Movie Recommendation System using Machine Learning

Sai Harsha[1] Namruth Tej [2] Manoj Eppa[3] Varun Kumar[4]
Umesh Chandra[5] Shiva Sai[6] Ranjith Chevula[7]

IIT JODHPUR
b22ee083@iitj.ac.in, b22ee039@iitj.ac.in, b22ai018@iitj.ac.in,
b22cs064@iitj.ac.in, b22cs029@iitj.ac.in, b22ee047@iitj.ac.in,
b22ee020@iitj.ac.in,

Abstract

Movie recommender systems are meant to give suggestions to the users based on the features they love the most. A highly performing movie recommendation will suggest movies that match the similarities with the highest degree of performance. This study conducts a systematic literature review on movie recommender systems. It highlights the filtering criteria in the recommender systems, algorithms implemented in movie recommender systems.

In an era inundated with an abundance of digital content, the role of recommendation systems has become indispensable. The purpose of recommendation systems is to cater to the diverse preferences of individuals by sifting through vast repositories of content to curate personalized suggestions. Utilizing artificial intelligence algorithms, these systems analyze an individual's profile, search and browsing history, as well as the behavior of others with similar traits or demographics. By employing sophisticated techniques, recommendation systems aim to present users with a tailored selection of items that are both interesting and relevant to their tastes and preferences.

Keywords: Decision Tree ,Linear Regression,K means clustreing,KNN

Click datasets to access the datasets.

Contents

1 Introduction

The problem at hand is to design and implement a movie recommendation system. The objective is to develop an algorithm capable of suggesting personalized movie recommendations to users based on various factors such as their profile, viewing history, preferences, and similarities to other users.

1.1 Figures

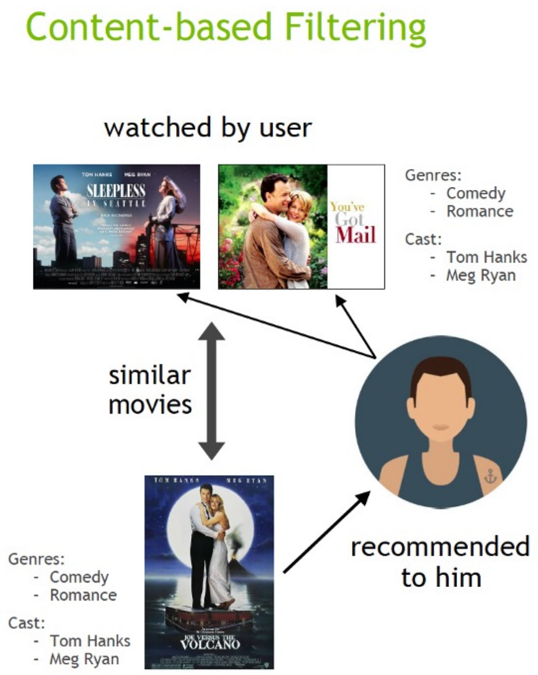


Figure 1: Content-Based Filtering

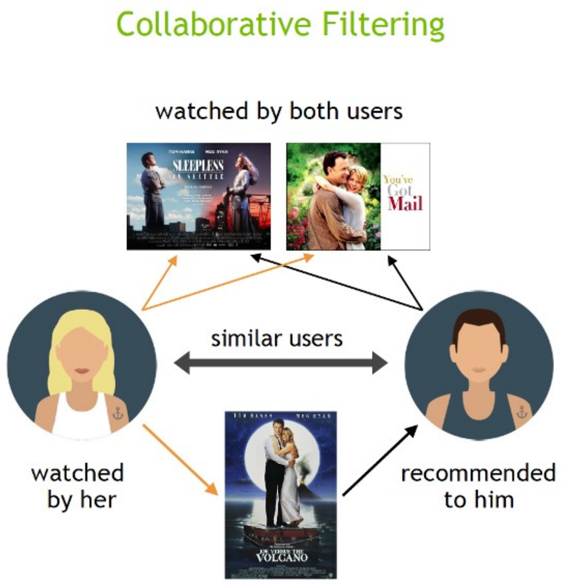


Figure 2: Collaborative Filtering

2 Approaches Tried

We tried five different approaches:

2.1 Using Decision Tree

Here is the detailed explanation of the approach using Decision Tree:

1. **Importing Libraries:** The code imports necessary libraries for data manipulation, model building, and evaluation.

2. **Loading Dataset:** The code loads the MovieLens dataset consisting of two CSV files: ratings.csv containing user ratings and movies.csv containing movie information.

3. **Preprocessing Data:** The code selects relevant columns (userId, movieId, rating) from the ratings DataFrame. It merges the ratings DataFrame with the movies DataFrame based on the movieId column to include movie titles in the data.

4. **Splitting Data:** The preprocessed data is split into training and testing sets using the train_test_split function from sklearn. This is done to evaluate the model's performance on unseen data.

5. **Training Decision Tree Regressor:** Features (userId and movieId) and the target variable (rating) are separated from the training data. A decision tree regressor model is initialized and trained using the training data.

6. **Making Predictions on Test Set:** The trained model is used to make predictions on the test set. Predictions are made for the ratings of movies in the test set.

7. **Evaluating the Model:** Mean squared error (MSE) is calculated to evaluate the performance of the model. MSE measures the average squared difference between the actual and predicted ratings.

8. **Movie Recommendations for a User:** A specific user ID is chosen for whom movie recommendations will be generated. Movies already rated by the user are identified. A list of movies that the user hasn't rated yet is created.

9. **Making Predictions for Unrated Movies:** The trained model is used to predict ratings for the movies not yet rated by the user.

10. **Sorting Recommended Movies:** Recommended movies are sorted based on their predicted ratings in descending order.

11. **Printing Top Recommendations:** The top recommended movies along with their predicted ratings are printed for the specified user.

2.2 KNN using Content Based Filtering

Here is the detailed explanation of the approach using KNN with Content Based Filtering:

1. **Importing Libraries:** The code imports necessary libraries such as `pandas`, `numpy`, `requests`, and modules from `sklearn`.

2. **Loading Datasets:** The code loads various datasets (`ratings.csv`, `movies.csv`, `tags.csv`, `links.csv`) using the `pd.read_csv` function.

3. **Data Preprocessing:** The code preprocesses the loaded datasets by dropping unnecessary columns (`timestamp`) and modifying the structure of the data to suit further analysis.

4. **Combining Tags and Genres:** The code combines the `tag` and `genres` columns into a single `tags` column to provide richer metadata for movie recommendations.

5. **Feature Extraction:** The code utilizes `CountVectorizer` from `sklearn.feature_extraction.text` to convert text data (`tags`) into numerical features.

6. **Calculating Similarity:** Cosine similarity is calculated using `cosine_similarity` from `sklearn.metrics. pairwise to measure the similarity between movies based on their tags.`

7. **Movie Poster Retrieval:** The `get_tmdb_poster` function utilizes the TMDb API to retrieve movie posters based on their TMDb IDs.

8. **Recommendation Function:** The `recommend` function generates movie recommendations for a given user ID. It filters high-rated movies (ratings >= 3) from the user's watched list, then selects a random high-rated movie or a random movie if no high-rated ones are available. It finds similar movies based on the selected movie using cosine similarity scores. It retrieves TMDb IDs for similar movies from the links dataset and fetches poster URLs using the TMDb API. Finally, it prints the recommended movie titles and their corresponding TMDb poster URLs.

2.3 Using K Means Clustering

Here is the detailed explanation of the approach using K Means Clustering:

1. **Importing Libraries:** This section imports necessary libraries such as `pandas`,`numpy`,`matplotlib.pyplot`,and modules from `sklearn`.

2. **Loading Datasets:** The code loads two datasets, `movies.csv` and `ratings.csv`, using the `pd.read_csv` function.

3. **Calculating Dataset Lengths:** It calculates the total number of movies and ratings in the datasets.

4. **Preprocessing:** It drops the unnecessary `timestamp` column from the `ratings_dataset`. Then, it merges the `ratings_dataset` and `movies_dataset` based on the `movieId` column.

5. **Converting Genres into Different Columns:** It splits the `genres` column into separate columns based on the pipe delimiter (—) and assigns a binary value (0 or 1) to each genre.

6. **Assigning 1 to All Columns Present in the Genres:** It assigns a value of 1 to all columns in the `new_dataset` DataFrame that are present in the `genres` column.

7. **Dropping Unnecessary Columns:** It drops the `genres` and `title` columns from the `new_dataset` as they are no longer needed.

8. **Handling Null Values:** It drops any rows containing null values from the `new_dataset`.

9. **Clustering Using K-Means:** It uses K-Means clustering algorithm to cluster the movies into 9 different clusters based on their features.

10. **Saving the Model:** It saves the trained K-Means model using `pickle` for future use.

11. **Creating a New DataFrame with Unique Clusters for Each Movie:** It creates a new DataFrame `new_data_frame` where each movie is assigned to a single cluster based on the majority cluster assigned to it.

12. **Merging DataFrames:** It merges the `new_data_frame` with the original `movies` dataset to get additional movie information.

13. **Filling Missing Values:** It fills any missing values in the DataFrame with a random integer between 0 and 9.

14. **User Input and Recommendation:** The function `select_cluster()` prompts the user to select movies they would like to watch and assigns a cluster based on the user's choice. The `main()` function recommends movies from the selected cluster to the user and allows them to choose whether they like the recommendations or want more.

2.4 KNN using Collaborative Filtering

Here is the detailed explanation of the approach using KNN with Collaborative Filtering:

1. **Importing Libraries and Loading Datasets:** The code begins by importing necessary libraries such as Pandas and NumPy. It then loads several datasets related to movie ratings, tags, and movie details from provided URLs using the pd.read_csv function. The tags dataset contains user-generated tags for movies, ratings contains user ratings for movies, links contains links to external movie databases, and movies contains general information about movies.

2. **Processing Data:** Unique user IDs and movie IDs are extracted from the tags and ratings datasets. A new DataFrame named New is created to store information about users, movies watched, and corresponding ratings. Additional movie information such as IMDb and TMDb IDs is merged into the DataFrame. Genre information is extracted and stored in a dictionary.

3. **Creating a DataFrame for Movie Details:** A DataFrame named Data is created to store movie details such as ID, title, IMDb ID, TMDb ID, and genre information. This DataFrame is constructed by extracting relevant columns from the movies and links datasets.

4. **Merging DataFrames:** The movie details DataFrame Data is merged with the ratings DataFrame to create a merged_dataset. The merged_dataset is further refined by grouping it by user ID and movie title, and aggregating the mean rating.

5. **Creating User-Movie Sparse Matrix:** The refined dataset is pivoted to create a user-movie matrix where rows represent users, columns represent movies, and values represent ratings. This user-movie matrix is converted to a sparse matrix using csr_matrix for efficient storage and computation.

6. **KNN Model Training:** A K-Nearest Neighbors (KNN) model is initialized with cosine similarity metric and brute-force algorithm. The KNN model is fitted with the user-movie sparse matrix.

7. **Model Function:** A function named model is defined to recommend movies to a given user based on similar users. It calculates similar users using KNN, computes weighted ratings, and recommends movies not seen by the user but seen by similar users.

8. **Saving the Model:** The model function is saved using pickle for future use.

9. **Loading and Using the Model:** The saved model is loaded from the pickle file. User input is taken to specify a user ID, and the model recommends movies for that user.

2.5 Using Linear Regression

This subsection covers the details of implementing a movie recommendation system using the Linear Regression algorithm.

Importing Libraries

1. **Pandas:** Used for data manipulation and analysis.

2. **NumPy:** Provides support for large, multi-dimensional arrays and matrices.

3. **Sklearn:** Tools for data mining and data analysis.

4. **Matplotlib:** Library for creating static, interactive, and animated visualizations in Python.

5. **Scipy:** Used for scientific computing and technical computing.

Function Definitions

load data Downloads two CSV files containing movies and ratings data. Returns the ratings and movies DataFrames.

prepare data Takes ratings DataFrame as input. Utilizes OneHotEncoder from Scikit-learn to encode userId and movieId which converts categorical variables into a form that could be provided to ML algorithms to do a better job in prediction. Creates a sparse matrix, X, that combines the encoded userId and movieId. Extracts the ratings as y to use as the target variable. Returns the matrix X, the target variable y, and the encoders for later use.

train model Receives the training data (X_train and y_train). Initializes and trains a LinearRegression model. Returns the trained model.

evaluate model Uses the trained model to predict ratings based on the test set (X test). Computes the Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared values to evaluate the model's performance. Displays the evaluation metrics. Returns the actual and predicted ratings for further analysis.

make recommendations Takes a user ID and generates recommendations. Encodes the user ID and movie IDs for all movies. Combines these encodings to predict ratings for all movies as if they were rated by the specified user. Filters out movies with a predicted rating above a specified threshold (e.g., 4.0). Merges the results with the movies DataFrame to display titles alongside predicted ratings.

plot results Plots two graphs: Residuals vs Predicted: Helps in understanding the variance of prediction errors. Actual vs. Predicted: Visualizes the accuracy of predictions. Both plots help to visually assess the performance and accuracy of the regression model.

Main Function main Coordinates the execution of the program. Loads data, prepares it, splits into training and test datasets, trains the model, and evaluates it. Calls make_recommendations to show movie recommendations for a specified user. Executes the plot_results function to display graphical analysis of model predictions.

Execution Trigger The `if __name__ == "__main__":` block ensures the main function runs only if the script is executed as the main module, allowing the script to be imported without running the main process.

Conclusion This script demonstrates an effective use of linear regression to build a recommendation system. By encoding user and movie IDs, the model is capable of learning to predict ratings based on past data. The recommendations are then generated by predicting high ratings for movies not yet watched by a given user, and the system's effectiveness is visually assessed through plots. This implementation provides a foundation that can be extended or improved with more complex algorithms and features, such as collaborative filtering or matrix factorization techniques for better recommendation accuracy.

3 Experiments and Results

3.1 DataSet

We will use a Movie Lens Small Latest Dataset which is available in Kaggle. This dataset can be accessed from this Link. This dataset contains four csv files:

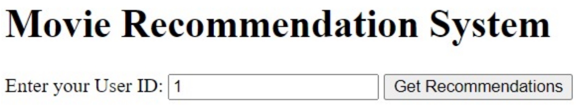
1. **links.csv:** This file acts like a translator between different movie identification systems. It contains three columns:
   - `movieId`: A unique identifier for a movie within the MovieLens dataset.
   - `imdbId`: The movie's ID on the Internet Movie Database (IMDb) website.
   - `tmdbId`: The movie's ID on The Movie Database (TMDB) website.

2. **movies.csv:** This file holds the core information about the movies themselves. It contains three columns:
   - `movieId`: The same unique movie identifier from the links.csv file.
   - `title`: The full title of the movie.
   - `genres`: genre of the movie. This data allows us to identify movies and potentially categorise them based on genre.
3. **ratings.csv:** This file captures user interactions with the movies. It contains three main columns and one additional timestamp column:
   - `userId`: A unique identifier for a user within the MovieLens dataset.
   - `movieId`: The movie identifier again, linking back to the movies.csv file.
   - `rating`: The rating a user gave to the specific movie (scale of 1 to 5).
   - `timestamp`: seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
4. **tags.csv:** This file captures additional user input on movies. It contains three main columns and one timestamp column:
   - `userId`: The same user identifier from the ratings.csv file.
   - `movieId`: The movie identifier again, linking back to the movies.csv file.
   - `tag`: A keyword user assigned to describe the movie.
   - `timestamp`: seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

Result

The analysis of the **linear regression model's** performance on the movie recommendation dataset yields the following metrics: **Mean squared error (MSE) is 0.80**, **Mean absolute error (MAE) is 0.68**, and the **R-squared value is 0.27**. Upon visual examination of the residual plots, a pattern of heteroscedasticity is observed, suggesting that the variance of the residuals is not constant across predictions. Additionally, the actual versus predicted plot shows a banding effect indicative of the model's inability to capture the discrete nature of user ratings. These results suggest that linear regression may not be the ideal choice for building a movie recommendation system. The model does not adequately capture the complex patterns in user preferences. Recommendation systems typically benefit from specialized models like collaborative filtering or matrix factorization, which are more adept at discerning the intricate user-item interactions inherent in this type of data. **decision tree regressor Mean Squared Error: 1.65**

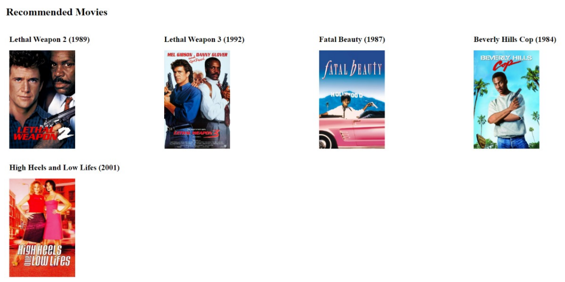
The graphs show that linear regression isn't the best choice for recommending movies. It's not capturing the patterns of what users like well enough. Models that are specially made for suggesting movies, like collaborative filtering or matrix factorization, usually work better for this kind of job.



Movie Recommendation System

Enter your User ID: 1

Figure 3: Input



Recommended Movies

Lethal Weapon 2 (1989)

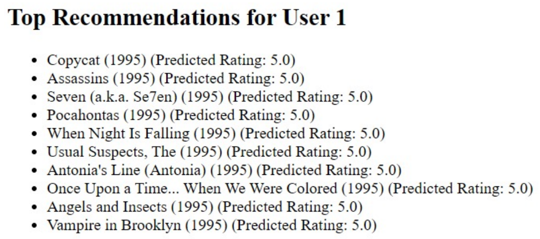
Lethal Weapon 3 (1992)

Fatal Beauty (1987)

Beverly Hills Cop (1984)

High Heels and Low Lifes (2001)

Figure 4: Output (KNN based on content similarity)



Top Recommendations for User 1

- Copycat (1995) (Predicted Rating: 5.0)
- Assassins (1995) (Predicted Rating: 5.0)
- Seven (a.k.a. Se7en) (1995) (Predicted Rating: 5.0)
- Pocahontas (1995) (Predicted Rating: 5.0)
- When Night Is Falling (1995) (Predicted Rating: 5.0)
- Usual Suspects, The (1995) (Predicted Rating: 5.0)
- Antonia's Line (Antonia) (1995) (Predicted Rating: 5.0)
- Once Upon a Time... When We Were Colored (1995) (Predicted Rating: 5.0)
- Angels and Insects (1995) (Predicted Rating: 5.0)
- Vampire in Brooklyn (1995) (Predicted Rating: 5.0)

Figure 5: Output (Decision Tree)

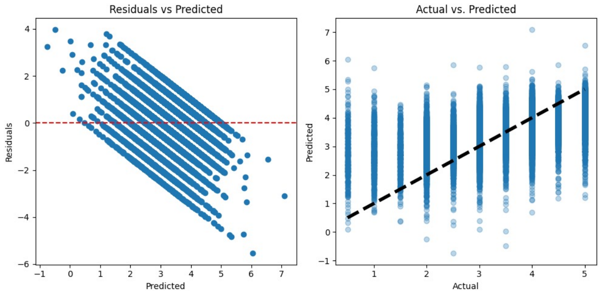


Figure 6: PLot using Linear Regression Model

4 Summary

The detailed approaches for building different types of movie recommendation systems using various machine learning techniques—Decision Tree, KNN with Content Based Filtering, K Means Clustering, KNN with Collaborative Filtering, and Linear Regression—are summarized below:

Decision Tree: This approach involves importing libraries, loading and preprocessing the MovieLens dataset to merge ratings and movie details, splitting data into training and testing sets, training a Decision Tree Regressor on user and movie IDs, making predictions on the test set, evaluating performance using MSE, and recommending movies by predicting ratings for unrated movies by a specific user.

KNN with Content Based Filtering: This method imports essential libraries and dataset files, preprocesses data by combining genres and tags, uses text vectorization for feature extraction, calculates movie similarities using cosine similarity, and recommends movies by selecting similar ones based on a high-rated movie watched by the user. Additionally, it retrieves movie posters from an external API based on the recommendations.

K Means Clustering: This strategy involves importing libraries, loading datasets, preprocessing data by converting genres into separate binary columns, dropping unnecessary columns, and handling null values. It clusters movies using the K Means algorithm into different clusters and recommends movies based on the user's selected cluster.

KNN with Collaborative Filtering: This method starts by loading necessary libraries and datasets, creating a user-movie matrix, training a KNN model to find similar users, and recommending movies based on the preferences of these similar users. It also involves saving and loading the model for future use.

Linear Regression: This technique uses libraries like Pandas, NumPy, Sklearn, and others to load and preprocess data including one-hot encoding of user and movie IDs. A Linear Regression model is trained and evaluated using metrics like MSE, MAE, and R-squared. Recommendations are generated by predicting ratings for movies not yet seen by the user and filtering those with ratings above a certain threshold. Each approach utilizes distinct algorithms and processes to provide personalized movie recommendations, demonstrating the diversity in methods available for recommendation systems in machine learning

References

Source for Figures in 1.1

CampusX

Towards Data Science

F. Maxwell Harper and Joseph A. Konstan. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19, 2015

MOVIE RECOMMENDER SYSTEM USING DECISION TREE METHOD Muhammad Bilal Rafif Azaki1), Z. K. A. Baizal*2

chatGPT for getting some HTML and CSS code for making project page

For making the webpage, we took suggestions from 4th year Students of IIT J.

5 Contribution of each member

1. **Sai Harsha**: Implemented using K Nearest Neighbors algorithm using content based filtering. Prepared web page for this model, mid report. Also, lead of the project.

2. **Umesh Chandra**: Implemented using Decision tree algorithm. Prepared web page for the project.

3. **Manoj Eppa**: Implemented using K-means clustering algorithm and Developed Movie Recommendation website for K-Means Clustering.

4. **Varun Kumar**: Implemented using K-means clustering algorithm.Movie Recommendation website, project page, and presentation.

5. **Namruth Tej**: Implemented using Linear RegressionModel,Prepared Report.

6. **Ranjith Chevula** : Implemented using K Nearest Neighbors algorithm using collaborative based filtering.

7. **Shiva Sai**: Movie Recommendation website and presentation.