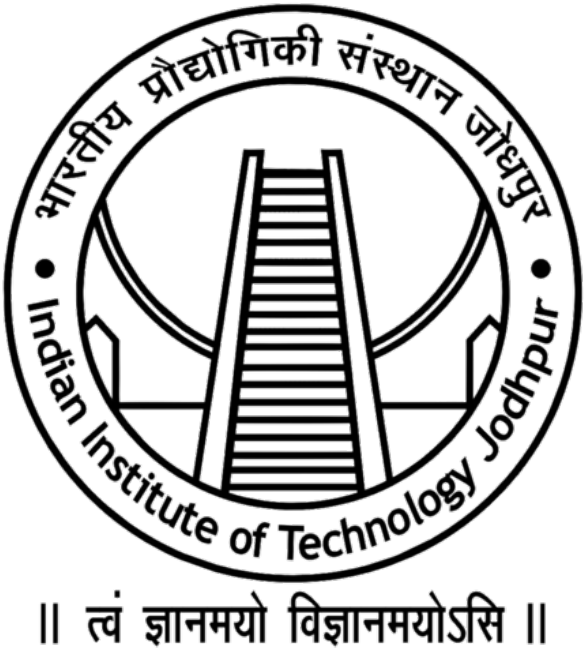
Final Report -Course Project PRML

Movie Recommendation System



Group 19

20th April, 2024

Abstract

Movie recommender systems have become an integral part of modern entertainment platforms, aiding users in discovering content tailored to their preferences. This report presents the design and implementation of a movie recommender system that employs collaborative filtering techniques to generate personalized recommendations. The system utilizes the MovieLens dataset from Kaggle to build a recommendation model, which is then used to suggest movies to users based on their past interactions. The effectiveness of the recommender system is evaluated through evaluation matrices and testing, demonstrating its ability to accurately predict user preferences and enhance the movie-watching experience.

Contents

1 Introduction

Movie recommender systems help users discover movies that align with their tastes and preferences. In this report, we present the design and implementation of a movie recommender system leveraging machine learning techniques. The system is built upon the MovieLens dataset obtained from Kaggle, which comprises a vast collection of movie ratings provided by users.

Our implementation majorly utilizes four distinct machine learning models to generate personalized movie recommendations:

- KNN - K nearest neighbour
- GMM - Gaussian Mixture Model
- SVR - Support Vector Regression
- ANN - Artificial Neural Network

Applied System: We deployed the following ML model which works differently for old users and new users:

1.1 New Users

For the new users using this system, they have two choices and we deployed different models for predictions based on comparison of evaluation matrices.

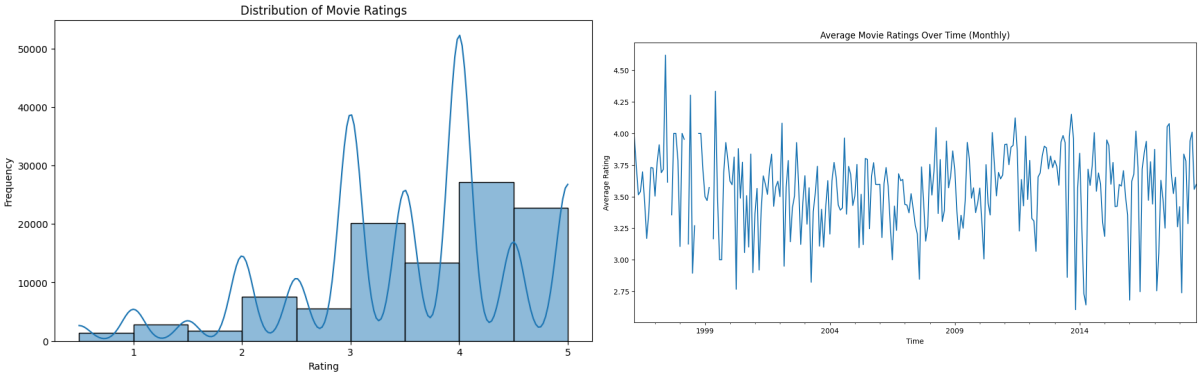
1. **Input: Movie Name** - The system uses KNN model for prediction of the movies to the user which are similar to the input movie of the user.
2. **Input: Genre, rating and feedback** - The system uses ANN model for prediction of the movie based on the feedback mechanism.

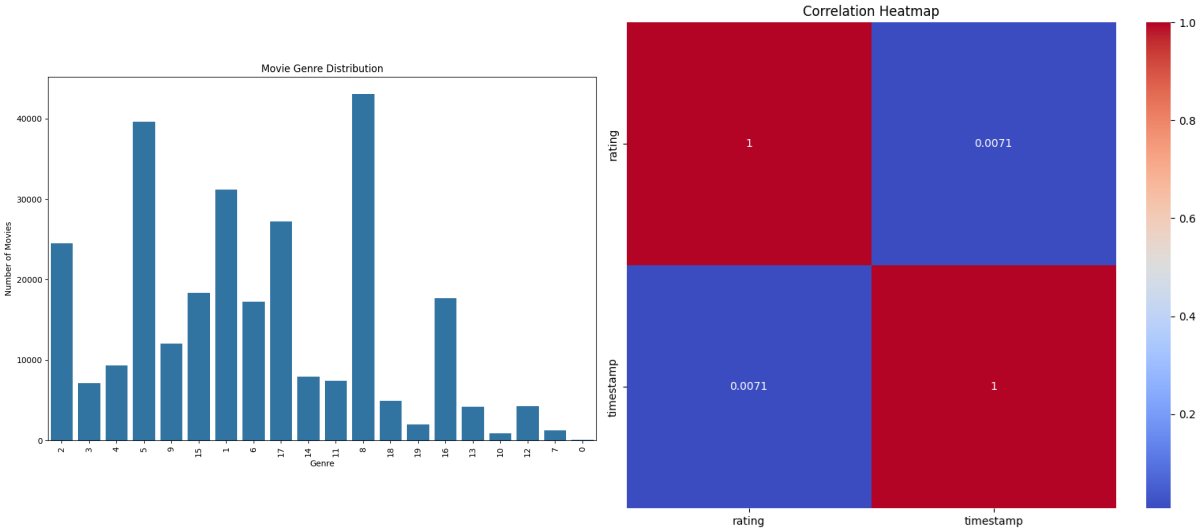
1.2 Old Users

For the old users using this system, they also have two choices and we deployed different models for predictions based on comparison of evaluation matrices.

1. **Input: User ID** - The system uses KNN model alongwith GMM for prediction of the movies to the user using the users database from our dataset.
2. **Input: User ID and rating** - The system uses SVR model for prediction of the movie based on the ratings given by the user.

2 Exploratory Data Analysis





3 Approaches Tried

The following are the models that we tried and implemented the four of these.

3.1 KNN

1. **Data Preparation and Refinement:**

   The initial phase of our project involved merging the ratings and movies datasets. During this process, it became apparent that a considerable number of movies had received zero ratings compared to other ratings. To address this imbalance, we applied a Log Transform to the count of each rating.

2. **Data Description (Continued):**

   Subsequent to the merging of datasets, a further refinement of the data was undertaken to facilitate the development of the recommendation system. The merged dataset was grouped by both 'userId' and 'title' to calculate the average rating given by each user to each movie. This step ensured resolution of any potential duplicate entries where a user might have rated the same movie multiple times by computing the mean of those ratings.

3. **Methodology:**

   The recommendation system was engineered to predict user preferences across a diverse array of movies. To achieve this, the refined dataset underwent transformation into a user-movie matrix. In this matrix:

   - Rows corresponded to individual users.
   - Columns represented movie titles.
   - Entries in the matrix indicated the average rating given by a user to a specific movie.

   Utilizing the `pivot` method, the 'refined_dataset' was transformed, indexing each 'userId' and converting each 'title' into a column. In instances where no rating was available (indicating that a user had not rated a movie), the value was filled with zero. This transformation was pivotal as it aligned the data structure with the prerequisites of most collaborative filtering algorithms, which rely on user-item matrices to function optimally.

   The resulting user-movie matrix exhibited sparsity, with most users having rated only a fraction of the total available movies. Addressing this sparsity emerged as a central challenge in constructing effective recommender systems.

   To efficiently manage the sparsity within the user-movie matrix and optimize computational performance, the matrix was converted into a sparse matrix format using SciPy's `csr_matrix` (Compressed Sparse Row) function. This format, renowned for its efficacy in row-wise matrix operations,

particularly suited the demands of many machine learning algorithms employed in recommendation systems.

This transformation was pivotal for the recommendation system as it:

- Reduced memory consumption by storing only non-zero elements.
- Accelerated arithmetic operations, notably vector and matrix multiplications, which constitute common procedures in recommendation algorithms.

For our Movie Recommender System, we adopted a model-based collaborative filtering approach employing the k-Nearest Neighbors (k-NN) algorithm. Renowned for its simplicity and efficacy in identifying similarities between users or items based on historical interactions, the k-NN model was configured to measure similarity using the cosine metric. This metric, which assesses the cosine of the angle between two vectors, is favored in recommendation systems owing to its adept handling of data sparsity and varying magnitudes of user rating vectors.

4. **Algorithm Configuration:**

   The k-NN model was implemented with the following configuration:

   - **Metric:** Cosine similarity, chosen for its capacity to compare users regardless of the number of ratings they have submitted. This metric focuses on the angle between rating vectors rather than their magnitude, rendering it ideal for comparing similarity in sparse data.
   - **Algorithm:** Brute force, selected for its comprehensive evaluation of distances between all pairs of points in the dataset. Despite its computational intensity, this method ensures that no potential connections are overlooked, a critical consideration in recommendation contexts where accuracy is paramount.

   This k-NN model operated directly on the sparse matrix created from the user-movie ratings, optimizing both memory usage and computational efficiency. The adoption of 'brute' as the algorithm underscored the model's evaluation of similarity between all pairs, ensuring a comprehensive assessment of user similarities.

   The training process involved fitting the k-NN model to the sparse user-movie matrix. As k-NN is a lazy learner, it eschews an explicit "training" phase, instead storing the dataset and performing computations when a prediction is necessitated.

5. **Generating Recommendations:**

   A pivotal aspect of the recommender system entailed identifying users akin to a given user. This similarity underpinned the prediction and suggestion of movies that the given user might appreciate but has yet to view. To facilitate this, the `get_similar_users` function was developed, comprising the subsequent steps:

   (a) **Input Preparation:** The function accepted two inputs: the target `user` (represented by their user index) and the number `n` specifying how many similar users to retrieve. It then accessed the corresponding user vector from the user-movie matrix.

   (b) **k-NN Search:** Leveraging the trained k-NN model, the function computed distances between the target user's vector and all other users' vectors in the dataset. It retrieved the `n+1` nearest neighbors, including the target user themselves.

   (c) **Output Formatting:** The function yielded the indices and distances of the top `n` similar users, excluding the target user to avert self-recommendation. This information was presented in a user-friendly format, showcasing user indices and their corresponding distances from the target user.

6. **Practical Application of Similarity Information:**

   This functionality was instrumental in systems where recommendations hinged not only on the target user's past behavior but also on the behaviors of other users sharing similar tastes. By discerning users with akin preferences, the system could recommend movies endorsed by these similar users, effectively harnessing the collective intelligence of the user base.

7. **Generating Movie Recommendations:**

   Building on the foundation of identifying similar users, the subsequent critical stride entailed translating this similarity into actionable movie recommendations. The `recMovies` function accomplished this by executing the ensuing key operations:

   (a) **Retrieving Similar Users and Distances:** The function commenced by invoking `get_similar_users` to procure a list of users similar to the given `user_id` alongside their respective distances, indicative of similarity levels.

   (b) **Calculating Weights for Similar Users:** To accommodate varying degrees of similarity, weights were computed for each similar user based on their distance. These weights were inversely proportional to the distance, accentuating closer (more similar) users more significantly in the recommendation process.

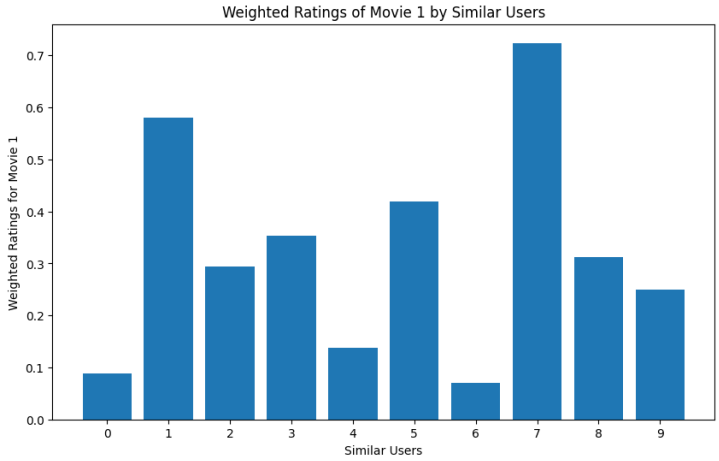
   (c) **Aggregating Ratings of Similar Users:**

   (d) **Compiling Recommendations:** The function subsequently tallied the weighted ratings for each movie, culminating in a mean rating list. Movies were ranked

8. **Practical Application of Similarity Information:**

   This functionality was instrumental in systems where recommendations hinged not only on the target user's past behavior but also on the behaviors of other users sharing similar tastes. By discerning users with akin preferences, the system could recommend movies endorsed by these similar users, effectively harnessing the collective intelligence of the user base.

9. **Generating Movie Recommendations:**



   Building on the foundation of identifying similar users, the subsequent critical stride entailed translating this similarity into actionable movie recommendations. The `recMovies` function accomplished this by executing the ensuing key operations:

   (a) **Retrieving Similar Users and Distances:** The function commenced by invoking `get_similar_users` to procure a list of users similar to the given `user_id` alongside their respective distances, indicative of similarity levels.

   (b) **Calculating Weights for Similar Users:** To accommodate varying degrees of similarity, weights were computed for each similar user based on their distance. These weights were inversely proportional to the distance, accentuating closer (more similar) users more significantly in the recommendation process.

   (c) **Aggregating Ratings of Similar Users:** Ratings of each similar user were retrieved from the user-movie matrix. A fresh rating matrix was constructed by multiplying these ratings by the respective weights, effectively engendering a weighted average of ratings across similar users.

   (d) **Compiling Recommendations:** The function subsequently tallied the weighted ratings for each movie, culminating in a mean rating list. Movies were ranked

3.2 SVR

While Support Vector Machines (SVM) are powerful classification algorithms, other suitable choices might exist for building recommendation systems. Recommendation systems typically involve predicting user preferences for items (movies in the case of MovieLens dataset), which is fundamentally different from a classification task. Here are several reasons why SVMs might not be the best choice for recommendation systems:

Support Vector Regression (SVR) is a variant of Support Vector Machines (SVM) that is specifically tailored for regression tasks. While SVMs are traditionally used for classification, SVR extends the SVM framework to handle continuous target variables, making it suitable for regression problems. Now, let's delve into why SVR could be a suitable choice for recommendation systems:

1. **Nonlinear Relationships:** Recommendation systems often involve complex and nonlinear relationships between users and items. SVR can effectively capture these nonlinear relationships by mapping input features into a high-dimensional feature space using the kernel trick, allowing it to find nonlinear decision boundaries that separate data points.

2. **Robustness to Outliers:** SVR is less sensitive to outliers compared to traditional regression techniques like linear regression. In recommendation systems, outliers may arise from noisy or erroneous user-item interactions. SVR's use of support vectors ensures that only a subset of training samples significantly influences the model, making it robust to outliers.

3. **Flexibility in Kernel Selection:** SVR offers flexibility in choosing different kernel functions, such as linear, polynomial, radial basis function (RBF), or sigmoid kernels. This flexibility allows practitioners to adapt the SVR model to the specific characteristics of the recommendation data, potentially capturing complex patterns and interactions between users and items.

Overall, SVR offers a powerful and flexible framework for building recommendation systems, particularly when dealing with nonlinear relationships, high-dimensional data, and sparse user-item interactions. By leveraging its ability to capture complex patterns and its robustness to outliers, SVR can effectively model user preferences and generate accurate recommendations in various application domains.

Implementing it on Small MovieLens Dataset:

The output from a movie recommender system made using SVR (Support Vector Regression) would typically include the predicted ratings for each user-item interaction in the dataset, as well as evaluation metrics to assess the performance of the recommender system.

- **Predicted Ratings:** For each user-item interaction in the dataset, the recommender system will generate predicted ratings. These ratings represent the model's estimate of how much a user would like a particular item (movie). The predicted ratings could be continuous values or discretized into predefined rating categories.

- **Evaluation Metrics:** After generating the predicted ratings, the recommender system evaluates its performance using various evaluation metrics like Mean Squared Error (MSE),Mean Absolute Error (MAE),R^2 score

- **Visualization:**Wealso visualize the predicted ratings or other relevant insights to provide a better understanding of the recommendations generated by the system. For example, we have created charts to show the distribution of predicted ratings, compare predicted ratings with actual ratings etc.

The time taken for Support Vector Regression (SVR) to train and make predictions depends on various factors, including the size of the dataset, the complexity of the model, and the computational resources available. Here are a few reasons why training and predicting with SVR might take significantly longer for larger datasets:

- **Computational Complexity:** SVR involves solving a quadratic programming problem, which can become computationally expensive as the size of the dataset increases. For larger datasets, the number of support vectors and the complexity of the optimization problem grow, leading to longer training and prediction times.

- **Memory Requirements:** SVR requires storing the kernel matrix, which can be memory-intensive, especially for large datasets. As the dataset size increases, the memory requirements for storing the kernel matrix also increase, potentially leading to memory limitations and slower processing times.
- **Algorithmic Complexity:** The computational complexity of SVR depends on the chosen kernel function and the algorithm used for optimization. Certain kernel functions, such as radial basis function (RBF) kernels, can be more computationally expensive than others, leading to longer training and prediction times.

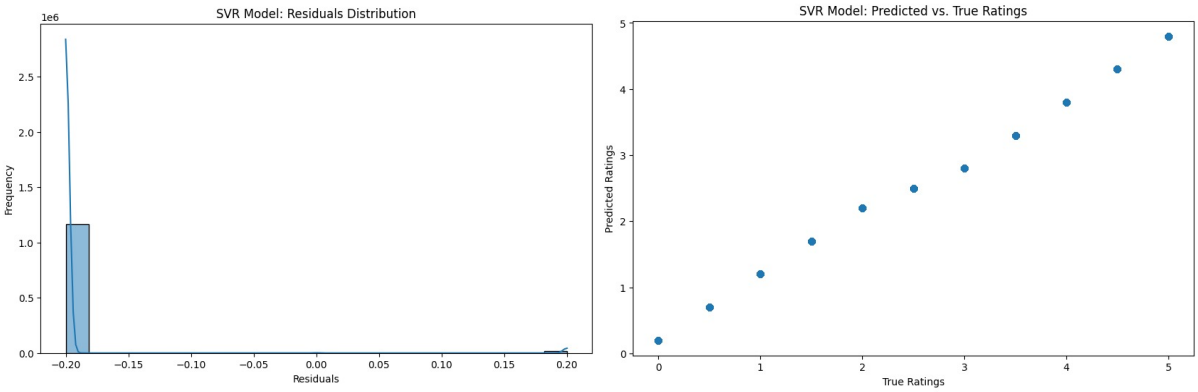
To address the long training and prediction times for larger datasets, we considered the following strategy:-

Implementation of Movie Recommender System using Filtering Techniques and SVR

This section provides a general overview of the implementation steps involved in building a movie recommender system using collaborative filtering, content-based filtering, and Support Vector Regression (SVR).

1. **Data Preprocessing:** The first step in building a movie recommender system is to preprocess the movie dataset. This typically involves the following steps:
   a. Loading the Dataset: Load the movie dataset from a specified source, such as a CSV file or a database.
   b. Handling Missing Data: Identify and handle missing values in the dataset, either by removing rows or columns with missing data or by imputing the missing values using appropriate techniques.
   c. Creating User-Item Matrix: Create a user-item matrix, where rows represent users, columns represent movies, and the corresponding entries are the ratings given by users to those movies.
2. **Collaborative Filtering:** Collaborative filtering is a widely used technique in recommender systems that leverages the rating patterns of users to make recommendations. The implementation of collaborative filtering typically involves the following steps:
   a. User-Based Collaborative Filtering: Train a nearest neighbors model using the user-item matrix to find similar users based on their rating patterns. This approach identifies users with similar preferences and recommends movies that were highly rated by their nearest neighbors.
   b. Item-Based Collaborative Filtering: Train a nearest neighbors model using the transposed user-item matrix to find similar movies based on their rating patterns. This approach identifies movies that are similar to the ones a user has liked in the past and recommends those similar movies.
3. **Content-Based Filtering:** Content-based filtering is another technique used in recommender systems that relies on the characteristics of the items (in this case, movies) to make recommendations. The implementation of content-based filtering typically involves the following steps:
   a. Feature Extraction: Extract relevant features from the movie dataset, such as movie titles, genres, directors, or plot summaries. These features are then transformed into numerical representations suitable for machine learning algorithms.
   b. Topic Modeling: Apply topic modeling techniques, such as Latent Dirichlet Allocation (LDA), to extract latent topics from the movie features. These topics can be used to represent movies in a lower-dimensional space and identify similarities between them.
   c. Profile Building: Build user profiles based on the movies they have rated or watched in the past. These profiles can be used to identify movies similar to the user's preferences.
4. **Support Vector Regression (SVR):** SVR is a machine learning technique used for regression tasks, such as predicting movie ratings. The implementation of SVR in a movie recommender system typically involves the following steps:
   a. Data Preparation: Split the user-item matrix into training and testing sets, flatten the matrices, and prepare the data in the format required by the SVR model.
   b. Model Training: Initialize an SVR model with appropriate parameters, such as the kernel function, regularisation parameter, and epsilon value. Train the model using the prepared training data.
   c. Prediction and Evaluation: Use the trained SVR model to predict the test data. Evaluate the model's performance using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R2 score.

5. **Recommendation Generation:** After implementing the various filtering techniques and the SVR model, the final step is to generate movie recommendations for users. This can be accomplished by combining the predictions from collaborative filtering, content-based filtering, and the SVR model. The recommendations can be ranked based on the predicted ratings or a weighted combination of different techniques.



Code Description:

1. **Dataset:** The MovieLens dataset, obtained from the url , serves as the foundation for this project. The dataset contains movie ratings from various users, along with relevant metadata such as movie titles and user IDs. The data is preprocessed by creating a user-item matrix, where rows represent users, columns represent movies, and the corresponding entries are the ratings users give to those movies. Missing values in the matrix are handled by filling them with zeros.

2. **Methodology:**
   3.1. Data Preparation
   The user-item matrix is split into training and testing sets using the `train_test_split` function from the `sklearn.model_selection` module.
   The matrices are then flattened to accommodate the input format required by the SVR model.

   3.2. Support Vector Regression (SVR) SVR is a powerful machine learning technique that can be used for regression tasks, such as predicting movie ratings. In this implementation, an SVR model is initialized with specific parameters, including the kernel function (rbf), regularization parameter (C=1.0), and epsilon value (epsilon=0.2). The model is trained using the flattened training data, and predictions are made on the flattened test data.

   3.3. Model Evaluation The performance of the SVR model is evaluated using three widely used metrics: Mean Squared Error (MSE): Measures the average squared difference between the predicted and actual ratings. Mean Absolute Error (MAE): Measures the average absolute difference between the predicted and actual ratings. R^2 Score: Indicates the proportion of the variance in the target variable that is explained by the model. These metrics provide insights into the model's accuracy and ability to predict movie ratings effectively.

   3.4 Cross-Validation We started by installing the scikit-surprise library, which is a powerful Python library specifically designed for building and evaluating recommender systems. After importing the necessary classes and functions from the library, we loaded the data from the original DataFrame into a format compatible with Surprise. This involved specifying the rating scale and converting the relevant columns into a Surprise Dataset object. Next, we defined a collaborative filtering algorithm using the KNNBasic class, which implements a basic k-nearest neighbors approach. We chose to use the cosine similarity metric and specified that we wanted to perform user-based collaborative filtering. With the algorithm and data ready, we performed cross-validation to evaluate the algorithm's performance. Cross-validation helps us assess how well the algorithm generalizes to unseen data by repeatedly splitting the data into training and test sets. We set the number of folds to 5 and specified that we wanted to use the Root Mean Squared Error (RMSE) as our evaluation metric. As the cross-validation process ran, we printed progress information to track its execution. Finally, we displayed the cross-validation results, which included the RMSE values

for each fold, as well as summary statistics like the mean and standard deviation across all folds. These results gave us insights into the algorithm's overall predictive accuracy.

3. **Results and Discussion**: The code outputs the computed values of MSE, MAE, and R2 score, allowing for an assessment of the model's performance. A lower MSE and MAE, along with a higher R2 score, indicate better predictive capabilities of the SVR model.

4. **Use Functionality**:

   - We defined a function called `recommend_movies` that takes a user ID, the user-item matrix, movie topics, user profiles, and an optional number of recommendations as input. First, we checked if the given user ID existed in the user-item matrix. If not, we printed a message and returned.
   - Next, we loaded the data into the Surprise format using the `Reader` and `Dataset` classes. We then performed a train-test split on the data, reserving 20 percent for testing. We defined the collaborative filtering algorithm using `KNNBasic` with the cosine similarity metric and user-based approach, and trained it on the training set.
   - To get top recommendations, we first retrieved the movies that the user had already rated and identified the unrated movies. We created a test set with the unrated movies, assigning a temporary rating of 4. We then used the `test` method of the trained algorithm to get predictions for these unrated movies.
   - We sorted the predictions in descending order of their estimated ratings and selected the top `num_recommendations`. We removed any duplicate movie IDs and retrieved the unique movie titles corresponding to these top recommendations from the original DataFrame.
   - Finally, we printed the recommended movie titles for the given user ID. We also included an example usage where we prompted the user to enter a user ID and called the `recommend_movies` function with the appropriate arguments.
   - Through this code, we implemented a recommender system that combines collaborative filtering with the user-item matrix, movie topics, and user profiles to suggest personalized movie recommendations for a given user.

3.3 GMM

Inputs:

1. User Data: The system collects data on users' interactions with movies, including ratings, viewing history, genres they prefer, and any explicit feedback they provide (e.g., likes or dislikes).

2. Movie Data: Information about movies such as genre, director, cast, release year, and plot summaries are essential inputs. This data helps the system understand the characteristics of movies and how they relate to user preferences.

3. Contextual Data (Optional): Additional contextual information such as time of day, location, or device used for accessing the platform can also be considered to tailor recommendations further.

Outputs:

1. Recommended Movies: The primary output of the recommendation system is a list of movies personalized for each user. These recommendations are based on the user's preferences and behaviour as inferred from the input data.

2. Confidence Scores (Optional): Along with recommended movies, the system may provide confidence scores indicating the likelihood of the user enjoying each recommended movie. Higher confidence scores suggest a stronger match between the movie and the user's preferences.

Procedure:
Data Analysis:

1. Data Cleaning and Pre-processing:

   - Loaded datasets: movies, ratings, tags, and links.

- Handled missing values by dropping rows with missing values.
- Converted data types if necessary.
- Removed duplicate entries from all datasets.

2. Exploratory Data Analysis (EDA):
   - Visualized the distribution of ratings using a count plot.
   - Computed the number of ratings per movie and per user.
   - Plotted histograms to visualize the distributions of ratings per movie and per user.
3. Insights:
   - The rating distribution indicates the frequency of each rating value given by users.
   - The histograms show the distribution of the number of ratings per movie and per user, providing insights into the engagement levels of users and popularity levels of movies within the dataset.

Model Training

1. Model Training:
   - Split the ratings data into training and testing sets with a 80-20 ratio.
   - Utilized Gaussian Mixture Models (GMM) for modeling with 10 components.
2. Movie Recommendation:
   - Defined a function to recommend movies to a user based on their ID and the trained GMM model.
   - Identified movies that the user has not yet rated to recommend unseen movies.
   - Scored the unseen movies using the trained GMM model and selected the top-rated ones as recommendations.
3. Recommendations for a Specific User:
   - Generated movie recommendations for a specific user (example user ID: 1).
   - Displayed the top 5 recommended movies for the user along with their titles.
4. Results:
   - Provided personalized movie recommendations for the specified user based on their movie preferences inferred from the ratings data.

Model Evaluation

1. Model Evaluation:
    - Split the ratings data into training and testing sets with a 80-20 ratio.
    - Utilized Gaussian Mixture Models (GMM) for modelling.
2. Hyperparameter Tuning:
    - Defined a parameter grid including the number of components and covariance type for GMM.
    - Conducted grid search cross-validation with 5 folds, optimizing for negative mean squared error.
    - Identified the best model based on the grid search results.
3. Evaluation of Best Model:
    - Made predictions on the test set using the best model obtained from grid search.
    - Calculated the Root Mean Squared Error (RMSE) between the predicted ratings and actual ratings to evaluate model performance.
    - Confidence Score Calculation: The confidence scores represent the maximum probability assigned to each prediction by the model. For each prediction, the maximum probability across all possible rating classes is determined, indicating the model's confidence in its prediction.

3.4 ANN

1. Data Loading and Preprocessing:
    - Imported necessary libraries for data analysis.
    - Loaded the dataset from a CSV file hosted on GitHub into a DataFrame.
    - Dropped the 'timestamp_y' column to streamline the data.
    - Filled missing values in the 'tag' column with "No Tag".
        (a) Data Loading:
            – Imported pandas, matplotlib, seaborn, and scikit-learn libraries.
            – Utilized the `pd.read_csv()` function to load the dataset from a CSV file.
        (b) Data Preprocessing:
            – Removed the 'timestamp_y' column to simplify the dataset.
            – Imputed missing values in the 'tag' column with "No Tag" to handle null values effectively.
2. Grouping and Aggregation:
    - Grouped the data by movie title.
    - Calculated the mean rating, mean timestamp, and first genre for each movie.
    - Reset the index of the grouped DataFrame for better manipulation.
3. Rating Distribution Analysis:
    - Examined the distribution of ratings across movies in the dataset.
    - Counted the number of movies falling into different rating categories, such as equal to 5, below 5, below 4, below 3, and below 2.
    - Provided insights into the distribution of ratings, aiding in understanding the overall rating landscape of the dataset.
4. Data Preparation:
    - Encoded categorical variable 'genres' using LabelEncoder.
    - Scaled numerical feature 'rating' using MinMaxScaler.
        (a) Encoding Categorical Variables:
            – Utilized LabelEncoder to transform categorical 'genres' into numerical labels.

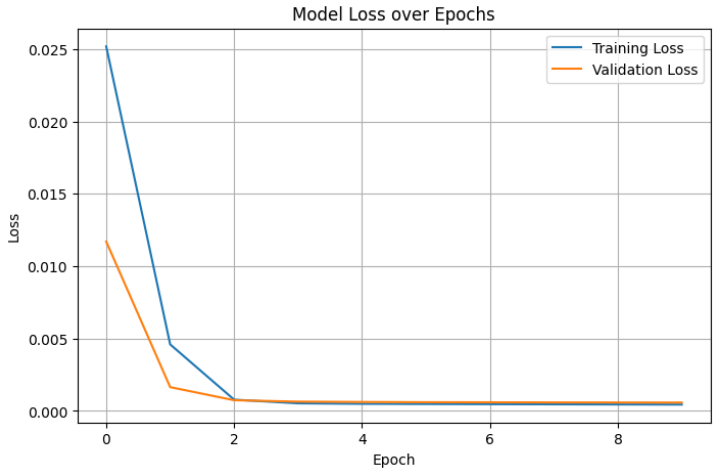
(b) Scaling Numerical Features:
   – Applied MinMaxScaler to scale 'rating' feature to a range between 0 and 1.

5. Model Building:

- Constructed an ANN model for movie recommendation.
- Defined input layers for genre and rating.
- Utilized Embedding layer for genre input and Concatenate layer to merge inputs.
- Comprised Dense layers for learning complex patterns.
- Compiled the model using mean squared error loss and Adam optimizer.

6. Model Training:

- Trained the ANN model on input data.
- Implemented EarlyStopping callback to prevent overfitting.
- Fitted the model on training data with validation split.



7. Recommendation Generation:

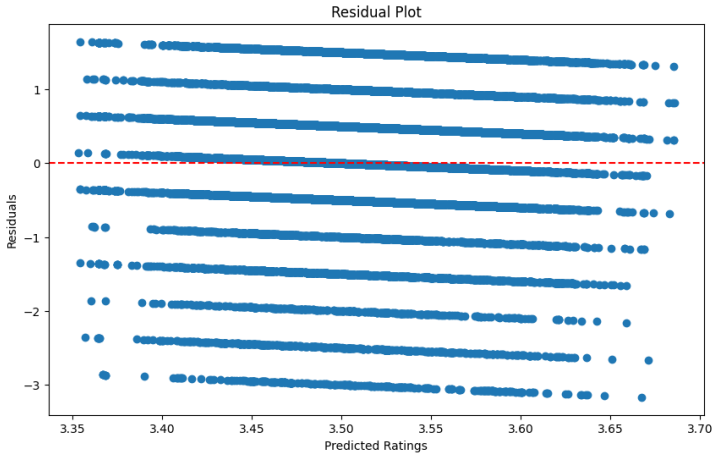
- Employed the trained model to generate movie recommendations.
- Predicted movie ratings based on genre and scaled rating inputs.
- Transformed predictions back to the original rating scale.
- Filtered movies based on user-defined genre and rating range.
- Ranked recommended movies by predicted ratings and extracted top recommendations.

8. User Interaction:

- Interacted with the user to input genre preference and rating range.
- Passed user inputs to the recommendation function to generate personalized movie suggestions.

This movie recommendation system leverages artificial neural networks to provide tailored movie recommendations based on user preferences.

The other ones like linear regression, PCA and decision trees were mentioned in the mid submission report and when implimented upon performed really poor as per the results shown in the next section.



4 Experiments and Results

The following were the evaluation metrics.

Model	**Metrics**
KNN	RMSE: 2.912 MAE: 2.647 MAPE: 75.43%
SVM	MSE: 0.04 MAE: 0.20 R2 Score: 0.836
GMM	RMSE: 3.35 Best Parameters: covariance_type: full, n_components: 5 RMSE Percentage: 74.40% Confidence Scores (in percentage): [91.13075468 97.4685502 97.03453028 ... 99.99366554 95.06600285 96.78346216]
ANN	Precision: 0.8 Recall: 1.0 MAE: 0.37
Linear Regression	RMSE: 1.05 MAE: 0.84 MAPE: 38.21%

5 Contribution of each member

- Arman Gupta(B22CS014): Pre-Processed the Data and worked on building the Model using KNN.
- Soumen Kumar(B22ES006): Worked on building the Model using SVR and made the project description website.
- Sangini Garg(B22CS045): Worked on building the Model using GMM
- Anuj Chincholikar(B22ES018): Worked on building the demo website and deploying it to showcase the Model
- Abhay Kashyap(B22CS001): Worked on building the Model using ANN and shot the Demo Video
- Yogita(B22CS068): Worked on building the Model using ANN and preparing the Report
- Anushka Dhadhich(B22CS097): Worked on building the Model using Linear Regression and created the Report compiling the content