset from options like MovieLens, IMDb, or Kaggle.
* Performing EDA to extract meaningful insights.
* Implementing basic (Cosine Similarity, SVD) and advanced models (NCF, RNN).
* Optimizing model performance through hyperparameter tuning.
* Evaluating models using standard metrics and train-test splits.

## Overview of Recommendation System

A **recommendation system** is an artificial intelligence or AI algorithm, usually associated with Machine learning, that uses Big Data to suggest or recommend additional products to consumers. These can be based on various criteria, including past purchases, search history, demographic information, and other factors. Recommender systems are highly useful as they help users discover products and services they might otherwise have not found on their own.

**Types of Recommendation Systems:**

There are a vast number of recommender algorithms and techniques, most fall into these broad categories: collaborative filtering, content filtering and context filtering.

**Content-Based Filtering**  
Content-based filtering suggests items based on their attributes or features. It examines the characteristics of items a user has interacted with and finds similar items. For example, if a user likes movies from a specific genre or with a particular cast, the system recommends movies with similar attributes.

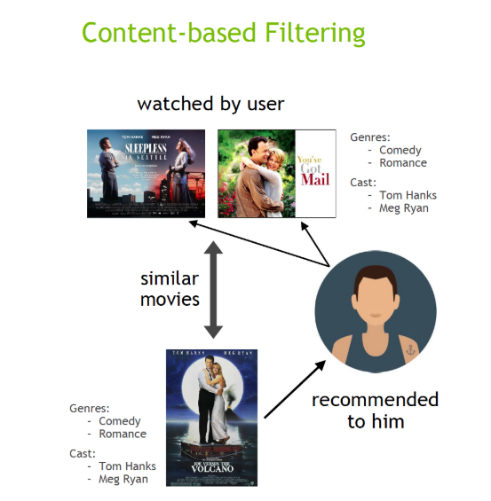
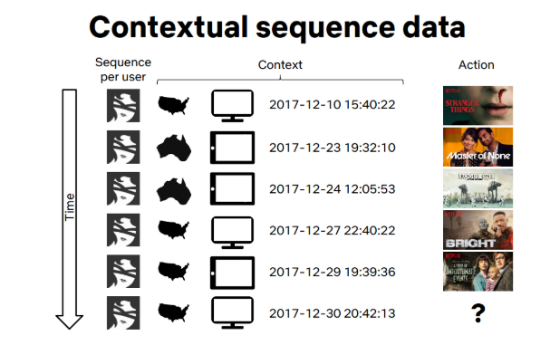
* **Techniques Used in the Project**: Cosine Similarity (CS).

**Collaborative Filtering**  
Collaborative filtering recommends items by analyzing the preferences of many users. It assumes that users who like similar movie’s in the past, will like similar movie’s in the future. This approach builds a model based on past user behavior, such as ratings, purchases, or interactions, and uses this information to predict future preferences.

* **Techniques Used in the Project**: Singular Value Decomposition (SVD) and Neural Collaborative Filtering (NCF).

**Context-Aware Filtering**  
Context-aware systems consider users’ contextual information, such as time, location, or device, to make more relevant recommendations. For example, a system might predict a user's next action based on their previous interactions and current context.

* **Techniques Used in the Project**: Recurrent Neural Network (RNN) is used to model sequences and predict future interactions based on context.





# Dataset Selection and Preprocessing

## Dataset Description

For this project, I chose the **MovieLens 1M dataset**, which is a well-known dataset for recommendation systems. It contains three main files:

1. **movies.dat**: This file provides details about movies with the following information:

* movie\_id: A unique identifier for each movie.
* title: The movie's name.
* genres: Categories or genres associated with the movie.
* **Size**: 3,883 movies are listed in this file.

1. **ratings.dat**: This file contains user ratings for movies with:

* user\_id: A unique identifier for users.
* movie\_id: The ID of the movie being rated.
* rating: The score given by the user to the movie.
* timestamp: The time when the rating was provided.
* **Size**: 100,029 user-movie interactions are recorded here.

1. **users.dat**: This file includes demographic information about the users, such as:

* user\_id: Unique user ID.
* gender: The gender of the user.
* age: The user’s age group.
* occupation: The user’s job type.
* zipcode: The user's location (ZIP code).
* **Size**: 6,040 users are described in this file

## Data Loading and Cleaning

I processed the files step by step to clean and prepare the data for further analysis and modeling:

1. **Loading and Cleaning ratings.dat**:

* I started with the ratings.dat file, checking for missing values, duplicate entries, and incorrect data types. Luckily, no issues were found, so I converted the file into a **CSV format** for easier handling in later stages. CSV files are more versatile, allowing smooth integration with Python libraries like pandas for analysis and modeling.

1. **Loading and Cleaning users.dat**:

For the users.dat file:

* **Mapped Age Groups**: I replaced the numerical age codes with readable age categories (e.g., "18-24" instead of 18).
* **Mapped Occupation Descriptions**: The numerical occupation codes were mapped to meaningful job titles (e.g., "programmer" instead of 12).
* After confirming there were no missing or duplicate values, I saved the processed file as users.csv.

1. **Loading and Cleaning movies.dat**:

The movies.dat file was loaded, and I checked for:

* + **Null values**: None were found.
  + **Duplicate entries**: None were identified.
  + **Data types**: All were consistent.
  + After validating the file, I saved it as movies.csv.

Converting from **.dat** to **.csv** format ensures compatibility with pandas and other libraries, simplifying operations like merging datasets and extracting features.

# Exploratory Data Analysis

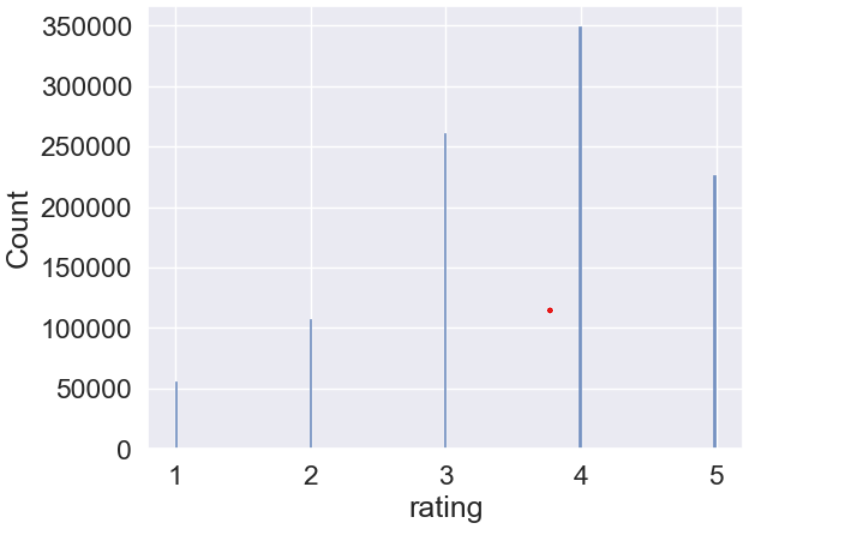
## Insights

### Insights from rating.csv

**Dataset Summary:**

The ratings.csv file contains **1,000,209** entries and **3 columns**: user\_id, movie\_id, and rating. There are **6,040 unique users** and **3,952 unique movies** in the dataset.

**Ratings Distribution:**

* + The mean rating is 3.58, indicating that users tend to rate movies positively on a scale of 1–5.
  + The median rating is 4.0, showing that half of the movies received a rating of 4 or higher.
  + The dataset takes 22.9 MB of memory, which is reasonable for handling in-memory operations during preprocessing.
  + The histogram below illustrates a skew towards higher ratings, with most ratings being 4 and 5, while ratings 1 and 2 are less frequent.

**User Activity:**

* + Each user rated at least 20 movies, with some users rating significantly more.

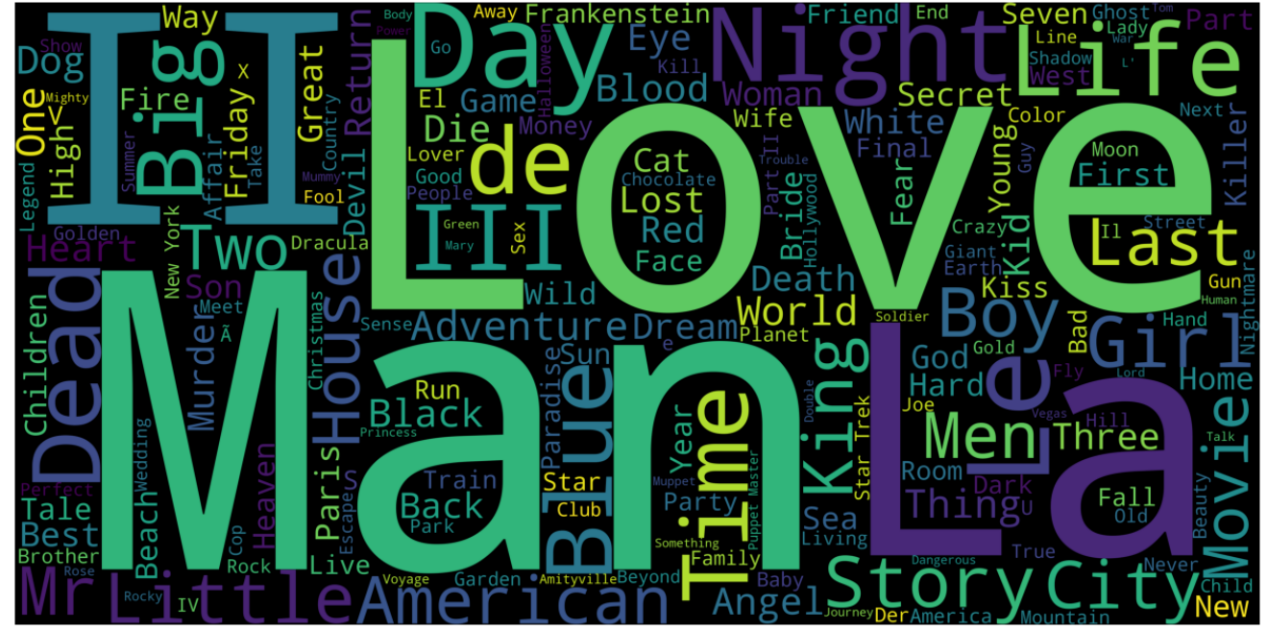
**For example:** User 5 rated 198 movies, and user 6036 rated 888 movies, showing variability in user activity levels.

### Insights From movies.csv:

**Analysis of Movie Titles:**

The below WordCloud from movie titles, highlighting frequently occurring words.

**Common words:** II, III, Day, Love, Life, Night, Man, Dead, and American.

These popular words suggest a concentration of themes around human emotions, time, and cultural representation.

**Genre Popularity:**

Genres play a critical role in movie recommendations based on content similarity. For instance, users who watch a lot of dramas might enjoy recommendations with similar genre labels. In the dataset below is count of most occurring Genres:

* + - Drama: 1,603 occurrences
    - Comedy: 1,200 occurrences
    - Action: 503 occurrences
    - Thriller: 492 occurrences
    - Romance: 471 occurrences

The below WordCloud visualization suggest that the Drama, Comedy, and Action are most occurring in the dataset.

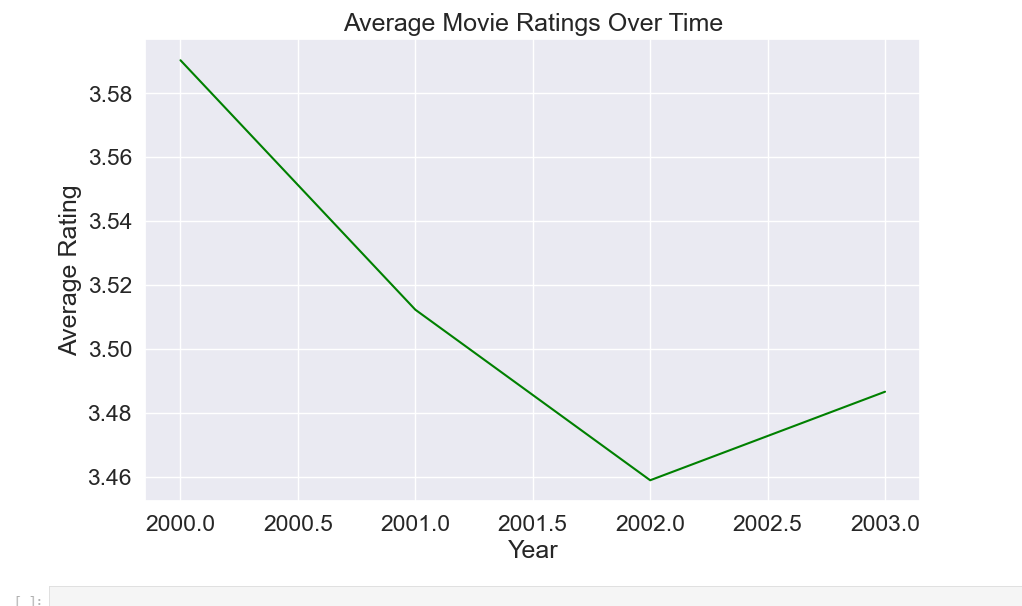


**Trends in Average Ratings Over Time:**

By Analysing the below chart , it can be clearly seen that the rating:

**Decline (2000-2002):** Ratings fall down from 3.58 (2000) to 3.46 (2002). This drop could be attributed to a decline in the quality of movies, stricter rating tendencies among users, or external factors like the post-9/11 impact on cinema themes.

**Recovery in 2003:** ratings showed a slight recovery to 3.48, suggesting improvements in movie quality or shifts in user behavior. These trends highlight how external events and content quality can influence audience perception over time.



# Model Development and Evaluation

## Cosine Similarity

Cosine Similarity is a mathematical technique to measure the similarity between two vectors based on the cosine of the angle between them. In content-based filtering, we utilize this technique to compare the features of items (e.g., genres of movies) and recommend similar items to a user. This approach is particularly effective when the features are represented as text or categorical data.

In this project, Cosine Similarity is implemented as a content-based filtering that is used to recommend movies based on their **genres**. The steps to achieve this are as follows:

1. **Data Preprocessing**:
   * The dataset *movies.csv* was loaded, and the **genres** column (which contains genres separated by "|") was split into lists and converted back to strings. This conversion is essential for text vectorization.
2. **Vectorization Using TF-IDF**:
   * The **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorizer was applied to the **genres** column. This process converts the textual genres into numerical vectors, giving more weight to unique genres while reducing the importance of common genres.
3. **Similarity Matrix Computation**:
   * The Cosine Similarity function was applied to the resulting TF-IDF matrix, producing a **similarity matrix**. Each value in this matrix represents the similarity score between two movies based on their genres. The matrix is stored as a DataFrame for better readability, with movie titles as both row and column indices.
4. **Recommendation Function**:
   * A custom function, recommend\_movies(), retrieves the top n most similar movies for a given movie title. The similarity scores are sorted in descending order, and the top recommendations (excluding the movie itself) are returned.

**Example Execution and Results**

* When a user inputs a movie title, e.g., "Toy Story (1995)", the system computes similarity scores and recommends the top 5 most similar movies based on their genres. Example output:

Top 5 Recommendations for 'Toy Story (1995)' are:

---> Bug's Life, A (1998)

---> Toy Story 2 (1999)

---> Rugrats Movie, The (1998)

---> Chicken Run (2000)

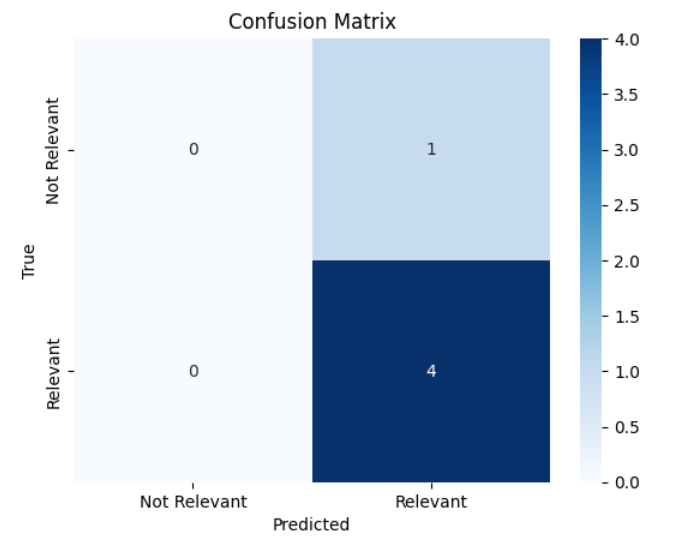
---> The Incredibles (2004)

**Evaluation of Recommendations**

To evaluate the performance of the recommendations, **Precision**, **Recall**, **F1-Score**, and **Accuracy** metrics were calculated using a ground truth relevance list for "Toy Story (1995)". The results are as follows:

* + **Precision:** 0.80
  + **Recall:** 1.00
  + **F1-Score:** 0.89
  + **Accuracy:** 0.80

**Visualization**

To further analyze the results, a heatmap of the confusion matrix was generated. This visualization highlights the true positive and false positive predictions made by the system.

## Singular Value Decomposition (SVD) in Collaborative Filtering

Singular Value Decomposition (SVD) is a matrix factorization technique widely used in collaborative filtering for recommendation systems. It decomposes the user-item interaction matrix into three components:

* U: Captures how much each user aligns with latent features.
* Sigma: Represents the strength of these features.
* V^T: Indicates how much each item aligns with the same latent features.

By leveraging these components, SVD predicts missing ratings, enabling personalized recommendations based on the latent relationships between users and items.

**Steps for implementation**

1. **User-Item Interaction Matrix**:

* The dataset was transformed into a **user-item interaction matrix**, where rows represent users, columns represent movies, and values are ratings. Missing ratings were filled with 0.

1. **Normalization**:

* Since user rating tendencies vary (some users rate generously while others rate strictly), normalization was applied to leverage each user’s ratings by subtracting their mean.

1. **Matrix Factorization**:

* SVD was applied to the normalized matrix R using **scipy.sparse.linalg.svds**. This decomposes R into U, Sigma, and V^T.
* The number of latent features k was adjusted to balance model performance and computational efficiency. Increasing k from 50 to 150 significantly improved the model’s performance by capturing more latent relationships.

1. **Prediction Reconstruction**:

* The predicted ratings were computed by reconstructing the matrix:

R = U⋅Σ⋅V^T + user mean ratings

* The reconstructed matrix provided predicted ratings for all user-movie pairs.

1. **Recommendation Function**:

A custom function was created to recommend movies to a specific user based on the highest predicted ratings for movies they haven’t already rated.

**Example Recommendations**

For User 1310:

* **Top 20 Movies Already Rated:** The function outputs movies with their actual ratings, providing insights into the user’s preferences.
* **Top 20 Predicted Movies:** The system recommends movies that the user might enjoy based on the latent feature alignment.

**Evaluation Metrics**

The model’s performance was evaluated using **Mean Absolute Error (MAE)** and **Root Mean Squared Error (RMSE)**:

* **MAE:** 1.2472
* **RMSE:** 1.5967

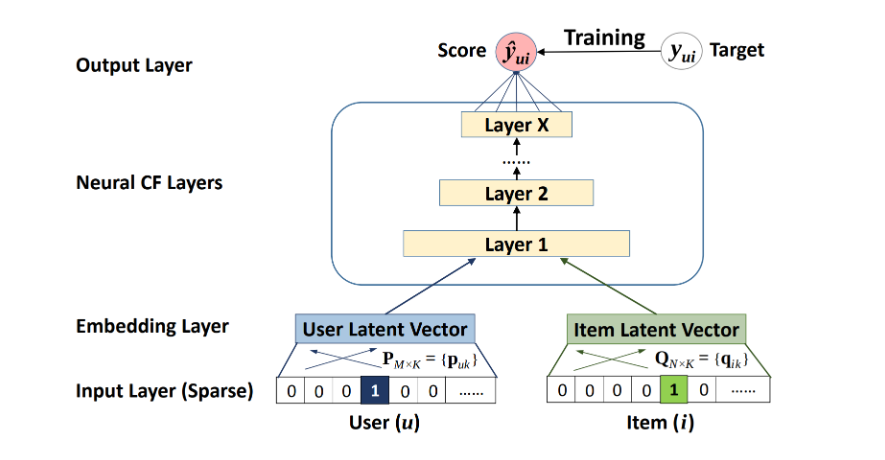
**Observations and Challenges**

* **Sparsity Issue**: The dataset exhibits a high sparsity level (~95.8%), with most user-movie interactions missing. This limits the model’s ability to generalize, as there is insufficient data for many users and movies.
* **Latent Feature Adjustment**: Increasing k (number of latent features) from 50 to 300 reduced MAE and RMSE, demonstrating improved performance. However, further increasing k led to diminishing returns due to the sparsity issue.
* **Low Accuracy**: Despite improvements, the model struggled to predict ratings accurately due to the lack of sufficient data for many user-item pairs.

## Neural Collaborative filtering (NCF)

The Neural Collaborative Filtering (NCF) model is a neural network that provides collaborative filtering based on user and item interactions. The model treats matrix factorization from a non-linearity perspective. NCF TensorFlow takes in a sequence of (user ID, item ID) pairs as inputs, then feeds them separately into a matrix factorization step (where the embeddings are multiplied) and into a multilayer perceptron (MLP) network.

The outputs of the matrix factorization and the MLP network are then combined and fed into a single dense layer that predicts whether the input user is likely to interact with the input item.



**Steps for Implementation:  
1. Data Loading and Preprocessing**:

* + The ratings and movies datasets were loaded using pandas from CSV files.
  + The unnecessary timestamp column was dropped from the ratings dataset.
  + A user-movie interaction matrix was created by pivoting the ratings data, where rows represent users, columns represent movies, and the values correspond to ratings.

1. **Data Encoding**:
   * Both user IDs and movie IDs were encoded into integer values. This step allows the neural network to handle categorical data.
   * The mapping from original user and movie IDs to encoded values was stored in dictionaries for efficient look-up.
2. **Train-Test Split**:
   * The dataset was shuffled and split into training, validation, and test sets. Specifically, 80% of the data was used for training, 10% for validation, and the remaining 10% for testing.
3. **Model Architecture**:
   * The model consists of two embedding layers, one for users and one for movies. Each embedding layer was set to an output dimension of 200, which was chosen after experimenting with different sizes to optimize performance.
   * The embeddings were reshaped and concatenated before passing them through dense layers. The final layer uses a sigmoid activation function to predict ratings between 0 and 1.
4. **Model Training**:
   * The model was trained using Mean Squared Error (MSE) loss function with the Adam optimizer. A batch size of 256 and 6 epochs were used for training, and the training history was plotted to visualize the loss during the training process.
5. **Rating Prediction**:
   * After training, the model was used to predict ratings for unseen user-item pairs in the test set. The predicted ratings were compared to the actual ratings to evaluate the performance.
6. **Movie Recommendations**:
   * A recommendation function was implemented to suggest movies to a specific user. The function predicts ratings for movies that the user has not yet rated and suggests the top 10 movies based on predicted ratings.
7. **Evaluation Metrics**:
   * **Mean Absolute Error (MAE)**: The model achieved an MAE of 0.1597, indicating the average difference between the predicted and actual ratings.
   * **Root Mean Squared Error (RMSE)**: The RMSE was computed as 0.2019, providing an additional measure of the model's prediction accuracy.

**Observations and Challenges**

**Hyperparameter Tuning**: The model's performance was improved by adjusting hyperparameters. Key adjustments included:

* + Increasing the embedding size from 50 to 200.
  + Increasing the batch size from 64 to 256 for more effective model learning.
  + Increasing the number of epochs from 4 to 6 to allow the model to train longer.

**Model Performance**: The model performed well, but further improvements can be made by experimenting with additional hyperparameters or fine-tuning the neural network architecture.

**Example Predictions and Recommendations**

For **User 1**, the model was able to predict ratings for movies the user had not rated yet. The system also recommended the top 10 movies based on these predictions. The model’s predictions were evaluated against actual ratings, and the system displayed both recommended movies and movies the user had already rated highly.

## Recurrent Neural Network (RNN):

Recurrent Neural Networks (RNNs) are a class of neural networks designed specifically for sequential data, where the order of inputs significantly impacts the learning process. Unlike traditional feedforward neural networks, RNNs have the capability to retain information across different time steps, making them suitable for tasks such as natural language processing, time series analysis, and recommendation systems.

The Recurrent Neural Network (RNN) in recommendation system is used to predict movie ratings based on user interactions with the system. The model leverages Long Short-Term Memory (LSTM) layers, which are well-suited for sequential data. By utilizing sequential information, the RNN captures the temporal relationships between movie ratings given by users over time.

In this model, both user and movie embeddings are used to capture latent features of movies and users. These embeddings are trained during the process, allowing the model to learn patterns and relationships in user behavior and movie characteristics. The ratings are normalized to a 0-1 scale, and the model outputs a predicted rating for a given user-movie pair

**Key Steps in the RNN Model:**

1. **Data Loading and Preprocessing:**

* Datasets (users.csv, ratings.csv, movies.csv) are loaded into dataframes and merged to combine relevant information.
* Categorical variables user\_id and movie\_id are encoded using LabelEncoder.
* Ratings are normalized to a range between 0 and 1 using MinMaxScaler.
* The dataset is sorted by user\_id and timestamp to create sequences for RNN modeling.

1. **Sequence Creation for RNN Input:**

* A function create\_sequences creates input sequences for each user. It generates sequences of movie IDs (as input) and the next rating (as the target output). The number of time steps in each sequence is specified as timesteps = 10.

1. **Model Building:**

* **Input layers:** Two inputs are defined for the model:
* movie\_input: The sequence of movies watched by a user (shape = (timesteps,)).
* user\_input: The ID of the user (shape = (1,)).
* **Embedding layers:**
* movie\_embedding: An embedding layer for movies with L2 regularization to capture movie features.
* user\_embedding: An embedding layer for users with L2 regularization to capture user-specific features.
* **LSTM layer:** Processes the movie sequence with LSTM, followed by dropout for regularization.
* **Concatenation:** Combines the LSTM output with the user embedding.
* **Dense layers:** Two fully connected layers with ReLU activation and L2 regularization. Dropout is used for regularization.
* **Output layer:** A dense output layer with sigmoid activation, although the loss function (mse) suggests regression, meaning ratings are predicted in a continuous range.

1. **Model Compilation and Training:**

* The model is compiled with Adam optimizer, using a learning rate of 0.001, mean squared error (mse) as the loss function, and mean absolute error (mae) as a metric.
* Two callbacks are used:
  + EarlyStopping: Stops training if the validation loss doesn’t improve for 5 consecutive epochs.
  + ReduceLROnPlateau: Reduces the learning rate when validation loss plateaus.

1. **Training the Model:**

* The model is trained for 10 epochs with batch size 128, using the training and validation data.

**Evaluation:**

* The **Root Mean Squared Error (RMSE)** is 0.2433, and the **Mean Absolute Error (MAE)** is 0.1932, indicating how well the model’s predictions match the true ratings.

**Hyperparameter Tuning:**

To improve the performance of the model and strike a balance between predictive accuracy and system resources, the following hyperparameters were tuned:

1. **Embedding Size**:
   * The embedding size was tuned between 64 and 128. This represents the dimensionality of the latent features for both users and movies. A larger embedding size allows the model to capture more complex relationships but increases computational demand.
2. **Batch Size**:
   * The batch size was tested between 64 and 128. The batch size defines the number of training samples processed before the model's internal parameters are updated. A larger batch size typically leads to more stable training, but it also requires more memory and computational power.
3. **Epochs**:
   * The number of epochs was adjusted between 5 and 10. An epoch defines one full pass through the entire training dataset. Increasing the number of epochs allows the model to learn more from the data, but it also increases training time and the risk of overfitting. A larger number of epochs generally leads to a more refined model, but system resources may be strained during training, especially when using higher batch sizes or embedding sizes.

**System Resource Considerations:**

Further improvements in MAE and RMSE could be achieved by increasing the embedding size, batch size, and number of epochs. However, these adjustments require more computational power, which caused system lag and failures during training. To avoid this, the current settings were chosen to balance performance and system limitations.

# Challenges encounter

* The first challenge was realizing that the initially selected IMDb dataset lacked sufficient features for implementing advanced algorithms. Consequently, I switched to the MovieLens 1M dataset.
* Another major challenge was the high computational demand of training neural network algorithms. My system often crashed during training, forcing me to adjust hyperparameters to manageable levels. Attempts to use Google Colab were also hindered by slow internet connectivity, limiting progress.

# Conclusion

This project allowed us to design and implement a movie recommendation system using various machine learning algorithms. By employing techniques like Cosine Similarity, SVD, NCF, and RNN, we explored multiple approaches to recommendation systems, addressing both basic and advanced methodologies. Despite challenges like dataset selection and computational resource limitations, we successfully optimized the models within system constraints. This experience enhanced our understanding of data preprocessing, exploratory data analysis, and algorithm evaluation, providing valuable insights for building effective and personalized recommendation systems.