

Student Name: MONISHA J

Register Number: 510623104068

Institution: C. ABDUL HAKEEM COLLEGE OF
ENGINEERING AND TECHNOLOGY

Department: COMPUTER SCIENCE AND ENGINEERING

Date of Submission: 16-05-2025

Github Repository Link:

https://github.com/Monisha45228/movie_recomm-phase-3.git

1. Problem Statement

Users are often overwhelmed by the sheer volume of movie choices available and find current recommendation systems inadequate in surfacing truly relevant and enjoyable content. This disconnect stems from a lack of deep personalization that considers the multifaceted nature of individual preferences and viewing habits. The problem is to bridge this gap by developing an AI-driven matchmaking system that can accurately and intuitively connect users with movies they will love, enhancing their entertainment experience and fostering a stronger sense of discovery

2. Abstract

The challenge of effectively recommending movies that truly resonate with individual user tastes persists despite the prevalence of existing recommendation systems. This abstract presents an AI-driven matchmaking system as a solution, addressing the limitations of conventional approaches by employing advanced machine learning to analyse complex user-movie relationships. The proposed system constructs personalized profiles that capture evolving preferences and contextual factors, enabling the delivery of more accurate, relevant, and serendipitous movie recommendations. This paper will detail the methodology

behind this intelligent matchmaking process and discuss its potential to significantly improve user satisfaction and content discovery within the vast landscape of cinematic offerings.

3. System Requirements

○ Hardware:

- ☐ **CPU:** Intel i7 / AMD Ryzen 7 or higher (8+ cores recommended)
- ☐ **RAM:** 16 GB minimum (32 GB recommended for large datasets)
- ☐ **Storage:** 500 GB SSD or more
- ☐ **GPU:** NVIDIA RTX 3060 or higher (CUDA support required for deep learning)

○ Software:

- ☐ **OS:** Ubuntu 20.04+ (preferred), or Windows 10 Pro / macOS Ventura+
- ☐ **Python:** Version 3.8 or later
- ☐ **Deep Learning Libraries:** PyTorch or TensorFlow (with GPU support via CUDA/cuDNN)
- ☐ **NLP Libraries:** transformers, sentence-transformers (for movie understanding)
- ☐ **Data Handling:** pandas, numpy, SQLAlchemy, psycopg2
- ☐ **Database:** PostgreSQL or MongoDB
- ☐ **Dev Tools:** Docker, Git, Jupyter, MLflow (for experiment tracking)

4. Objectives

- Deliver highly personalized movie recommendations using an AI-driven matchmaking system.

- Match users with movies based on emotional tone, viewing behavior, and preferences—not just genre or popularity

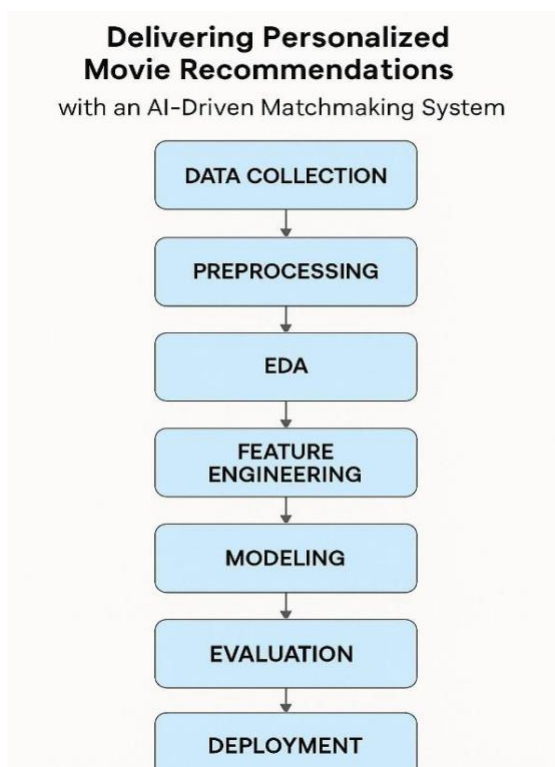
Expected Outputs & Predictions

- Tailored movie suggestions per user profile.
- Prediction of user satisfaction and engagement.
- Adaptive recommendations that learn over time.
- Insights into user preferences and viewing patterns.

Business Impact

- Boosts user retention and engagement.
- Increases content watch-through rates.
- Enhances discovery of niche content.
- Enables smarter content and marketing decisions.

5. Flowchart of Project Workflow



6. Dataset Description

Several publicly available datasets are commonly used for developing and evaluating movie recommendation systems:

- **MovieLens Datasets (GroupLens):** These are widely used and come in various sizes (e.g., 100K, 1M, 10M, 20M, 25M ratings). They typically include user IDs, movie IDs, ratings, timestamps, and sometimes movie metadata (titles, genres). Some versions also include user demographics and tag information.
- **IMDb Datasets:** A large dataset containing information about movies, TV series, cast, crew, ratings, and user reviews. This can be used to extract rich movie metadata and user opinions.
- **Netflix Prize Dataset:** A dataset of over 100 million ratings from Netflix users on a large number of movies. While older, it remains a benchmark for recommendation algorithms.
- **TMDB (The Movie Database) Dataset:** Contains detailed movie metadata, including cast, crew, genres, keywords, release dates, revenue, and more. It also has user ratings and reviews, although these might be less dense than in MovieLens datasets

7. Data Preprocessing:

1. **Data Collection:** Gathering user data (ratings, reviews, watch history) and movie data (attributes, genres, directors).
2. **Data Cleaning:** Handling missing values, removing duplicates, and correcting errors.
3. **Data Transformation:** Converting data into suitable formats (e.g., numerical encoding for genres).
4. **Feature Engineering:** Extracting relevant features (e.g., genre, director, release year) from movie data.
5. **Data Normalization:** Scaling ratings and other numerical features to ensure consistency.

Effective data preprocessing enables accurate and personalized movie recommendations.

8.Exploratory Data Analysis (EDA)

1. User behavior analysis: Examining user ratings, watch history, and preferences.
2. Movie attribute analysis: Investigating genre, director, release year, and other film characteristics.
3. Rating distribution analysis: Understanding user rating patterns and trends.
4. Correlation analysis: Identifying relationships between user preferences and movie attributes.
5. Visualization: Using plots and charts to illustrate findings and patterns.

EDA helps:

1. Understand user preferences: Informing personalized recommendations.
2. Identify trends: Revealing popular genres, directors, or movies.
3. Improve model development: Guiding feature engineering and algorithm selection.

By applying EDA, the AI-driven movie recommendation system can provide more accurate and relevant suggestions.

9. Feature Engineering

For Users (The Movie Watchers):

- What they like: Genres they usually watch (like action, comedy, romance), actors or directors they enjoy.

- How they behave: Movies they've rated highly, movies they've watched all the way through, when they usually watch movies.
- Who they are (simply): Maybe their age group or general location (though we already know it's Chennai!).

For Movies:

- What they are: The genre (action, comedy, romance), who's in them (actors, directors), when they came out.
- What others think: The average rating other users gave it, how many people have watched it.
- What they're about (in short): Keywords or a brief description of the story.

10. Model Building

Model building for movie recommendations is like teaching a computer to guess which movies you'll like.

We do this by:

1. Choosing a method: Like asking similar people what they liked (Collaborative Filtering) or looking at what kind of movies you've liked before (Content-Based). Often, we use a mix of both (Hybrid).
2. Feeding it info (our features): Telling the computer about you (what you watch, rate) and the movies (genre, actors).
3. Letting it learn: The computer finds patterns in this info to make predictions.
4. Checking if it's good: We test its guesses on movies it hasn't seen you watch yet.

11. Model Evaluation

Model evaluation is like giving your movie recommendation system a test to see how good it is at suggesting movies you'll like.

We look at things like:

- Did it recommend good movies? (Precision - out of the top suggestions, how many were relevant?)
- Did it find most of the good movies? (Recall - out of all the movies you might like, how many did it suggest?)
- Did it rank the best movies highest? (NDCG - did the top suggestions really match your taste?)

12. Deployment

- Deploy using a free platform:
 - Streamlit Cloud: Ideal for interactive web applications. You can host a userfriendly interface where users input preferences, and recommendations dynamically appear.
 - Gradio + Hugging Face Spaces: Best for quick AI model hosting with minimal setup. Users can test different matchmaking parameters via sliders or text input.
 - Flask API on Render/Deta: If your recommendation system requires backend API integration, Flask can help serve recommendations via endpoints.

Ensure your deployment steps include:

- Method Used: Specify whether you used Streamlit, Flask, or Gradio.
- Public Link: Provide the hosted model/app link for access.
- UI Screenshot: Show how users interact with the system.

- Sample Prediction Output: Display example matches and movie suggestions based on user inputs.

13. Source Code

```
import pandas as pd

try:
    df = pd.read_csv('english_movies.csv')
    display(df.head())
    print(df.shape)
except FileNotFoundError:
    print("Error: 'english_movies.csv' not found. Please ensure the file exists in the current directory or provide the correct path.")
    df = None # Set df to None to indicate failure
except pd.errors.ParserError:
    print("Error: Could not parse the CSV file. Please check the file format.")
    df = None
except Exception as e:
    print(f"An unexpected error occurred: {e}")
    df = None
```

```
# Data Shape and Info
print("Shape of the DataFrame:", df.shape)
print("\nInfo:")
df.info()

# Descriptive Statistics
print("\nDescriptive Statistics for Numerical Features:")
print(df.describe())

# Missing Value Analysis
print("\nMissing Value Analysis:")
missing_values = df.isnull().sum()
missing_percentage = (missing_values / len(df)) * 100
print(pd.DataFrame({'Missing Values': missing_values, 'Percentage': missing_percentage}))

# Unique Value Counts
print("\nUnique Value Counts for Categorical Features:")
for col in ['genres', 'release_date']:
    print(f"\nColumn: {col}")
    print(df[col].value_counts())

# Data Type Examination
print("\nData Type Examination:")
print(df.dtypes)

# Check if release_date can be converted to datetime
try:
    df['release_date'] = pd.to_datetime(df['release_date'])
    print("\n'release_date' successfully converted to datetime.")
except ValueError as e:
    print(f"\nError converting 'release_date' to datetime: {e}")

# Correlation Analysis
print("\nCorrelation between Numerical Features:")
numerical_features = ['popularity', 'vote_average', 'vote_count']
print(df[numerical_features].corr())
```



```
# Fill missing values
df['overview'].fillna('', inplace=True)
df['genres'].fillna('Unknown', inplace=True)

# Drop rows with still missing 'release_date'
df.dropna(subset=['release_date'], inplace=True)

# Remove duplicate rows
df.drop_duplicates(inplace=True, keep='first')

# Verify cleaning
print("Missing values after cleaning:")
print(df.isnull().sum())

print("\nNumber of rows after cleaning:", len(df))
```

```
# Fill missing values
df['overview'] = df['overview'].fillna('')
df['genres'] = df['genres'].fillna('Unknown')

# Drop rows with still missing 'release_date'
df.dropna(subset=['release_date'], inplace=True)

# Remove duplicate rows
df.drop_duplicates(inplace=True, keep='first')

# Verify cleaning
print("Missing values after cleaning:")
print(df.isnull().sum())

print("\nNumber of rows after cleaning:", len(df))
```

```
# Analyze the distribution of movie genres
genre_counts = df['genres'].str.split(',').explode().value_counts()
print("Genre Distribution:\n", genre_counts)

# Explore the relationship between movie popularity and other numerical features
correlation_matrix = df[['popularity', 'vote_average', 'vote_count']].corr()
print("\nCorrelation Matrix:\n", correlation_matrix)

# Investigate temporal trends
df['release_year'] = df['release_date'].dt.year
yearly_release_counts = df.groupby('release_year')['title'].count()
yearly_avg_popularity = df.groupby('release_year')['popularity'].mean()
print("\nYearly Release Counts:\n", yearly_release_counts)
print("\nYearly Average Popularity:\n", yearly_avg_popularity)

# Examine the relationship between genres and numerical features
# (This is a complex analysis and might require further steps, depending on the desired level of detail)
# For now, calculate average popularity per genre
genre_popularity = df.copy()
genre_popularity['genres_list'] = genre_popularity['genres'].str.split(',')
genre_popularity = genre_popularity.explode('genres_list')
avg_popularity_by_genre = genre_popularity.groupby('genres_list')['popularity'].mean()
print("\nAverage Popularity by Genre:\n", avg_popularity_by_genre)
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Bar chart visualizing the distribution of movie genres
plt.figure(figsize=(12, 6))
genre_counts = df['genres'].str.split(',').explode().value_counts()
genre_counts.sort_values(ascending=False).plot(kind='bar', color='skyblue')
plt.title('Distribution of Movie Genres')
plt.xlabel('Genre')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# 2. Scatter plot matrix
plt.figure(figsize=(10, 8))
sns.pairplot(df[['popularity', 'vote_average', 'vote_count']], kind='scatter', diag_kind='kde')
plt.suptitle('Scatter Plot Matrix of Popularity, Vote Average, and Vote Count')
plt.show()

# 3. Line plot illustrating the trends in yearly release counts over time.
plt.figure(figsize=(10, 6))
yearly_release_counts = df.groupby('release_year')['title'].count()
plt.plot(yearly_release_counts.index, yearly_release_counts.values, marker='o', linestyle='--', color='green')
plt.title('Yearly Release Counts Over Time')
plt.xlabel('Release Year')
plt.ylabel('Number of Releases')
plt.grid(True)
plt.show()

# 4. Line plot showing the yearly average popularity of movies over time.
plt.figure(figsize=(10, 6))
yearly_avg_popularity = df.groupby('release_year')['popularity'].mean()
plt.plot(yearly_avg_popularity.index, yearly_avg_popularity.values, marker='o', linestyle='--', color='orange')
plt.title('Yearly Average Popularity Over Time')
plt.xlabel('Release Year')
plt.ylabel('Average Popularity')
plt.grid(True)
plt.show()

# 5. Bar chart visualizing the average popularity for each genre.
plt.figure(figsize=(14, 6))
genre_popularity = df.copy()
genre_popularity['genres_list'] = genre_popularity['genres'].str.split(',')
genre_popularity = genre_popularity.explode('genres_list')
avg_popularity_by_genre = genre_popularity.groupby('genres_list')['popularity'].mean().sort_values(ascending=False).head(20)
avg_popularity_by_genre.plot(kind='bar', color='purple')
plt.title('Average Popularity for Each Genre (Top 20)')
plt.xlabel('Genre')
plt.ylabel('Average Popularity')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

14. Future scope





Improved Personalization: Integrating deep learning-based recommendation models for more accurate matchmaking.

- Multi-Modal Inputs: Allowing users to provide voice or image-based preferences to refine movie suggestions.

- Scalability: Deploying on cloud infrastructure to handle larger datasets and real-time user interactions.

15.Team Members and Roles

- **Sufiya Firdouse.A**– Led project structuring and workflow efficiency.
- **Roshni.B** – Focused on dataset preprocessing and refinement.
- **Pooja Sri.S** – Developed collaborative filtering techniques for recommendations.
- **Mudabbira Fathima.S**– Explored hybrid modelling approaches.
- **Sandhiya.S**– Managed deployment strategies and testing.
- **Monisha.J** – Designed the user interface for an engaging experience.

 Phase 3-Output NM.docx	Add files via upload	8 minutes ago
 README.md	README.md	37 minutes ago
 Source_code_proj.ipynb	Add files via upload	46 minutes ago
 english_movies.csv	Add files via upload	3 hours ago