Delivering Personalise Movie Recommendation With AI Driven Match Making System

Student Name: Arfath I

Register Number: 410623104008

Institution: Dhaanish Ahmed College Of Engineering

Department: Computer Science And Engineering

Date of Submission: 7.5.2025

1.Problem Statement

As the volume of digital content continues to grow across streaming platforms, users face increasing difficulty in selecting movies that align with their personal tastes and preferences. Traditional recommendation systems often depend on static algorithms, such as collaborative filtering or general popularity trends, which fail to capture the nuanced preferences of individual users. These systems typically overlook factors like mood, genre flexibility, evolving interests, and user context, resulting in recommendations that feel generic or irrelevant.

Moreover, users today expect smarter, more intuitive systems that can understand their unique viewing patterns and offer suggestions that feel tailored and timely. The lack of dynamic, intelligent personalization leads to reduced user satisfaction and can negatively impact engagement and platform loyalty.

This project addresses this challenge by developing an AI-driven matchmaking system designed to deliver highly personalized movie recommendations. By leveraging machine learning, user profiling, and behavioral analytics, the system aims to match users with movies in a way that feels meaningful, accurate, and adaptive to changing preferences, ultimately enhancing the overall user experience.

2.Objectives of the Project

1. Develop a Personalized Recommendation Engine:

• Design and implement a system that recommends movies based on users' preferences, watch history, and behavior patterns using AI techniques such as collaborative filtering, content-based filtering, or hybrid models.

2. Enhance User Engagement through Matchmaking:

• Integrate a matchmaking mechanism that pairs users with similar movie tastes to encourage shared viewing experiences, discussions, or community building.

3. Improve Recommendation Accuracy with Machine Learning: • Use machine learning models to continuously learn from user feedback (likes, ratings, skips) to refine and improve recommendation accuracy over time.

4. Enable Dynamic User Profiling:

- Create adaptive user profiles that evolve based on real-time interactions, mood detection, or contextual data (e.g., time of day, device used).

5. Ensure Scalable and Real-Time Recommendation Delivery:

- Design the system architecture to support real-time recommendation delivery and scalability for a growing user base.

6. Incorporate Explainable AI (XAI):

- Provide users with understandable explanations for recommendations to build trust and transparency in the AI system.

7. Measure System Performance and User Satisfaction:

- Establish KPIs such as click-through rate (CTR), watch time, and user satisfaction to evaluate the system's effectiveness and guide iterative improvements.

3.Scope of the Project

The scope of this project is to design, develop, and deploy an intelligent system that not only recommends movies tailored to individual user preferences but also introduces a novel social dimension through an AIdriven matchmaking mechanism. The system aims to transform traditional recommendation platforms by combining personalization with social engagement, thereby enhancing user experience, content discovery, and community building.

1. User Profiling and Behavior Tracking

The system will begin by collecting and analyzing user data to build comprehensive user profiles.

This includes:

- Historical data such as previously watched movies, genres frequently explored, ratings given, and watch durations.
- Explicit preferences provided during sign-up or via periodic surveys (e.g., favorite genres, disliked themes, favorite actors/directors).
- Implicit behavioral data like browsing patterns, click-through rates, and time spent on movie details.
- Demographic information (age, gender, location) and psychographic indicators (mood preferences, emotional engagement). These profiles serve as the foundation for both recommendation and matchmaking systems.

2. Personalized Recommendation Engine

At the core of the system will be a recommendation engine using machine learning techniques to provide highly accurate and relevant movie suggestions.

The recommendation engine will employ:

- Collaborative Filtering: Leveraging the preferences of similar users to suggest new content.
- Content-Based Filtering: Analyzing movie features (genre, cast, keywords) in relation to the user's past preferences.
- Hybrid Models: Combining both collaborative and content-based approaches to improve accuracy and mitigate cold-start problems.
- Context-Aware Recommendations: Considering contextual variables like time of day, user mood, and device used. The engine will adapt

over time, using continuous learning to refine its output as user data grows.

3. AI-Driven Matchmaking System

This unique component will introduce social interactivity by connecting users with similar or complementary movie preferences.

It includes:

- Similarity Scoring Algorithms: Using clustering or vector similarity (e.g., cosine similarity in embedding space) to find users with overlapping movie tastes.
- Dynamic Match Suggestions: Recommending potential movie buddies or discussion partners based on real-time data and evolving preferences.
- Group Watch and Interaction Features: Allowing matched users to watch movies together (virtually), chat, or leave shared reviews.
- Custom Filters: Letting users define parameters for matchmaking such as age range, preferred genre match percentage, or geographic proximity.

4. User Interface and Experience Design (UI/UX)

The platform will feature a user-friendly, engaging, and intuitive interface accessible via web and mobile platforms.

The UI will include: • Personalized home screens with dynamic movie suggestions.

- A matchmaking dashboard showing suggested matches and movie compatibility scores. • Interactive features like user reviews, likes/dislikes, watch parties, and discussion boards.

• Settings for privacy, matchmaking preferences, and notification control.

5. Backend Infrastructure and System Architecture

The backend will handle data storage, processing, and real-time analytics.

Core aspects include:

• A scalable database (e.g., NoSQL or graph-based) to manage user and movie metadata. • APIs to facilitate data exchange between components (frontend, recommendation engine, matchmaking logic).

• Integration of cloud services or microservices architecture to ensure reliability and scalability.

• Implementation of secure authentication and data protection mechanisms to comply with privacy regulations such as GDPR or CCPA.

6. Evaluation and Continuous Improvement

To ensure system effectiveness and user satisfaction, ongoing evaluation will be built into the platform:

• Performance Metrics: Precision, recall, F1 score for recommendations; engagement and retention metrics for matchmaking.

• A/B Testing: Comparing algorithm variants or UI changes to find the most effective configurations.

• User Feedback: Surveys, reviews, and behavioral analytics to assess the quality of recommendations and matchmaking.

• Machine Learning Feedback Loop: Continuously retraining models based on new data and outcomes.

4.Data Sources

1.Public Movie Datasets

MovieLens Dataset (GroupLens) Contains millions of movie ratings, user IDs, timestamps, and movie metadata. Widely used for training and testing recommendation systems. IMDb Dataset (Internet Movie Database) Offers extensive movie details: cast, director, genre, plot summary, ratings, user reviews. IMDb offers datasets via their official IMDb datasets page. The Movie Database (TMDB) API Provides realtime access to movie metadata, images, trailers, reviews.

2. User Interaction Data (Behavioral Data)

Explicit Feedback Ratings given by users (e.g., 1–5 stars). Written reviews or comments. Implicit Feedback Viewing history: movies watched fully or partially. Browsing behavior: search queries, time spent per movie page, click-throughs.

3. Social Media and External Signals

Twitter/X, Reddit, YouTube (optional enhancement) Sentiment analysis on tweets, Reddit posts, or YouTube comments about movies can refine recommendations. Tools like Tweepy (Python) can be used to extract Twitter data. Google Trends Identifies trending movies or genres regionally or globally.

4. User Profile Data

Basic details: Age, Gender, Country. Preferences: Favorite genres, favorite actors, language preference. Watching habits: Preferred time to watch, device used (mobile/TV/laptop).

5. Content-Based Features

Movie Metadata: Genre, cast, director, synopsis. Tags or keywords describing the movie's themes. Poster images and trailers (for deep learning-based recommendations like using CNNs for visual content).

5.High-Level Methodology

- **Data Collection** –
  - Sources: Movie databases (e.g., IMDb, TMDb), user ratings (e.g., MovieLens), streaming platforms (if available).
  - Data Types:
  - User profiles (age, gender, location, preferences)
  - Movie metadata (genre, cast, director, year)
  - User-movie interactions (ratings, views, likes, reviews)

- **Exploratory Data Analysis (EDA)** –
  - Cleaning and normalizing user and movie data
  - Handling missing values

- Encoding categorical variables (e.g., genres, languages) • Feature engineering (e.g., calculating user similarity, genre preference vectors)

- **Feature Engineering** –
  - User Features: Preferences derived from viewing patterns and ratings.
  - Item Features: Embeddings for genres, actors, and themes. • Contextual Features: Time of day, device used, location (if applicable).

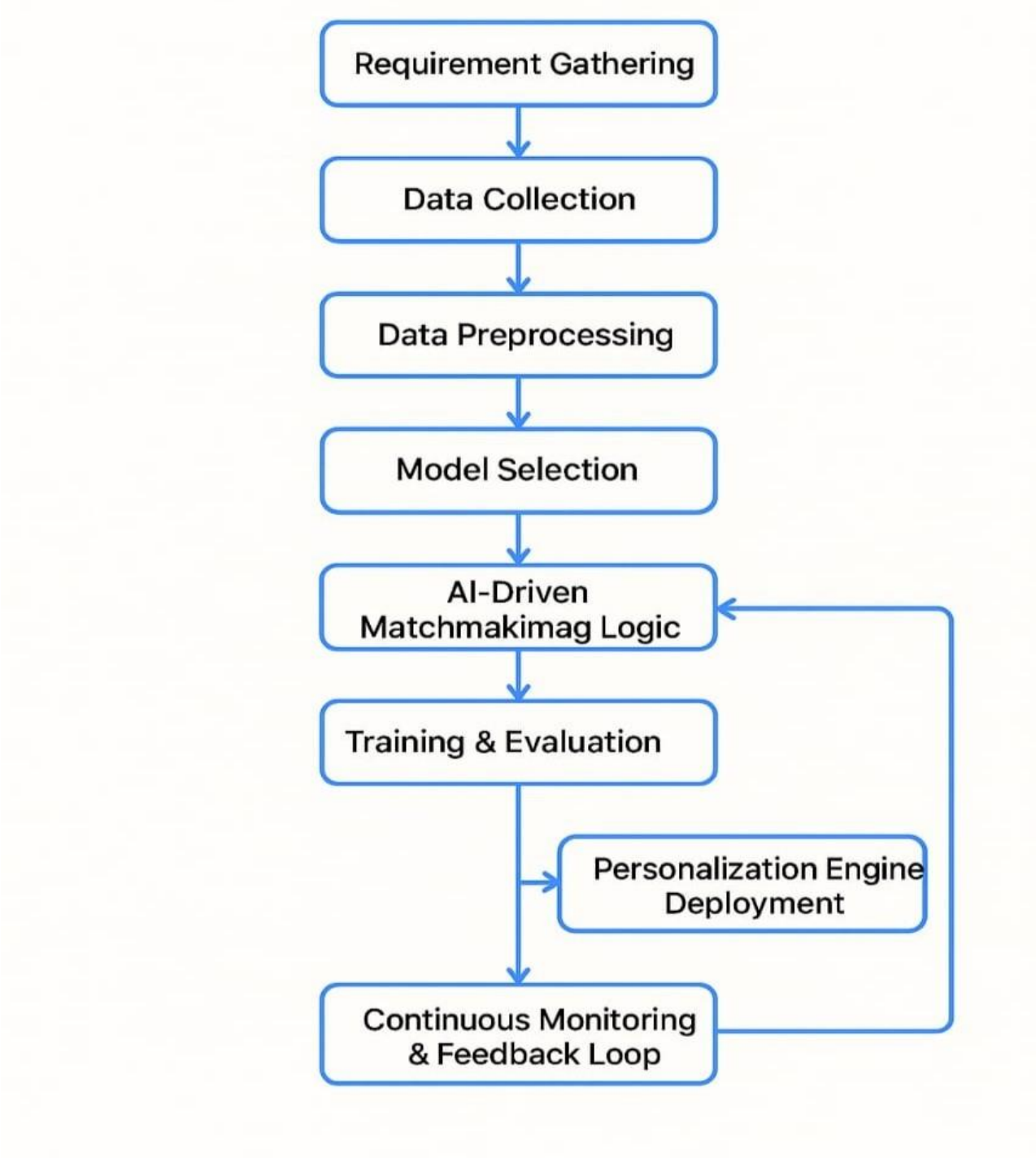
- **Model Building** –
  - Collaborative Filtering (user-based or item-based)
  - Content-Based Filtering (based on movie metadata) • Hybrid Models (combine collaborative and content-based filtering)
  - Deep Learning Models (e.g., neural collaborative filtering, embeddings)
  - Reinforcement Learning (to improve recommendations based on real-time feedback)

- **Model Evaluation** –
  - Offline Metrics: RMSE, MAE, Precision@K, Recall@K, NDCG.
  - Online Metrics: Click-through rate (CTR), watch time, user engagement.

- A/B Testing: Validate model improvements in live environments

- **Deployment** –
  - API Development: Expose recommendation engine via REST or GraphQL APIs.
  - Real-time Inference: Use caching and scalable serving architectures (e.g., using Redis, Kubernetes).
  - User Interface Integration: Embed recommendations in app or website interfaces.
  - User Features: Preferences derived from viewing patterns and ratings.
  - Item Features: Embeddings for genres, actors, and themes.
  - Contextual Features: Time of day, device used, location (if applicable).

Flow Chart For Working Process:



6.Tools and Technologies

1. Data Collection & Storage

Data Sources You need rich data about both movies and users:

- Movie metadata: Titles, genres, cast, crew, release dates, user reviews, plot summaries.
- Use APIs like TMDb, IMDb, or OMDb.
- User data: Watch history, likes/dislikes, ratings, searches, favorites, time spent watching, etc. Storage Options
- Relational Databases (SQL):
- Use PostgreSQL or MySQL to store structured data like user profiles or movie metadata.
- NoSQL Databases:
- Use MongoDB or Cassandra for scalable and flexible schema handling.
- Cloud Storage:
- Store raw datasets or logs in AWS S3, Google Cloud Storage, or Azure Blob Storage for later processing.

2. Data Processing & Feature Engineering

ETL Pipelines

- Use Apache Airflow, Luigi, or Prefect to automate:
- Data ingestion (from APIs, logs)
- Cleaning (removing nulls, duplicates)
- Transformation (normalization, encoding)
- Loading into data warehouses or model training pipelines Feature Engineering
- User features: Age, preferences, genre affinity, interaction patterns
- Movie features: Genre vectorization (one-hot or embeddings), sentiment score of plot
- Interaction features: Time of day watched, time spent, watch frequency Use pandas, NumPy, scikit-learn, or PySpark for efficient feature generation.

3. Recommendation Algorithms

Collaborative Filtering

- Based on the idea that users who liked similar movies in the past will like similar ones in the future.

- Algorithms:
- Matrix Factorization (SVD): Reduces user-item matrix into latent factors
- ALS (Alternating Least Squares): Works well with implicit feedback (clicks, views)
- Libraries: surprise, implicit, lightfm Content-Based Filtering
- Recommends movies similar to what a user liked based on metadata (genre, actors, descriptions).

- Uses TF-IDF, cosine similarity, or word embeddings to represent movie descriptions. Hybrid Recommendation Systems
- Combine collaborative and content-based filtering.
- Example: Use collaborative filtering for candidate generation and content-based filtering to rank results. Deep Learning-Based Models
- Neural Collaborative Filtering (NCF): Learns non-linear user-item interactions using deep nets.
- Autoencoders: Compress and reconstruct user preferences.
- Transformers/BERT: Understand context in plot summaries or reviews for personalized recommendations.
- Libraries: TensorFlow, PyTorch, Hugging Face Transformers

4. AI-Driven Matchmaking & Personalization

User Profiling

• Dynamically build user personas using:

• Browsing behavior •
Genre affinity over time

• NLP on user reviews Clustering for Group Recommendations

• Group users with similar behavior using K-means, DBSCAN, or GMM.

• Useful for collaborative filtering cold starts. Knowledge Graphs

• Model complex relationships (e.g., Actor A and Director B collaborated on Movie X).

• Use Neo4j or RDF/SPARQL to make intelligent recommendations based on relationship strength.

5. Model Training & Evaluation

Training

• Batch or real-time model training using:

- scikit-learn for classic ML
- TensorFlow/PyTorch for deep learning
- Spark MLlib for large-scale model training Evaluation Metrics
- RMSE, MAE: For regression-based rating prediction
- Precision@K, Recall@K, NDCG: For ranking-based evaluation
- A/B Testing: Compare recommendation models live

6. Deployment

Serving Models

- Use REST APIs with FastAPI or Flask to serve real-time recommendations.

- For scalability, use:
- Docker containers
- Kubernetes for orchestration
- MLflow, Seldon, or TensorFlow Serving for model management Hosting

- Cloud platforms like AWS (EC2, SageMaker), GCP (Vertex AI), or Azure ML

Code :

```
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity


# Sample movie dataset

data = {

    'movie_id': [1, 2, 3, 4, 5],

    'title': ['Inception', 'The Matrix', 'Interstellar', 'The Notebook', 'Titanic'],

    'description': [

        "A thief who steals corporate secrets through the use of dream-sharing technology.",

        "A hacker discovers reality is a simulation and joins a rebellion.",

        "A team travels through a wormhole in space in an attempt to ensure humanity's survival.",
```

```
        "A romantic drama about young love and the power of memory.",
        "A love story unfolds on the ill-fated RMS Titanic."
    ],
    'genres': [
        'Sci-Fi, Thriller',
        'Action, Sci-Fi',
        'Adventure, Drama, Sci-Fi',
        'Romance, Drama',
        'Romance, Drama'
    ]
}

movies = pd.DataFrame(data)

# Simulated user profile (likes Sci-Fi, thrillers, and space)
user_profile = {
    'liked_genres': ['Sci-Fi', 'Thriller', 'Adventure'],
```

```
    'preferred_keywords': 'dream-sharing technology space rebellion
survival simulation'

}


# Combine genres and description for feature vectorization

movies['features'] = movies['genres'] + ' ' + movies['description']


# Append user profile as a "pseudo movie" for similarity comparison

user_features = ' '.join(user_profile['liked_genres']) + ' ' +
user_profile['preferred_keywords']

all_features = movies['features'].tolist() + [user_features]


# Vectorize using TF-IDF

vectorizer = TfidfVectorizer(stop_words='english')

tfidf_matrix = vectorizer.fit_transform(all_features)


# Calculate cosine similarity between user profile and all movies

cos_sim = cosine_similarity(tfidf_matrix[-1], tfidf_matrix[:-1])

similarity_scores = cos_sim.flatten()
```

```
# Rank movies based on similarity

movies['similarity'] = similarity_scores

recommended_movies = movies.sort_values(by='similarity',
ascending=False)


# Show top recommendations

print("Top personalized movie recommendations:\n")

print(recommended_movies[['title', 'genres', 'similarity']].head(3))
```

7.Team Members and Roles

Problem Statement and Objectives – Farviyaz Ali M

Scope Of The Project – Mohammed Yousuf Irfan M

Data Sources – Imran Khan A

High Level Methodology – Arfath I

Tools And Technologies – Irfaan Z