Movie Recommendation System

Goals and Requirements:

Core functionalities:

In the context of the evolving recommendation system, data sources are critical for powering the recommendation logic, enabling personalization, and adapting to new requirements. The system will rely on a variety of data sources, each contributing specific information to improve recommendations, including metadata about movies, user preferences, and feedback.

1. Data integration and preprocessing
2. Recommendation algorithms
3. User profile management
4. System resilience
5. Real time monitoring

Unique features:

These unique features make the recommendation system not only more personalized but also more flexible, adaptive, and resilient. The system is designed to evolve alongside user preferences and provide users with a rich, diverse set of recommendations, including leveraging novel data points like directors. Additionally, by offering real-time performance tracking and fallback mechanisms, the system ensures reliability and responsiveness, making it a robust solution for dynamic and scalable recommendation needs.

Architectural Decisions

- The system uses few microservices like mangoDB to allow better scalability of the system.
- The current system uses synchronous communications for requests handling, database operations and external monitoring calls
- The inference is done on the server-side to ensure reliability and efficient handling of requests.
- In our system we use centralized data storage using MongoDB because decentralized data storage is either not free or doesn't allow overwriting, but for real-time implementation decentralized data storage is preferred to avoid a single point of failure.
- We ensure data privacy in our prototype by hashing users’ passwords stored in the database.

- For model retraining, we use the users' feedback data to update profile clustering which is then used as a part of the recommendation process.

System overview:

Hybrid architecture:
The core system employs a hybrid recommendation model integrating collaborative filtering (via clustering) and content-based filtering (using genre preferences). Upon receiving a user's demographic information (age, gender, occupation) and favorite genres, the system constructs a feature vector from this data. A pre-trained KMeans clustering model then compares this vector to the centroids of existing clusters, which represent predefined user profiles. New users are assigned to the cluster (profile) with the closest centroid, enabling collaborative filtering. Meanwhile, genre-based filtering handles the content-driven aspect of recommendations. This dual approach ensures personalized suggestions by leveraging both user behavior patterns and content preferences. The **content-based filtering** part comes into play when the system recommends specific movies within the cluster.

How Content-Based Filtering is Applied:
1.After assigning the user to a cluster, the system filters movies that match the user's preferred genres.
2. For each movie, the system calculates how well its genres match the user's preferences.
If a movie has genres "Action" and "Crime," and the user likes "Action" and "Adventure," the genre match score will be:
Genre Match Score = (Number of matching genres) / (Total number of preferred genres)
= 1 / 2 = 0.5
Genre Match Score: This is calculated as the proportion of the movie's genres that match the user's preferred genres.
For example: If a movie has 3 genres and 2 of them match the user's preferences, the genre match score is 2/3 ≈ 0.67.
3. The system combines the predictions from collaborative filtering and the genre match score from content-based filtering to give a final score for each movie.

Final Score = Predicted Rating * (1 + Genre Match Score)
where,
Predicted Rating: This is the output of the pre-trained model (dot product between embeddings).
Genre Match Score: This is calculated based on how well the movie's genres match the user's preferred genres.

The user feedback collected affects the system by increasing or decreasing the weights which updates the recommendation.

Database :

We have used MongoDB as our main database to store user data due to following reasons:

Flexible Schema: Stores dynamic user data (preferences, feedback, session genres) in JSON-like documents, adapting easily to evolving requirements

Scalability: MongoDB Atlas (cloud) handles distributed data and scales seamlessly for growing user bases.

Real-Time Updates: Efficiently updates user profiles (e.g., session_genres, feedback) and cluster assignments (profile_id) without rigid table structures.

Model Retraining:

When the user gives feedback, the model weights are updated and based on them the user remains in one cluster or is assigned a different cluster. When the user gives a positive feedback the weights are increased by 20 % and when the user gives a negative feedback, the weights are decreased by 30% the cluster is switched when the weights are less than 70%. Additionally, when the feedback accuracy drops below a threshold (60%) the user is prompted to update the preferred genre

Adaptive Features and Model enhancements :

Temporary session genres vs permanent storage

Accuracy-triggered genre adjustment prompts

Explicit genre change capability

Session validation before each request

Automatic logout on invalid sessions

Feedback history stored in MongoDB so movies are not repeated after feedback

User Experience:

Registration: A new user visits the registration page and enters details like username, password, age, gender, occupation, and selects favorite movie genres (e.g., Action, Comedy). The system validates inputs (e.g., age must be 1–100, genres must be selected) and ensures the username is unique. After validation, the user is assigned to a profile cluster based on their demographics and genre choices. Their data is securely stored in MongoDB with hashed passwords.

Login: The user logs in with their credentials. If valid, they are redirected to the recommendation page. The session initializes with their stored preferences and profile ID.

Movie Recommendations: The system displays one movie at a time (e.g., "Inception (Action/Sci-Fi)"). Recommendations are generated by filtering movies matching the user's current genre preferences, using a neural network to predict interest based on their cluster (profile_id). Prioritizing movies not previously recommended.

Feedback Loop: The user clicks "Yes" (liked) or "No" (disliked). Feedback is recorded in the session and MongoDB. Genre change is triggered when accuracy is lower than 60% however user can change the genre whenever they want

Logout/Completion: If the user has changed their genre at any point during recommendation, the system asks the user before logging out whether they want to store their genre or not if yes then the preferred genre is overwritten with new genre and if no then preferred genre is kept same and new genre is deleted.

Security & Monitoring: Passwords are hashed, and sessions expire securely. System health (CPU, memory) is monitored in real-time via a background thread.

The system consists of long term and session based profiles. The long term profile is the one that is created during the registration process. When a user creates an account the genre is stored in "Preferred " and is used to generate recommendations. At any point if the genre is changed (due to low accuracy or manually by the user) the new genre is added into the 'session genre ' table and the preferred genre is kept as it is. After the recommendation process is complete and the user logs out, the system checks if the preferred genre table and the session genre table has different genres. If yes, then it prompts the user if they want to save the new genre (session genre) or discard it. If the user saves the new genre the preferred genre is overwritten by the content of the session genre and the session genre becomes empty.

Example :

User registers with 'action' as their preferred genre and action movies are recommended
The user changed their genre manually to comedy during the recommendation and the system displays 'comedy' movies
A new table called session genre is created in DB which stores the genre 'comedy' and the preferred genre table still has the genre 'action' which is now not used
When the user tries to logout, the system alerts the user to either save 'comedy' or discard it
If the user saves it, the session genre becomes empty and preferred genre now becomes 'comedy'
If the user does not saves it, the session genre is discarded and the preferred genre is kept same (action)